ISECON '88

INFORMATION SYSTEMS EDUCATION CONFERENCE

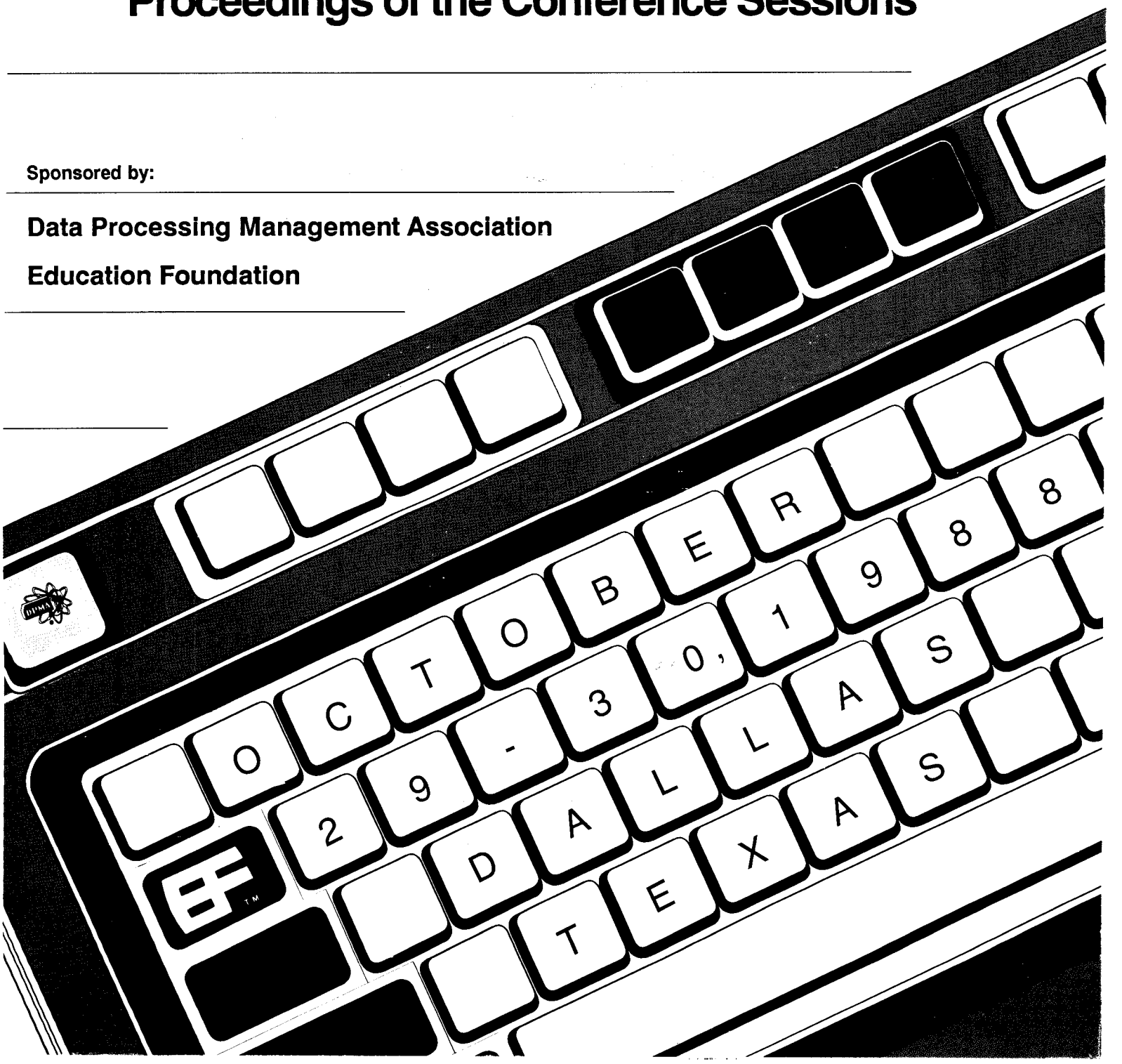
Seventh Annual

Proceedings of the Conference Sessions

Sponsored by:

Data Processing Management Association

Education Foundation



ISECON '88

Seventh Annual

INFORMATION SYSTEMS EDUCATION CONFERENCE

Proceedings of the
Conference Sessions

October 29-30, 1988
Dallas, Texas

Sponsored by:
Data Processing Management Association
Education Foundation

ACKNOWLEDGEMENTS

Corporate Support

A $20,000 grant from International Business Machines Corporation (IBM) has enabled the DPMA Education Foundation to offer reasonable registration fees for ISECON '88.

IBM has followed with great interest the ISECON conferences over the past several years. A company spokesman for IBM said, "This is a way of showing that IBM supports DPMA's goals for preparing our industry with a better educated and more efficient work force."

A total of 89 papers were submitted to ISECON '88. Of those submitted, 48 were accepted. Each paper was reviewed by at least two referees who submitted individual paper reviews, detailed comments, and ranking of the papers reviewed.

WELCOME TO ISECON '88

"The Key to information systems education is information exchange." And the exchange of information -- about curriculum, about information systems, about the industry -- is the key to ISECON '88.

To facilitate this exchange, ISECON '88 provides opportunities for formal presentations, informal discussions, demonstrations, and workshops. You enhance the exchange by responding to the presentations, and actively participating in the various informal sessions. The more you participate, the more wide-spread and valuable will be our informaton exchange.

Information systems educators and industry trainers face a multi-faceted challenge: how to continue to teach the necessary skills for today's information systems environment and yet prepare for the vast number of technological changes already forecast for the not-too-distant future. How do we prepare technologcally competent professionals who also have the interpersonal and business-related skills required in our rapidly evolving information industry?

ISECON '88 will not, of course, provide all the answers! But we hope you wil find direction, scope, ideas, and resources which will help in your response to the challenge.

Welcome to ISECON '88!

Best wishes,

Beverly B. Madron, Ph.D., CDP
President, Board of Regents
DPMA Education Foundation

ISECON '88

Steering Committee

Conference Chairperson

Ronald J. Kizior
Management Science Department
Loyola University of Chicago

Program Chairperson

Robert Zant
MIS/MSC Department
University of Alabama, Huntsville

Facilities and Local Arrangements Chairperson

Robert H. Dunikoski
Graduate School of Management
University of Dallas

Placement Chairperson

R. Wayne Headrick
Department of Business Analysis
Texas A & M University

Exhibits Chairperson

Ilene Siegel Deutsch
Information Systems Department
Pace University

Marketing / Industry Liaison Chairperson

Stan Piercefield
Information and Telecommunications Services
Public Service Indiana

Ex-Officio

Kathy Brittain White
Department of Information Systems
University of North Carolina, Greensboro

ISECON '88 Reviewers

Ageloff, Roy
University of Rhode Island

Ameen, David
Virginia Commonwealth University

Beccue, Barbara
Illinois State University

Benham, Harry C.
The University of Oklahoma

Blanton, Ellis
University of South Florida

Callahan, Elias R., Jr.
Mississippi State University

Chrisman, Carol
Illinois State University

Christenson, Edward A.
California State University - Sacramento

Cook, Janet
Illinois State University

Dekleva, Sasa
DePaul University

Discenza, Richard
University of Colorado at Colorado Springs

Englander, Irvin S.
Bentley College

Fowler, George
Texas A & M University

Gibson, Michael L.
Auburn University

Hodge, Bart
Virginia Commonwealth University

Hodgson, John S.
University of South Florida

Hoeke, Marilynn Carol
The University of North Texas

Hughes, Cary
The University of North Texas

Hulme, Fred
Baylor University

Kasper, George M.
Texas Tech University

Khosrowpour, Mehdi
The Pennsylvania State University at Harrisburg

King, James R., Jr.
Baylor University

Luce, Thomas G.
Ohio University

MacKay, Jane
Texas Christian University

McGrath, Roger
University of South Florida

McIntyre, Scott
University of Colorado at Colorado Springs

Milligan, Pati
Baylor University

Morgan, George W.
Southwest Texas State University

Nagarajan, N.
Central Connecticut State University

Nitterhouse, Denise
DePaul University

Ramsower, Reagan M.
Baylor University

Richards, Thomas C.
The University of North Texas

Saunders, Carol S.
Texas Christian University

Solano, Judith L.
University of North Florida

Sutherland, David Earl
Ohio University

Swanson, Wayne
University of South Florida

Willis, G. W.
Baylor University

Wynne, Albert J.
Virginia Commonwealth University

TABLE OF CONTENTS

TRACK 2 INNOVATIVE TEACHING METHODS

TRACK 4 RESEARCH IN TEACHING

TRACK 5 GRADUATE MIS

UNEXPECTED SIDE EFFECTS FROM FINE TUNING THE CURRICULUM

DR. JACOB GERLACH
UNIVERSITY OF WISCONSIN-WHITEWATER
800 W. MAIN STREET
WHITEWATER, WI 53190

Abstract:

This paper deals with curriculum changes, student course load and student programming maturity. Student maturity is often considered in terms of courses taken; these findings suggest that maturity may be more specific and in fact, may be traced to a single program or project.

INTRODUCTION

At the University of Wisconsin-Whitewater we are constantly struggling to keep our program current and meaningful. We recently went through a curriculum revision where we made some changes in the programming segment of the curriculum. Under the old curriculum every student was required to take Introduction to Programming, a course taught at Whitewater by the Mathematics and Computer Science Department. The student then took three MIS programming courses. These three courses were:

Concepts of Programming using PL/I
Data Structures and File Structures
COBOL and Database.

The Introduction to Programming course is used by many disciplines and it is not ours to change; but some changes had to be made in the other three courses. Over time the courses had expanded to the point that each tried to cover entirely too much material; further, we felt the need to expand the database portion to at least a full semester.

To make the material covered in each course more manageable and to allow a full semester for database required the addition of a new course. To compensate for the additional three hours of required courses, we reduced the number of required elective credits. The new curriculum has:

Concepts of Programming

using PL/I (slightly revised by moving a few topics into the new course and reducing the required number of programs from nine to seven.)

Data Structures

(the File Structures material was moved to the new course, and the number of programming assignments was reduced.)

COBOL and File Structures

(a new course, introduced to cover all of the necessary material removed from the other courses.)

Database.

To avoid lengthening the overall time required to complete the major, it was decided that students could take the COBOL and File Structures course concurrently with the Data Structures course.

OBSERVATIONS

The new courses were phased in to ease disruption to the students. I had the privilege of teaching both the last section of COBOL and Database under the old curriculum and the first section of COBOL and File Structures under the new curriculum. The students in these courses had either two or three semesters of programming, so the main task in teaching COBOL was syntax. In advanced classes it has been my teaching style to be more of a manager and make the students use manuals for most of their questions. I used the management style in both classes, but I was not prepared for the differences between the classes. The COBOL and Database class accepted everything readily as they had in past semesters and progressed nicely. The COBOL and File Structures class floundered about hopelessly almost from the start. I immediately suspected that I had weaker students in the COBOL and File Structures course. I asked the instructor that had taught the first group and was teaching the second group in Data Structures if he noticed a difference in the ability level and he said that the current students were at least as good as the previous class in his course.

In order to find out exactly what the difference was we surveyed students in each class as well as graduating seniors and faculty. There appeared to be two main problems for the students in the COBOL and File Structures class:

(1) Most of these students had been taking only one programming course per semester; under the old curriculum that would involve developing roughly nine programs. They now had two programming courses at the same time and, while each course had fewer programs, there were still more total programs to be developed.

(2) These students lack "programming maturity". For some time our Data Structures course has included a large simulation program that requires the careful management of many queues and/or stacks using linked lists. This program is usually about 1000 lines of code and is very messy. It is the first assignment where the program itself prevents students from skipping the design stage and going directly to coding. Both students and the faculty notice a big difference in the students' programming abilities after they have finished the simulation program.

While the first problem of time and number of programs is serious, in the past we have had some students take two programming courses in the same semester and do well, so that could not explain all of the problem. The second problem seemed to be at the heart of the difficulties and had been totally unexpected. We have always looked at the courses and their content when establishing prerequisites. Maturity was considered important; however, the measuring device for maturity was content of courses taken and programs written. It did not occur to us that a maturity watershed point in a student's collegiate MIS career could be traced to the writing of a single program.

Our curriculum is under constant revision; and we are making changes to offset the problems. First, we are advising students that two programming courses in the same semester is very difficult and that they should not undertake it lightly. Second, we are changing the way the COBOL and File Structures course is being taught. We are reserving the managerial teaching style and extensive manual use for the Database course.

Finally the faculty is now painfully aware of the maturity level of the students and, in particular, the maturity gained by doing certain specific programs or projects. We would advise anyone who is changing their curriculum to look beyond the courses to the specific programs or projects completed when making maturity judgments.

DEVELOPMENT OF MIS PROGRAMS IN AN INTERNATIONALLY ORIENTED UNIVERSITY

by
Dr. Robert Trippi
and
Dr. Efraim Turban*

School of Business and Management
United States International University
San Diego, California

ABSTRACT

This paper describes the major considerations governing the design of MIS educational programs for a private university with a unique international ambience. In addition, an overview of the programs themselves is presented.

U.S. INTERNATIONAL UNIVERSITY

United States International University (USIU) warrants the description international in two respects. It has physical plants in the U.S (San Diego), the U.K (London), Mexico (Mexico City), and Kenya (Nairobi), as well as being in the planning stage for new degree programs in Japan (Osaka), Caroline Islands, and Hong Kong. The second aspect is the composition of the student body. About 20% of the total 3500 students study in the overseas campuses. In addition, about 26% of the 2800 in the San Diego campus are from outside the U.S.

The main (San Diego) campus has a distinct international ambience due to a significant portion of the student body being drawn from outside of the U.S. Many of the programs at the main campus have an orientation designed to appeal to and benefit the foreign student. One of the primary perceived missions of the University is to foster international and inter-cultural understanding through contact of individuals engaged in the educational process. The School of Business and Management, School of Education, School of Human Behavior, and School of Performing and Visual Arts constitute the major educational segments of the university. There are also a School of Hospitality Management and a very small School of Engineering. Within the School of Business and Management, which accommodates about 1/3 of the students at the main campus, a relatively large proportion of courses deal with problems unique to international business operations. The areas

*The authors are grateful to Assistant Dean Mink Stavenga for helpful comments in the preparation of this manuscript.

represented include international marketing, international finance, cross-cultural management, strategic management, and international economics. Degrees are offered at the bachelors, masters, and doctoral levels. Indicative of the thrust of the School of Business, the International Business Administration (IBA) specialization is the most popular of the three current doctoral concentrations.

DIVERSITY OF THE STUDENT BODY

The School of Business student body differs considerably in diversity at the various levels. The undergraduate and MBA programs are dominated by foreign students (about 55% and 60%, respectively). Many of the foreign MBA students are employed in the U.S. or on leave from companies in their own countries while undertaking graduate study. The doctoral program appeals most to American students (80%), including those who are residents of and employed by companies in the San Diego area. The greater appeal of the graduate programs to the domestic and employed student markets is reinforced by the offering of such courses almost exclusively at night and on weekends, which being on a quarter system meet for one 5-hour session per week for ten weeks. Undergraduate courses are offered predominantly in the daytime since most undergraduate foreign students are full-time and resident on campus or in the immediate vicinity of the University. In the design of the new MIS programs to satisfy both the undergraduate and graduate markets the differing constituency of the student bodies at these levels was a paramount consideration.

THE STUDENT POPULATION FOR THE MIS PROGRAM

The differing levels of diversity and motivations of undergraduate and graduate students interested in pursuing a degree program in MIS dictates a differentiation of the product offered to each. American employers are generally less interested in undergraduate degrees in MIS than are foreign employers, particularly those in the developing countries. In the U.S. a more technical orientation, as represented by a bachelors degree in computer science, appears to offer greater employment opportunities to the younger new baccalaureate graduate. However, in many of the developing countries, particularly in the Far East, a bachelors degree in MIS is highly marketable. It is estimated by the Assistant Dean for Admissions that 25% of the foreign students applying to the School of Business and Management prefer the bachelor in MIS degree to the BBA, while only 10% of the American students would prefer the BS in MIS. This derives from the generally greater ease with which foreign students can combine MIS skills with typical entry job responsibilities in their own countries.

In contrast, it is the masters degree in MIS that has a greater perceived value to the American student and the foreign student whose goal is employment in the U.S. or overseas with a multi-national corporation. Older and usually currently or previously employed in middle-management positions, a highly technical program is less relevant to job duties and career ladders than a program emphasizing the managerial perspective. The MS in MIS program is thus designed primarily with this type of student in mind.

THE MIS PROGRAMS

There are several distinct functions for our MIS curriculum being offered by the School of Business and Management. First, as with most universities, the offerings provide an understanding of MIS concepts needed

for the support managerial decision making in the other disciplines. Secondly, it offers specific skills in using spreadsheet and other software development and application packages, which are complementing many of the textbooks in the functional areas. This function may actually be an acculturalization process for many international students if the penetration of computer technology is very low in their own countries. However, the difference in the degree of the acculturization aspect between students from industrialized and developing countries is declining. It is required that the students will have a computer literacy course as a part of their general education at the freshman level.

The third primary function of the MIS curriculum is to provide a concentration distinct from the traditional "generalist" orientation required in AACSB accredited business school programs. The MIS degree is designed primarily to attract the foreign undergraduate student who knows at time of application that he/she desires this degree rather than the BBA and the graduate student (American and foreign) who views the MIS degree as more valuable for career advancement than the MBA. In addition, the advanced MIS courses are expected to enroll about 15% of non-MIS majors in the BBA and MBA programs who become interested in learning more about MIS after being exposed to the computer through their course work in the functional areas or through the required MIS courses. Many of these would be currently employed graduate students who discover the usefulness of such course work in increasing their personal productivity at work.

As a result of the factors discussed above, the MIS program content at the graduate and undergraduate levels differ considerably from each other. Moreover, the programs differ somewhat from the traditional DPMA and ACM recommended curriculum (see references (1) and (2)

PROGRAM CONTENT

(a) Undergraduate

The AACSB core requirements constitute an integral part of the undergraduate curriculum since the undergraduate business school population is mostly foreign, young, and lacking in previous business education. Next, since the students are expected to assume significant responsibilities in the area of computers such as office automation upon returning to their home countries, the BS in MIS places greater emphasis on desktop computers, LAN's, use of canned software, and dissemination of computer skills to others in the organization. These students are less likely to have access to mainframe and large minicomputer systems on the job. The undergraduate courses are:

NUMBER	COURSE TITLE
ENG 102	Library Skills
ENG 103	Developmental Reading
ENG 104	Public Speaking
ENG 105	Composition
BUS 101	Intro. to Business Organization
ECO 106	Principles of Micro-Economics
ECO 107	Principles of Macro-Economics
MIS 101	Computer Applications
MIS 240	Software Applications in Business
MTH 105	Algebra in the Practical Context
PSY 205	Personal Imperatives
BUS 203	Principles of Accounting I
BUS 204	Principles of Accounting II
BUS 220	Mathematics for Business & Finance
ENG 205	Crit. Thinking & Logical Presentation
IIS 212	Science, Art, Literature

IIS 213	Phil., Technology, Exploration
MTH 210	Intro. to Applied Statistics
NSC 213	Time, Space, and Technology
PVA 205	Arts and Ideas
BUS 304	Principles of Marketing
BUS 306	Organizational Financial Management
BUS 311	Business Law
BUS 312	Management Practice and Org. Behavior
BUS 325	Production Management
HB 305	Law, Values and Society
MIS 3XX	Management Information Systems
MIS 3XY	Computer Programming and Logic
MIS 3XZ	Systems Development
BUS 520	Business Policies
IIS 411	The Living Constitution
IIS 505	Intercultural Understanding
IR 405	Contemporary International Relations
MIS 320	Data Base Structures
MIS 405	Operations Analysis and Modeling
MIS 4XX	Teleprocessing and Computer Networks
MIS 420	Seminar in M.I.S.

(b) Graduate

In contrast to the undergraduate program, the graduate MS in MIS program has a greater emphasis on mini/mainframe computer selection and use, management of the MIS software development process, peoples' issues, data communications, the selection of vendors, and effective management of large databases. These are the problems most likely to be confronted by middle managers in an American or multinational corporation. Many of the graduate students have bachelors degrees in business and thus do not have to take some of the business core courses. Instead they can take elective courses in the MIS area. The graduate courses are as follows:

NUMBER	COURSE TITLE
BUS 602	The Business System
BUS 612	Accounting for Managers
BUS 614	Financial Resource Management
BUS 617	Operations Management
BUS 624	Marketing Management
BUS 626	Corp., Society, and the Legal Environment
BUS 650	Strategy and Organization
BUS 685	The Economic of Pol. & Social Issues
BUS 6XX	Quantitative Methods for Business
MIS 620	Management Information Systems
HR 601	Groups & Organizations
MIS 630	System Analysis and Design
MIS 650	Networks & Distribution
MIS 670	Data Base Management
MIS 680	Topics in Management Information Systems
MIS 6XX	Information Resource Management
MIS 6XY	Computerized Management Support Systems

SUMMARY AND CONCLUSIONS

This paper has discussed some reasons for offering unique MIS programs at the undergraduate and graduate levels in a university servicing both a domestic and international student market. The two student populations have differing levels of work experience, responsibilities, and employment opportunities. These differences dictate overall program emphases and specific course requirements that will meet the goals of each of the respective market segments in the most appropriate manner. For a private university such as USIU, the success of any new program will be determined by its ability to attract a sufficient number of students (to have economical class sizes), to support a critical mass of faculty (in numbers and diversity), and to justify the investment in computer laboratory and

other equipment required by the MIS program.

REFERENCES

(1) DPMA Model Curriculum for Information Systems Education, Data Processing Management Association, 1987. Southwestern Publishing Co.

(2) ACM Recommended Curriculum for Computer Science and Information Processing Programs in Colleges and Universities 1968-81, ACM Education Board, Association for Computing Machinery, 1981. See also Draft Report of the ACM Task Force on the Core of Computer Science, ACM Education Board, Assn. for Computing Machinery, 1987.

TWO CLOSELY RELATED INFORMATION SYSTEMS CURRICULA IN SUPPORT OF ORGANIZATIONAL MANAGEMENT AND IS TECHNICAL PROFESSIONALS

Herbert E. Longenecker, Jr.(1), Warren A. Beatty(2), Lothar Zenkert(1)
David L. Feinstein(1), V. Gordon Moulton(1) and Carl C. Moore(2)

(1) Division of Computer and Information Science, and
(2) Department of Management, College of Business
University of South Alabama, Mobile AL 36688

IS Programs Philosophy

In the ACM-1984 and DPMA-1986 curriculum models for Information Systems, the concept of training IS students to become IS professionals differ from traditional models in which computer professionals received extremely technical training with few organizational skills. In these new curricula a considerable fraction of the courses are devoted to developing organizational skills. Both of the newer models require inclusion of significant organizational as well as liberal arts studies to produce information systems graduates more capable of handling real-world people-related organizational issues.

In a similar manner, business management students have been educated primarily in business and organizational skills. Our current business management curriculum requires several courses that introduce students to Information Systems, microcomputer techniques, database methods and techniques, as well as the introductory concepts of computerized information systems in organizations.

We suggest that a new educational emphasis is required in order to graduate technically trained business managers capable of intimate knowledge of Information Systems. Also, we suggest that it is necessary to continue to train IS professionals who will have a close relation, by training, to the organizational executives. These IS professionals should have very strong course work in organizational and managerial skills in addition to technical training in IS.

What is new is that we feel business managers should also have a very strong training in the technical aspects of IS. These new technical skills of business managers should complement the skills of the IS professional. More effective system development teamwork between business management and IS professionals is a goal of these programs.

The New Approach

In business, government, private, and other organizations there is an increasing awaraness of the capabilities of Information Systems, as well as an increasing dependency and necessity for these systems because the systems make possible achievement of goals of the organizations in a cost beneficial manner.

Thus, in planning the curriculum for Information Systems two groups of individuals are acknowledged who require significant education: 1) Organizational Management and 2) IS Technical Specialists. Two programs have been developed in support of these educational goals.

Both of the programs descreibed below will be described in the 1988-1989 Catalog for the University. Program 1, the Business Management BS with MIS Concentration, comes directly within the acreditation standards of the AACSB. It is technically a management degree whose electives are drawn almost exclusively from the CIS Division offerings (outside the College of Business).

Program 2, Computer and Information Sciences BS with IS Specialization, is designed to be compatible with either the ACM IS curriculum model or the DPMA CIS undergraduate model. The program requires more course work than either ACM or DPMA model, but is compatible with demands of employers of our students.

Program 1 -- Business Management BS with MIS Concentration:

The business management bachelor of science degree with a concentration in MIS is a business degree supervised by business coordinators and faculty to prepare a student for a career in business management, but who has sufficient IS foundation skills they could be competitive for entry level IS technical positions. Specific program objectives include:

1 - Provide an AACSB certifiable degree in Business Management Sciences.

2- Develop significant Information System Foundation skills by taking course work specified by a CIS/MIS Foundation sequence.

3- Develop significant organizational/ business skills by completing required course work ordinarily a component of the management degree.

Program 2 -- Computer and Information Sciences BS with IS Specialization:

The CIS bachelor of science degree with a specialization in information science is one of three programs in the Division of Computer Science (other two being programs is Computer Science and Computer Development/Engineering). The IS program is coordinated and executed by IS faculty. The program will prepare students for entry level IS technical positions, the traditional DP route, information officer, database administrator, or other IS positions. The IS graduate will have in addition to the IS/MIS foundation sequence, additional training in an IS Technical Specialization.

1- Develop significant Information System Foundation skills by taking course work specified by a CIS/MIS Foundation sequence.

2- Develop significant organizational skills (as a component of the IS/MIS Foundation Sequence).

3- Develop a significant IS technical specialization through additional IS technical courses.

The CIS/MIS Foundation Sequence

It is contended that in current organizations both groups share a common requirement for broad and specific training to develop an adequate foundation for a productive organizational relationship. Table 1 contains a statement of Foundation Objectives and related supporting course work.

This foundation sequence has evolved both in the College of Business and the Division of Computer Science over several years. The sequence is not stated explicitly in catalog copy. Rather, the sequence is apparent on analysis of both published curricula. The foundation sequence accounts for almost 84 % of the student's curriculum.

Both programs require strong backgrounds in computer science skills, as well as the basics of information systems. The computer sequence consists of an introduction to problem solving methods, a rigorous PASCAL syntax and philosophical program development methods sequence, a course in computer concepts, architecture and an introduction to operating systems. An additional course provides considerable experience with hands on microcomputer experience. The COBOL sequence provids entry level programmer training, and is a rigorous introduction into the programming of systems and data file manipulation. Course graduates are effective programmers in any language.

The IS Sequence involves an introduction to the theory of information system usage in an organization, life cycle concepts, and opportunities in the field. The Database course addresses more complex data aggregates and the process of normalization as well as conceptual model formation. The course in Analysis concentrates on conversion of physical flows to a database, as well as the development of information flows and requirements. Human behavior and project management are included in the analysis course. The systems design course is a continuation of the analysis course. The goal is to complete the study of the system development life cycle, excluding programming. Audit controls and decision support concepts are covered in two additional courses.

One course in calculus is required since we feel that many organizational models depend on a basic knowledge of that level of mathematics. Courses involving measurement of organizational performance include courses in statistics and business research techniques.

Accounting, economics and finance are regarded as the "language" of organizations and form the backbone of the organization. Management science and understanding of the Law are involved in controlling an organization in an appropriate manner, especially as external activities are involved.

IS/MIS Professional Component Sequence

Both business management and IS professionals have a common necessity for a thorough understanding of goals and methodologies of computerized information systems. This sequence provides the technical foundation for both professions. This sequence is described at the bottom of Table 2.

The sequence begins with an introduction to the principles of information systems, including general system theory, decision theory, information theory, organizational planning and control. The sequence then specifically examines the above topics, as well as database development, systems analysis and design, audit control theory, and decision support and expert systems.

Career Specific Professional Component Sequences

Both programs require completion sequences (16 %) tc confer distinct flavors to the two degrees. These sequences are presented in Table 2. The IS/MIS foundation sequence provides entry level competence for functioning effectively in a contemporary organization. The career specific professional component sequences provide advanced course work to build either business management skills or IS professional skills.

The business graduates (Program 1) require more knowledge of business than the IS/MIS Foundation sequence contains to have a working knowledge of business, and for the program to be AACSB certifiable. These courses include additional emphasis on finance and banking, operations management, organizational communications, marketing and business policy.

Likewise, the IS professional must develop increased skill levels in systems design, programming techniques, and documentation. The data communications software engineering and principles of management of MIS form a significant extention. Time is allowed in the curriculum for several electives to taylor the curriculum. Advanced course work may be chosen from the following areas: Discrete mathematics, advanced data structures, operating systems, graphics, modeling and simulation, artificial intelligence and artificial intelligence programming.

While it would be nice for the IS Professional to have these course experiences, there is not time in the standard curriculum. Also, there is more need for the IS Professional to learn considerably more about IS. These capstone sequences are the branching points in which business management starts becoming distince from the IS professional.

Student Advising and Activity

Each program is responsible for advising its students. This assures appropriate interactions between program faculty and students. These course sequences are rigorous and require careful attention to schedule and special needs of the students. Any deficiencies of the student or alterations of schedule may easily add an additional year of study.

Table 1 -- Common Curriculum Sequences

(72%) IS/MIS Foundation Objectives and Sequence

The sequences below will provide undergraduates with a broad background enabling them to be effective participants in a business or other organization in which they may be required to utilize mangagement and information systems skills. This IS/MIS Foundation is common between both Program 1 and Program 2.

(14%) Computer Science--Problem Solving, Pascal, Program Design Data Structures, Searching and Sorting, COBOL, Files, Indexed Files, Microcomputer Techniques (spreadsheets, word processing, database, scheduling).

- 3c Problem Solving, Pascal, Data Structures
- 1c Computer Concepts, Arcitecture, Operating Systems
- 1c Microcomputer Techniques
- 2c COBOL Syntax and File Processing

(16%) Organizational Skills--Accounting and measuring organization position and peformance, Macro and Micro Economics, B. Law

- 3c Accounting
- 2c Macro and Micro Economics
- 1c Management Science
- 2c Business Law

(14%) Mathematics/Statistics/Science--Algebra, trigonometry, an intro to differentiation and integration, statistics and operations research

- 1c Calculus (others if needed to satisfy prereqs)
- 3c Statistics including quantative methods
- 2-4c Laboratory Science
- 1c Philosophy of Logic

(28%) General Liberal Arts--English, Writing, Communications, Humanities, Fine arts, Sociology, Psychology, Philosophy

- 4c English, Communications
- 6c Humanities
- 2c Social Sciences
- 2c Physical Education/Military Science

(12%) IS/MIS Professional Component Sequence

(12%) Information Systems--Physical Flows, Information Flows, Database, Analysis, Design, Implementation and Auditing of an Information System, Decision Making and Support, Expet Systems

- 1c Principles of Information Systems
- 3c Database, Analysis and Design
- 1c Audit Controls
- 1c Descision Support and Expert Systems

Note: Courses (c) are taught in 10 week quarters, meeting 4 hours per

week. Percentages are percent of total curriculum, assuming there are 50 total courses in the degree program.

Table 2 -- Career Specific Professional Component Requirements

The Foundation Sequence of both programs is essentially identical. However, the capstone sequences provide significant differentiation between the two programs.

The Business MIS concentration degree trains potential high level managers by focusing on finance, operations, management and marketing courses essential to the success of a business.

In a similar fashion, the CIS IS Speciliazation provides opportunity to train graduates in software and computer system development skills required to implement Information systems.

Program 1 -- Business Management BS with MIS Concentration

The foundation sequence develops broad skills which are refined in the capstone sequence. Effective business leaders will require MIS skills, but still require the traditional complement of the AACSB body of common knowledge inorder to provide an appropriate frame of reference. The MIS skills will enable the business manager to knowledgeably participate in analytic and highly technical developments, but with a broader knowledge of business skills.

(16%) Balance of Course Work--AACSB degree requirements

- 2c Finance and Banking
- 1c Organizational Communications
- 1c Operations Management
- 2c Marketing
- 1c Business Policy
- 1c Seminar on Current Management Issues

Program 2 -- Computer and Information Sciences BS with IS Specialization

The foundation will prepare graduates with minimal level of IS entry skills. The IS capstone sequence is designed to significantly augment program and information system design and management skills. The IS graduate has a significant organizational skill level not common to graduates of highly technical programs. The IS graduate will understand business and organizational problems and will be an effective leader. The technical skill level of the IS graduate will enhance entry level performance.

(16%) Balance of Course Work--IS Specialization

- 1c Software Engineering
- 1c Data Communications
- 1c Management of Management Information Systems
- 1c Computers and Society
- 3c Technical Electives(e.g. Data Structures/AI, etc.)
- 1c Elective

A UNIFIED APPROACH TO TEACHING SYSTEM DEVELOPMENT COURSES

Iraj Hirmanpour
Computer Science Department
Embry-Riddle Aeronautical University
Daytona Beach, Florida 32014

ABSTRACT

The main goal of the systems analysis course is to teach students how to collect requirements. While the subject of requirements collection is covered again in both database design and software design courses, the approaches is often different. This paper is a discussion of a modeling methodology that offer a unified approach to analysis and design of both data and processes. Using this approach, the product of analysis called the essential model, can be used as input to logical database design in the database course and to logical software design in the software design course.

INTRODUCTION

The problem of information system development consists of three major sub-problems. They are the requirements specifications, design, and construction. The requirements specification phase seeks to identify the needs of the ensuing system. It primarily covers the "what" questions, i.e. what are the system's requirements and "what" processes, data flows and data stores are needed to satisfy these requirements. The design activity deals with the problem of identifying a system configuration that satisfies the stated requirements. The design phase, therefore, deals with the "how" question. "How" should the system be configured to meet the required objectives? The construction phase consists of writing the program codes and building the databases to implement the design, then testing the system to insure that it is error free and that it indeed satisfies the stated requirements.

The purpose of this paper is to discuss a modeling methodology that deals with both dimensions-- data and process -- of an information system. Such an approach, if taught in the systems courses, can provide students with a better understanding of the life cycle.

THE MODELING PROCESS

The modeling approach under consideration consists of building two models: the Essential Model, and the Implementation Model. The Essential Model describes the system's data and process requirements, and is implementation independent. The essential model is then mapped

into an Implementation Model which shows the final design and implementation of the requirements.

THE ESSENTIAL MODEL

The Essential Model identifies the essential processes and data stores that are needed to meet the system's objectives. The main focus of this model is on processes and data stores are needed to carry out the system's mission; It is not concerned with "how" a process is carried out and in "what way" data are stored. This model consists of three sub-models: (1) The Environmental Model, (2) The Data Model, (3) and The Behavioral Model.

THE ENVIRONMENTAL MODEL: The Purpose of the Environmental Model is to identify the system's boundaries and the scope of the problem. The model consists of three components: (1) A statement of the system's purpose, (2) an event list, and (3) a context diagram. Figure 1 shows components of the environmental model.

THE DATA MODEL: For a data dominant system, as used in the majority of business data processing systems, a data model is built using the constructs of the Entity Relationship data model. The event list is used for initial entity and relationship identification.

THE BEHAVIORAL MODEL: The Behavioral Model identifies the detailed processing needed to satisfy the busines's requirements. its is built in steps. The first step is to develop an enent-partitioned data flow

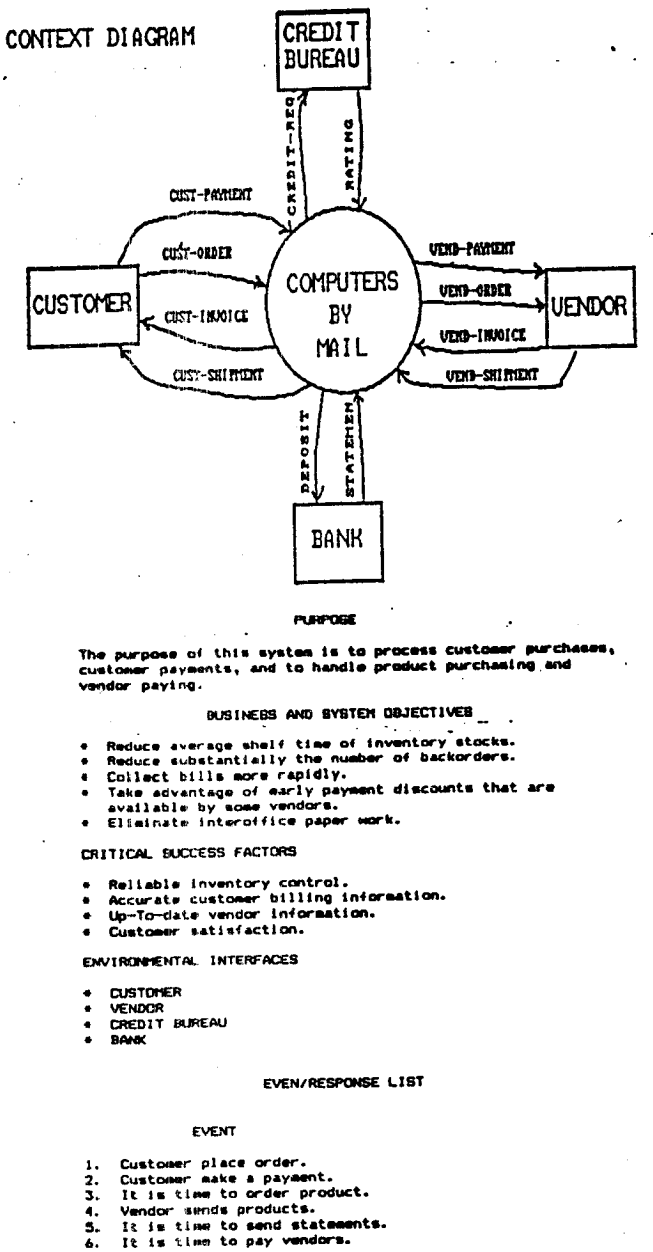


Figure 1: Environmental Model

diagram. This is done by identifying a response for each event, and input/output data flows and data stores for each response. By aggregating processes a higher level diagram can be created which reduces the overcrowding of an event partitioned DFD. This higher level diagram is often referred to as the system level diagram. Each group of processes that are

combined to form a system level bubble becomes the child of that bubble. The procedure outlined so far thus provides three views of system's functions: the context diagram provides a high level view, the system level identifies major business functions the system should support, and the detail level identifies each business function. Additional decompositions can be carried out if it helps to further clarify the system.

THE IMPLEMENTATION MODEL

As the essential model defines the problem domain, the implementation model defines the solution space. For instance, it describes the design and implementation which answers specific requirements. This model also consists of three sub-models: (1)

The Processor Configuration Model, (2) The Software Configuration Model, and (3) The Code Organization Model.

PROCESSOR CONFIGURATION MODEL: The basic component of the implementation model is the "processor." A processor is a person or a machine able to carry out instructions and store data. At one extreme, activities of the system may be carried out by one processor; at another extreme, it may be a network of computers and people, each of which carry out a fragment of the system's activities.

SOFTWARE CONFIGURATION MODEL: In this phase, the computer processors of the previous model are repackaged according to tasks; that is, all interactive tasks are packaged together. Similarly, non-interactive tasks and processes that have the same run cycle are packaged together.

CODE ORGANIZATION MODEL: At this stage, the software configuration model has divided the system into design units. The principles of structure design (as proposed by Yourdon & Constantene) are then applied to each design unit to create a software blue print. It is this resultant model that is coded and tested. Figure 2 shows an abstract representation of components of the implementation model.

The models described are primarily graphic ones which are easier to understand than textual descriptions. However, not all elements of a system can be documented by graphical means. Textual descriptions must be provided to fill in details needed for building the system. The project dictionary is a compendium of these graphs which describe the details of data flows, data stores, entities, and processes. The project dictionary will not be discussed here; it is, however, an important element of the modeling process.

CONCLUSIONS

The modeling approach described offers a unified approach to systems analysis and design. It clearly separates analysis and design activities. But treats data requirements and process requirements specifications as a unified activity.

This approach can be introduced in the systems analysis course, where the building of the essential model becomes the major activity of the course. The

database design course can then start with logical database design rather than engaging in the discussion of requirements analysis. The software design course will then emphasize the design and implementation details of software and data distribution. This new method teaches students how to use a unified methodology with the life cycle approach for building large scale information systems.

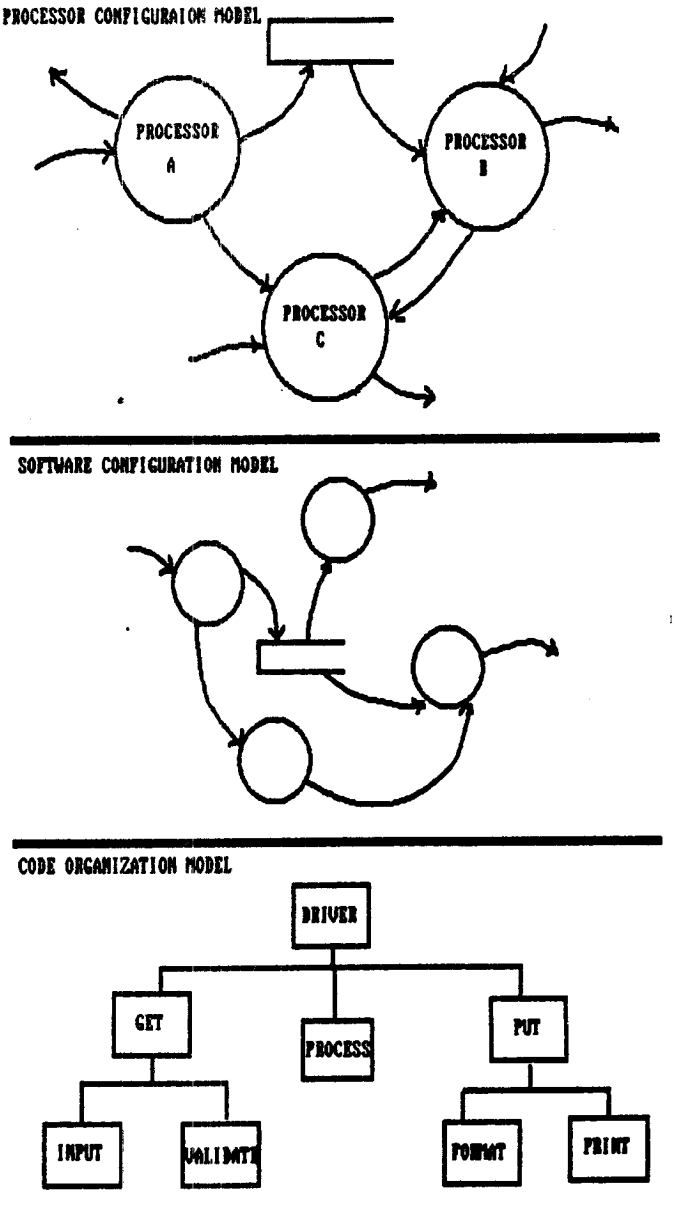


Figure 2: Components of Implementation Model

REFERENCES

1. P. P. Chen, "Entity Relatio nship Model: Toward a Unified View of Data", ACM-TODS 1, 9-36, 1977.

2. J. Palmer and S. McMenamin, Essential Systems Analysis (New York: Yourdon Press, 1984).

3. Douglas T. Ross, "Structured Analysis (SA): A Language For Communicating Ideas", IEEE Transactions on Software Engineering, vol. SE-3, No. 1, pp. 16-34, 1977.

4. Paul T. Ward, Systems Development Without Pain: a user's guide to modeling organizational patterns, (New York: 1984).

MAKING THE COMPUTING STUDENT A SKILLED COMMUNICATOR

Alka R. Harriger
Associate Professor of Computer Technology
KNOY Hall of Technology
Purdue University
West Lafayette, IN 47907

Ann Stuart
Professor of English
1800 Lincoln Avenue
University of Evansville
Evansville, IN 47722

ABSTRACT: Most computing professionals agree that effective communication skills are essential for success in most areas within data processing. To this end, most college-level baccalaureate programs require students to successfully complete a certain number of English/Communication courses. In the Computer Technology Department at Purdue University, we have found that it is also important to apply the students' communication skills directly to major computing courses. This paper identifies six communication skills students need to practice within the major area of study, offers successful applications suitable to a variety of computing courses, and lists the additional time required to implement these recommendations.

INTRODUCTION:

The Computer Technology Department at Purdue University offers a two-plus-two year degree program in Computer Information Systems (CIS). Students learn important computing concepts and fundamental problem-solving skills in the first two years. After that, if they choose to continue with the program, they study higher level computing topics and gain practical experiences through coursework involving case studies.

In an effort to provide industry with a very marketable CIS graduate, our department regularly consults with an Industrial Advisory Committee. This committee consists of the president of our student alumni organization and 12-15 representatives from companies who typically hire our graduates.

In 1982, this committee advised that our graduates needed more practice with their written and oral communication skills. At that point, the department identified two alternative solutions for improving our students' communications skills:

(1) integrate written and oral presentations within assignments of existing computing courses, or

(2) add a new course devoted to building communication skills for computing professionals.

The department acted upon both alternatives.

Many of the computing instructors developed assignments that called for written and oral work. Through informal discussions, faculty shared successful techniques with others, and workable assignments were developed.

For example, in the freshman course, "Introduction to Computer Based Systems," a term paper is required. Students are guided through the process of selecting and narrowing a subject, planning their schedule effectively so the project can be completed, conducting research, determining an effective organization, documenting resources correctly, and writing and editing their work. More details on this course and the term paper can be found in [1]. In a sophomore course, "Systems Analysis and Design Methods," written documentation is required for each step of the system development life cycle. In a senior course, "DBMS Comparative Analysis," formal reports and oral presentations on the Data Base Management Systems under study are required.

The other alternative which the department is currently applying in its CIS curriculum is to offer a new course "Reporting and Documentation Techniques." The guidelines for course development have been the "CIS Communication, Reporting, and Documentation Techniques Course" outlined in the Data Processing Management Association's 1986 Model Curriculum for Undergraduate CIS Education. [2] A suitable textbook for such a course came out this year and is being used this semester. [3]

The pilot course was offered in fall 1987. A major emphasis is to address communication problems common to the CIS undergraduate. We determined six basic needs. These needs are identified in the next section along with suggested assignments to practice the skills and additional time needed to implement these assignments. Such assignments, or ones like them, can be easily used by others teaching CIS students. They are adaptable to assignments in general CIS courses or can be the basis for developing a special course similar to the one we are introducing at Purdue.

COMMUNICATION NEEDS AND APPLICATION ASSIGNMENTS:

ASSIGNMENT ANALYSIS:

Problem. Some students are careless and inaccurate readers. They are unable to interpret assignments accurately and precisely. We find that they too often do not write the the precise task assigned and too often misunderstand or present too generally information they were responsible for reading, understanding, and acting upon. If left untreated, this problem leads students to misrepresent facts and fail in performing the required tasks.

Suggested assignment. Each writing or speaking assignment is designed so that it comes from a particular person with whom a typical CIS employee communicates (client, supervisor, user, marketing, etc.), and it asks for a specific task (feasibility study, analysis of problem, proposed solution, set of instructions, market analysis, etc.) based upon specific information handed out to the students. The work is evaluated according to how well the student addressed the particular task in a way suitable to the particular audience and how accurately and directly he or she dealt with the given information. General, vague, catch-all writing is not rewarded.

Additional time. When the specific audience and task are part of the assignment specifications, additional class time on the subject of assignment analysis is minimal. When the assignment is distributed, students should be reminded that they must do the assigned task for the correct audience. Although not necessary, it is the instructor's option to add a milestone to the assignment which requires the student to precisely define the task and audience.

PROBLEM SOLVING:

Problem. Some students are poor problem solvers because they do not analyze in depth, consider alternatives, and foresee the possible problems with their suggested solutions. When submitting their solutions to assigned problems, students submit the results of their "brainstorming" rather than more organized results. Finally, if they begin pursuing an important point, they fall short of completing the thought.

Suggested assignments. CIS professionals are experts at problem solving since it is closely related to the program development life cycle and the systems development life cycle. If the students can be shown the direct application of a familiar CIS topic to the writing process, their writing task is simplified.

One application is the term project in the freshman CIS course discussed earlier. The instructor has subdivided the project into a few milestones, where each milestone corresponds to a major stage of the systems development life cycle (or problem solving process). Although many of the milestones are assigned by the instructor, the students are required to add milestones of their own. Then, the students are shown how each milestone builds upon the results of the preceding milestones. By successfully completing each milestone in order, just like each stage in the systems development process must be completed, the final product is better and easier to complete.

Additional time. Assignments to help students improve their problem solving skills do require more planning. In lower-level courses, we recommend that the instructor define the individual milestones for major class projects. In higher-level courses, the students should be required to define their own milestones and be graded on meeting their set goals. Although the instructor has more milestones to grade, the grading effort is simplified since "pieces" of the final products have been graded previously in earlier milestones. The additional planning and grading time are worth the effort, since the students' final products are much better in the long run.

EFFECTIVE VISUALS:

Problem. Some students do not consider alternatives to the written word. They present everything in writing when often a visual would communicate more effectively. We have found that students frequently do not select visuals because they do not know how to create reliable and effective visuals. They also do not know how to evaluate whether a visual is legitimate or not. We need to show the students how easy it is to misrepresent information.

Suggested assignments. In order to educate students to be creators and users of effective graphics, we show actual advertisements for computer hardware/software from newspapers, magazines, etc., which make ineffective use of visuals (price comparison chart, sample output, etc.). We emphasize the ethical creation of graphics by illustrating a variety of untruthful graphics. In some cases, the class is divided into groups of 3-5 students. Each group is asked to rate the advertisements, identify problems, and redesign the advertisement so that the visuals are effective.

A variation of the above suggestion which allows more student involvement and interest is to ask the students to locate articles, advertisements, etc. from newspapers or other literature where visuals are used both effectively and ineffectively. They copy and critique the visuals. If time allows, the students are asked to prepare one or two visuals for display on the overhead and give a five-minute presentation of their analyses.

Additionally, by means of an overhead projector, we show a variety of common visuals--graphs, drawings, tables, pictures, flowcharts, and so forth. Each illustrates a principle of effective or ineffective construction. In cases where inaccurate information is given, we discuss ways to promote the product while maintaining accuracy. Our intention is to teach students not only how to create and integrate visuals into their texts but also how to recognize the value of reliable visuals presented to them for use.

Finally, we discuss the option of substituting visuals for written text. Students are asked to justify during their planning time on certain assignments where visual presentation of information would be more effective than the traditional text.

Additional time. Incorporating visuals into the CIS class can require from part of one class to several classes, depending upon the instructor's preferences. From our experience, the students enjoy these classes a great deal, since they have an opportunity for input. Introducing the importance of visuals can be done in a 20-minute class meeting. Adding examples to the introduction will take an entire class period. When student involvement is added, the number of additional classes depends upon the number of presentations to be made.

EFFECTIVE FORMATTING

Problem. Some students are too often tied to the paragraph presentation of information. They need to understand the reader's need for clarity in reading and for quick referencing of information. Students need to be sensitized to the arrangement of the written words on the page as well as the use of appropriate formatting to draw the reader's attention to important points.

Suggested assignments. In the area of programming, most schools require certain standards for documentation. We constantly stress the life cycle of the program and the fact that it would be seen by other people. This shows the students a good reason for following stated standards for headings, indentation styles for varying levels of control structures and placement of comments.

In the area of other written assignments, we integrate formatting into every possible assignment. Students are continually asked to consider the readers' needs for easy reading and access of information. If word processing is not available, we often allow students to suggest layouts by printing different type sizes and fonts. Our goal is to encourage students to develop an eye for the way information looks on the page and to sense how useful the layout is for the reader.

Again, we use examples from existing computing literature to emphasize the importance of using:

- consistent headings,
- white space,
- different fonts, print sizes, boldface, underlining, etc.
- lists in place of paragraphs,
- numbered items on a list,
- bullets with lists,
- left, right, and full justification, and
- a variety of indentation levels

to denote varying importance of stated points or to draw attention to important sections. We also use examples which make ineffective use of formatting, thus confusing the reader.

Additional time. Twenty minutes should be sufficient for discussing the ideas above. From that point, formatting should be built into the assignments; thus, no additional class time would be needed.

EFFECTIVE SPEAKING:

Problem. Some students are ineffective speakers. Our goal is to allow sufficient opportunities for oral presentations and varying types of presentations (interviews, group presentation, formal presentation, introductions, etc.) to help students become comfortable speaking their thoughts.

Suggested assignments. Perhaps the easiest method to give students practice speaking is in the form of questions asked during class. When a student volunteers (or is called upon) to answer a question, he or she must identify himself or herself and state the answer clearly and audibly. Sometimes, as instructors, we tend to repeat the student's response instead of insisting that the student project his or her voice the first time. If the class did not hear or understand the student's response, we ask the student to repeat the answer to the class.

We integrate speaking into every assignment possible so that the students can speak about their ideas. We try to make every student comment a presentation. If others in the room cannot hear the speaker or understand what was said, we encourage students to project and speak clearly. One habit we encourage is courtesy. Students often do not remember to thank the person who introduced them or do not introduce the next speaker in the group. Our intent is to instill good business manners at the same time as we encourage effective speaking.

Additional time. Again, the instructor has a range of choices here. If the instructor just requires students to respond to posed questions in an audible manner, no additional class time is needed. If the instructor desires to give students opportunities for singular or group presentations, then the additional class time would depend upon the number and length of all presentations.

PEER EVALUATIONS:

Problem. Some students do not effectively give constructive criticism of their peers and do not receive constructive criticism well. We provide opportunities for students to react to each others' work and comment on it thoughtfully and responsibly.

Suggested assignments. In the programming area, we require students to do group walkthroughs (logic and code). This helps the students understand how a fresh view can find a simple mistake hidden within lines of logic or code. They also realize that if they help someone else, that favor would be returned.

For other written works, students are encouraged to offer each other the service of a peer review. It is important to stress that one cannot review another's work until one's own work has been completed. Many of us are involved in writing ourselves. We relate the importance of receiving comments from our peers before we submit papers to various conference proceedings. Again, we try to have the students play the role of the reviewer. We ask them to answer the question, "What impression would he or she have if a paper was submitted with grammatical/spelling errors or misleading facts?"

For oral works, every student in the audience is required to complete an evaluation form for every student speaker. In addition to specifying the degree of effectiveness in various areas, the evaluator must explain in a constructive manner why each area was effective (or ineffective). When the speaker receives all of the evaluations, both the good and bad points become clear through similar comments made by several evaluators.

Additional time. We consider code and logic walkthroughs to be a very important part of the program development life cycle; therefore, we have students perform them during class time. If we made walkthroughs out of class assignments, we would have no way of enforcing that they were done at all or done correctly.

Peer evaluations of other written works are done out of class. Sample grading checklists are distributed so that the students know our grading criteria for their assignments. This process requires no additional class time, although the grading checklists must be written a few weeks before students submit the completed assignments.

Peer evaluations of speakers are completed during their speeches. We usually give one additional minute after each speech to let the evaluators finalize their comments. Therefore, very little additional class time is required (about one minute per speech). Again, the instructor must create and duplicate sufficient evaluation forms prior to any speech.

CONCLUSION:

A widely acclaimed national study has shown that fundamental skills of an increasing number of college-aged students need great improvement and suggests ways to improve the situation. [4] Another study has revealed that basic skill deficiencies, including the areas of reading, reasoning, writing, and interpersonal skills, are limiting opportunities for advancement in the job market. [5] These deficiencies pervade all educational arenas,

including computing. In fact, some have gone as far as saying that the ability to communicate effectively is the most important requirement for a data processing professional. [6]

As CIS educators, we are responsible for graduating students who not only possess technical expertise, but also have the ability to communicate their ideas effectively. We feel that all CIS educators can give CIS students practice refining their communication skills. In this paper, we presented six problem areas, suggested assignments to give practice in each area, and discussed the additional time needed to implement our suggestions into any course. At the presentation of this paper, we will bring specific examples of our suggested assignments and discuss the varying ways in which they were implemented in our program.

REFERENCES:

[1] Harriger, Alka R. and Helene P. Baouendi. "A Research Case Study for CIS I Students." *ISECON '87 Proceedings of the Sixth Annual Information Systems Education Conference*, October 31 - November 1, 1987, pp. 107-110.

[2] Data Processing Management Association. "CIS/86-20 CIS Communication, Reporting, and Documentation Techniques." *The DPMA Model Curriculum for Undergraduate Computer Information Systems*, October 1985, pp. 54-56.

[3] Stuart, Ann. *The Technical Writer*. New York, New York: Holt, Reinhart and Winston, 1988.

[4] The National Commission on Excellence in Education. "Text of the Report: 'A Nation at Risk: The Imperative for Educational Reform.'" Vol. 26, No. 10, *Chronicle of Higher Education*, May 4, 1983, pp. 11-16.

[5] Sticht, Thomas G. and Larry Mikulecky. "Job Related Basic Skills: Cases and Conclusions." *Information Series No. 285*.

[6] Walton, Richard and Ron A. DiBattista. "Communication Skills: The Achilles Heel of your Computer Education Program? - Part I." Vol. 1, No. 4, *Interface*, Winter 1979. pp. 36-40.

CURRICULUM DESIGN FOR END-USER/DEPARTMENTAL COMPUTING

Richard Discenza, University of Colorado at Colorado Springs
Alden C. Lorents, Northern Arizona University

ABSTRACT

The increased usage of fourth generation software packages has prompted educators to rethink attitudes about information systems and how to consider modifying current CIS programs to incorporate these new tools. This paper presents some specific suggestions for modifying CIS curricula so that they incorporate end-user/departmental computing concepts. The suggestions are presented in such a way so that they can be merged into the DPMA Model Curriculum (1986 version) without eliminating the educational experience designed for developing the traditional programmer/analyst.

INTRODUCTION

Microcomputers and fourth generation languages are now shaping the development of information processing in a very fundamental way. In the late 1970s and early 1980s, computer time became less costly, personnel costs increased, and programming backlogs continued to grow. The average backlog for getting new software applications ranged from two to four years and some organizations have reported backlogs of five or even ten years (1, 2). In addition, maintenance backlogs reached the point where they would require many months or even years to eliminate if no new requests were accepted (2).

Given this environment, more end-users began turning to fourth generation languages. These languages allowed for end-users to bypass programmer/analysts in the quest for information. Martin (3) has identified fourth generation languages as:

1. Increasing user productivity by factors as great as ten to one
2. Allowing hands-on usage by end-users within three days after an individual fourth generation language has become operational at a work unit

He stated as early as 1982 that "In typical corporations with substantial well-designed data bases in existence, seventy percent of end-user needs can be met with query languages and report generators. In many cases, less than ten percent of the end-user demands for new applications require conventional DP development with formal programming specifications and languages such as COBOL or PL/1" (4). Organizations such as Aetna Life and Casualty, Chase Manhatten Bank, Metropolitan Life, and Arthur Young and Company are finding fourth generation language alternatives much more satisfactory than the traditional data processing approaches for meeting their departmental information processing requirements (6). Because of this trend, it has become important to rethink attitudes about information systems and how this important subject is being taught in schools of business.

In CIS '86, the DPMA Model Curriculum for Undergraduate Computer Information Systems presented a new departure by introducing CIS/86-2 "Microcomputer Applications in Business" (5). This course reflected two major trends in information systems: first, the course provided computer instruction in non-procedural fourth generation languages and second, it set up a specific course in the curriculum for microcomputing. Now, many CIS departments face an additional challenge: How can those schools who have essentially adopted the CIS '86 Model Curriculum incorporate additional emphases for end-user computing into their curriculums?

Previous literature has been directed towards the need for greater training in end-user computing and some pedological approaches for teaching specific aspects of end-user computing; see for example (2, 7, 8, 9). This paper presents some detailed suggestions for curriculum design in end-user/departmental computing that can be used to augment the CIS '86, the DPMA Model Curriculum for both undergraduate and graduate programs.

It should also be noted that computer literacy is fast becoming an important component of business education in general. Hence, there is a need to integrate end-user computing with the functional areas of accounting, finance, operations, and marketing. The proposed end-user/departmental computing curriculum will also assist schools in developing programs that have a dual emphasis, such as a major in finance with an emphasis in end-user computing. The program would then serve multiple objectives for students.

A CURRICULUM TRACK IN END USER/DEPARTMENTAL COMPUTING

An emphasis in end-user computing would typically be housed in the College of Business under the CIS area. The program could serve multiple objectives for students:

1. Taken by majors in other areas of business to augment their program
2. Taken with a group of business electives to give the non-business students more business perspective

The program would meet the typical standards of AACSB as well as the general studies requirement of the college or university. In some cases, a student may have to take a few extra hours than is required for graduation if that student is augmenting a major field such as accounting.

PROGRAM PRE/CO-REQUISITES

The program would require the basic computer and information system literacy courses at the 100 level and 300 level. This normally consists of the following at the 100 level:

1. Hardware terminology and concepts
2. Software terminology and concepts
3. Data concepts and elementary structure
4. Word processing
5. Spreadsheet
6. Simple data base query, reporting, data entry

The 300 level course covers the introduction to information systems:

1. Systems and information systems concepts
2. Management dimensions of information systems
3. System development concepts and tools
4. Overview of business systems
5. Data base concepts
6. Cases using decision support/4GL tools

EMPHASIS IN END-USER/ DEPARTMENTAL COMPUTING

The program should try to concentrate on five core courses. Electives should be minimized because of resource constraints. The content of the program could be organized into five courses in the following way:

1. Micro-Based Information Systems:

   This would be a beginning course to get the student into developing systems using a product like DBASE IV or RBASE for DOS with SQL support. The course would quickly immerse the student in special concepts, related to

information systems development and use. Included would be the following:

a. Relational data base concepts
b. Design of screens, data validation, user dialogue
c. Concepts of relational operations
d. SQL language
e. Query and report formatting with control breaks
f. Back-up considerations and referential integrity
g. Normalization concepts

2. Information System Development Tools:

This course would cover the concepts and tools of system development. Potential content would include:

a. Some coverage of design methodologies
b. Use of data dictionaries/system repositories
c. Some exposure to Excelerator or Arthur Anderson's Design/1
d. Prototyping
e. Exposure to workbench products
f. Exposure to system generation products

3. Mainframe-Based Information Systems:

The purpose of this course would be to give the student some exposure to interfacing with mainframe software. Software that could be included are as follows:

a. Use of CMS, ISPF, and related tasks such as setting up data sets, using the editor, getting data to the printer, moving data around, etc.
b. Exposure to IMS or IDMS data base concepts and related definitions for extracting data from these data bases
c. Exposure to DB2 and related tools for extracting data from a DB2 data base
d. Exposure to a 4GL such as NOMAD, FOCUS, or RAMIS

4. Communication and Office Systems:

This course would give the student exposure to several concepts related to information flow in the office. Topics would include:

a. Introduction to communications and networks
b. Using a network
c. Downloading and uploading data, PC/Mainframe interface
d. Exposure to good word processing package
e. Presentation of graphics
f. Electronic mail and document management
g. Installing software packages
h. Managing a local area network
i. Communication with outside networks
j. OS/2 operating system

5. Decision Support Software:

This course would concentrate on the application of software in the decision support area. Cases would be used to get some practice in applying this software to real problems. Topics covered would include:

a. Spreadsheet applications
b. Use of SAS, DB2 to SAS data transfers
c. Presentation graphics
d. Exposure to expert system software

This course content could be integrated with other courses. If the general core curriculum in business is using these kinds of tools on a regular basis, then there may not be as much need for this type of course.

There is a real need for the integration of an end-user computing curriculum with the core curriculum in business. All business students will be end-users. The more they understand about how to be an end-user, the more independent they will be in their own decision making.

THE GRADUATE PROGRAM

This curriculum can also be very appropriate for an MBA/MIS track. MBA students who plan on going into marketing, finance, ac-

counting, or one of the management areas would enhance their marketability considerably with this type of emphasis. Other potential opportunities for MBAs with this background would include:

1. Management consulting firms
2. Information centers
3. Departmental computing areas
4. Firms selling departmental computing services including hardware, software, and communication/office systems.

CONCLUSION

There is no doubt that end-user computing is important today and that it will become more important in the future. The DPMA Model Curriculum '86 for computer information systems has begun to address end-user education; however, because the trend has rapidly accelerated, it is important that DPMA continue to stay abreast with information technology and to provide the needed emphasis in terms of educational guidelines in the end-user computing area. This trend is significant for several reasons. First, greater emphasis on fourth generation software reflects the reality of the business environment. Fourth generation languages are providing solutions to business problems involving analysis, design, and development of computer information systems. This reality must be incorporated into the education acquired by business students. Second, using fourth generation languages improves the ability of instructors to teach the process of analysis, design development, and implementation of computerized information systems by reducing the amount of a student's time spent on specific traditional third generation methodologies.

The proposed curriculum track is not intended to eliminate the traditional programmer/analyst academic preparation. It will take a number of years before the need for COBOL-based system developers will fade out completely since many system generation tools turn out COBOL code. However, this curriculum will change as more use is made of PC-based workbenches, computer maintenance tools, reverse engineering tools, and system repositories. These suggestions are designed to help colleges and universities continue to meet the computing needs of businesses and other organizations.

REFERENCES

1. McFadden, F. and Discenza R., "Confronting the Software Crisis," Business Horizons (November-December, 1987) Vol. 30: 6, pp. 68-73.
2. Prickett, J., "Fourth-Generation Languages and the 'CIS' Curriculum: The Analysis, Design, and Development of Computer Information Systems," Proceedings of the Fifth Annual Information Systems Education Conference (October 25-26, 1986) Atlanta, GA, pp. 23-25.
3. Martin, J., Fourth-Generation Languages -- Volume I, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985.
4. Martin, J., Applications Development Without Programmers, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982, p. 83.
5. CIS '86: The DPMA Model Curriculum for Undergraduate Computer Information Systems, Data Processing Management Association, Park Ridge, IL, October, 1985, 56 pp.
6. Kador, J., "Departmental Computing: Getting Down to Business," Business Software Review (May, 1986) Vol. 5: 5, pp. 51-57.
7. Hayen, R. L., Pietron, L. R., and Johnson, V., "Computers in Business: An End-User Perspective," Interface (Winter, 1987-88) Vol. 9: 4, pp. 9-15.
8. Agganual, A. and Shao, Jr., S., "Fourth-Generation Software Training: A Decision-Making Approach," Interface (Fall, 1987) Vol. 9: 3, pp. 42-46.
9. Wojtkowski, W. G. and Wojtkowski, W., "Fourth Generation Language as a Required CIS Course," Proceedings of Sixth Annual Information Systems Education Conference (ISECON '87), October 31-November 1, 1987, San Francisco, CA, pp. 51-54.

Forming an MIS Corporate Advisory Board: Potential Benefits for Your Program

Dr. Thomas A. Pollack
Dr. John C. Shepherd

Duquesne University
School of Business and Administration
Pittsburgh, PA 15282

ABSTRACT

In today's high technology society, it is important for the educational community to collaborate and communicate with the corporate sector. Collaboration is particularly helpful in a rapidly evolving field such as MIS. One means of establishing this business-higher education alliance is through the formation of a corporate advisory board. Corporate advisory boards offer potential benefits in such areas as curriculum development, student internships, student placement, student aid, faculty development, and fund raising. Those experienced with the administration of corporate advisory boards, however, issue several cautions for those venturing into an advisory board experience for the first time. Common guidelines for forming and managing a corporate advisory board will result in a mutually beneficial experience for the educational entity and the corporate executive.

INTRODUCTION

In an article written in 1979, Timothy I. Healy, the president of Georgetown University stated:

> Business and higher education, as partners, have created for this Republic the most advanced technology and production on the face of the earth, certainly the finest scientific research, the best health care, and above all the profound confidence in our national and individual ability to face almost any problem ... (4, p. 279)

This excerpt depicts the importance for a university to maintain a solid and ongoing working relationship with its surrounding corporate community. After all, both the university and the business community share a mutual interest for the local business environment. The university is obviously interested in producing graduates who are as well-prepared as they can be and, of course, the corporate sector is interested in hiring those graduates who are well-prepared. The need for direct communication between business and education has never been more important than it is today. This is particularly true for programs emanating from a school of business administration and/or those of a technical and scientific nature. The academician must be aware of technology and practices being employed in the corporate sector, especially in the MIS field.

The Management Information Systems (MIS) area has been especially subject to ongoing change in technology and is depicted as a field in transition over the past several years. MIS is a dynamic field, and if an MIS program of studies is to maintain state-of-the-art status, the academic entity cannot afford to become complacent. Curriculum analysis and modification must occur on a continual basis. Advances in the computing field have been rapid and numerous. In spite of the rapid development in computing, however, documents such as the President's Science Advisory Committee Report state that the computing field "now appears to be approaching its infancy". If indeed this is the case, how rapidly will this field evolve in the future? How do we as educators maintain the pace? These are just a couple of the questions constantly being asked.

The answers to the above questions can be found in several areas. The most obvious means for an educational department to maintain currency in its programs is through quality faculty research projects. Ongoing faculty research and publications are vital to maintenance of "state-of-the-art" in curricular offerings. However, faculty must receive a commitment of support from the university for these research projects. Hardware and software offerings must constantly be evaluated and updated. Library resources must be maintained, and access to on-line database sources must be provided. The

curriculum must be constantly updated to reflect research findings and results. This can obviously become an expensive commitment on the part of the university.

Another possible answer to the problem of maintaining currency in curriculum, particularly for a business school, lies in the collaboration between business and education. The communication of what constitutes "state-of-the-art" in the business sector may influence the thinking and allocation of financial expenditures approved by the university administration. In the information systems field, much of what is important to the business sector is also important to the educational sector. Although much of the educational development in information systems involves the mastery of basic competencies, it is also important for the student to gain experience working with current, widely-used hardware and software. Perhaps this business-higher education alliance can help to provide more meaningful learning experiences for students.

THE CORPORATE ADVISORY BOARD

It is quite a challenge to the faculty of business schools in this country to train its future business leaders to be knowledge-current in this age of rapidly evolving technology with its applications to the overall business environment. Many universities feel that the corporate advisory board can help to alleviate at least part of the problem. Advisory boards provide meaningful advice and also provide a communication link with the business community. In a comprehensive 1986 report compiled by the Association for University Business and Economic Research (AUBER) for the American Assembly of Collegiate Schools of Business (AACSB), it was revealed that about fifty percent of AACSB-accredited schools have advisory boards. Furthermore, seventy-five percent of those without advisory boards had plans to start one within the next two years. (3, p. 2) Thus, the notion of a corporate advisory board is one which is rather popular in today's business schools.

Advisory boards exist in several forms. The basic functions of the advisory board are essentially the same, regardless of the form selected. Many schools of business administration have an overall advisory board. Others prefer to have a specialized advisory board for each area of concentration within the school. Many researchers feel that in the technical and scientific areas, advisory boards help professors keep up with technical developments. (2, p. 33) The remainder of this paper deals with the formation and management of a corporate advisory board. The realistic functions and expectations of the board are also discussed.

RATIONALE FOR THE CORPORATE ADVISORY BOARD

There are many advantages to be gained by having a corporate advisory board.

The AACSB-AUBER report cites the following advantages:

- Help establish and maintain contacts with the corporate world;
- Give faculty exposure to business;
- Assist the university in faculty development;
- Bring additional funds to the university in the form of donations; and
- Enhance the reputation of the university with a potentially major source of financial support. (3, p. 1)

In addition to concurring with the above, other potential advantages of the corporate advisory board cited by others who have had experience with advisory boards include:

- Develop and enhance corporate internship programs;
- Conduct career planning seminars; (2, p. 35)
- Serve as a "sounding board" for college programs in executive education, international management and faculty internships in business;
- Identify local and regional business needs for continuing management development programs, and
- Match faculty expertise and resources to business problems requiring research and consultation. (5, p. 9)

The corporate partners also receive benefits. C. Eugene Sturgeon, a partner in Touche Ross & Co., and a member of the corporate advisory board of Illinois State University, feels that it is an honor to be asked to join an advisory board and business people enjoy lending their experience and viewpoints to a prestigious school. Sturgeon also states that, for their service, board members get something in return, such as faculty assistance in helping to solve problems and a good source for recruiting business graduates. Sturgeon continues,

"Being a good corporate citizen takes many forms, but no better expression of this responsibility can be made than supporting higher education." (5, p. 9) By joining with educational interests, corporations increase their influence on the social, economic and cultural life of the community and the country.

It is obviously beneficial to a corporation to hire graduates who are well-prepared. By helping to shape this preparation, business can help mold the knowledge and preparation of its future employees. Typically, corporations invest today for tomorrow's gains. (2, p. 33) By serving on a corporate advisory board, corporate leaders can lend a voice to the preparation of graduates, and by doing so, invest in the future.

FORMATION OF THE CORPORATE ADVISORY BOARD

Numerous benefits are cited as reasons for forming a corporate advisory board. The author's school, Duquesne University's School of Business and Administration, recently formed an MIS Corporate Advisory Board. Certain common guidelines were followed in forming the board. In that Duquesne University is located within five minutes walking distance of the downtown Pittsburgh business district, corporate MIS leaders are readily available. We attempted to gain the participation of the highest ranking corporate officer in the information systems area from each of the corporations we contacted. Duquesne's MIS Advisory Board currently numbers sixteen. Our primary purposes in forming the advisory board parallel many of those cited earlier. We are interested in establishing communications with the corporate community. We want that corporate community to be totally aware of our programs, both undergraduate and graduate. We are interested in the Board's feedback concerning curriculum structure, software and hardware utilization, and perception of the preparedness and competency of our students. We want our students to be fully prepared to cope with the corporate environment which awaits them upon graduation. We believe, as do many others, that corporate executives will care most about our university if they have the opportunity to interact with the people who make us what we are. Good relations begin and end with personal contact. (2, p. 35)

The AACSB-AUBER Survey provides a wealth of guidelines on the process of initiating advisory boards and conducting subsequent activities. This information is the result of the compilation of data from questionnaires returned by the deans of 336 educational institutions, 52% of which had functioning advisory boards when surveyed.

The most important step in the formation of an advisory board is to establish clear objectives and a definition of its purpose. Objectives should be in keeping with the overall philosophy of the school and the university in general. The expectation is that the advisory board will be an active participant in the program. The advisory board should be given a full agenda so as to maximize the benefits of the time they spend with the group. Also, it should be understood from the outset that the co-coordination of the advisory board consumes a great deal of time and effort on the part of the school or department. (3, p. 4) If an individual within the school or department is not willing to commit this time, perhaps it is best not to venture into the formation of an advisory board.

In establishing the objectives for the advisory board, it is important to also keep in mind those things that should <u>not</u> be expected of an advisory board. Examples cited include the exercise of understanding that corporate officials may not be totally aware of procedures in higher education and the fact that as "advisors" the advisory board should not be asked to do everything the administration does not have time to do. (3, p. 4)

When the MIS Advisory Board was formed at Duquesne University, we decided to establish an initial size of fifteen to eighteen members. According to the AACSB-AUBER report, nearly one-half of the advisory boards from the survey ranged between 11 and 20 members. Our initial meeting was a breakfast meeting and our attendance was 100%. That is rather unusual as the average attendance among those surveyed is about 68%. (3, p. 7) This should probably be considered as one plans an advisory board and attempts to anticipate the number of individuals who will typically be present for a meeting.

In addition to having specific objectives for an advisory board, there should also be a formal charter which delineates the roles of the members as well as the definition of the mission. (3, p. 4) The charter can actually be a rather simple document which names the advisory board and lists its areas of involvement.

One additional item to consider in the formative stage of the advisory board is the length of an appointment on the board. Many of the AACSB-AUBER survey respondents recommended a term of three years and staggered memberships. Fixed membership provides the opportunity to eliminate nonproductive members without causing resentment. Of course, the option to re-appoint productive members also exists. The staggered membership concept allows for new members to be added and integrated with veteran members for the sake of continuity. (3, p. 7)

In that the school or department is typically asking the advisory board to participate in the solution of a problem, when the advisory board makes a recommendation, even if the recommendation will not be implemented, the advisory board should be informed of the course of action elected. This area can be rather awkward at times. For instance, the MIS faculty at Duquesne discussed this subject. There will be times when the advice of an advisory board can become too vocationally-oriented. Care must be exercised so that the advisory board does not become so influential that they begin to dictate curriculum. This can result in a "trade school" approach. The discretion of the faculty in making final decisions on curricular matters must prevail.

At the initial meeting of the MIS Advisory Board of Duquesne University, one of the agenda items was entitled "The Role of the Advisory Board," and the speaker was William P. Buckley of the Aluminum Company of American (ALCOA), who is currently serving as chairman of the Ohio State University MIS Advisory Board. The Ohio State MIS Advisory Board was formed in 1982 and thus has experienced, first-hand, a variety of situations.

Mr. Buckley provided some excellent advice for the ongoing operation of an advisory board. As advocated by nearly all of the research and reporting done on this particular topic, Buckley suggested the formation of a clear "mission statement". He shared the following mission statement as adopted at Ohio State:

... to establish a dialogue with the faculty of the College in the current application of information technology in business. The emphasis of this dialogue is on the communication to the faculty of current issues, practice, and problems within business. The Board is responsible for familiarizing itself with the programs, organization, and objectives of the College in order to carry out this advisory role. (1, p. 2)

Other sound advice included the maintenance of diversity of backgrounds and industries of board members. This, of course, encourages a broader-based viewpoint in discussions of what is important. Also encouraged was the availability of manageable, specific assignments prior to the actual advisory board meeting. Aligned with this point is the need for the MIS faculty to be involved in providing continuing new challenges to maintain board members' interest. Since it is important for the advisory board to interact with all of the constituents of the MIS department, it was also suggested that student presentations be incorporated in the meetings of the advisory board. Buckley also contends that, even though the advisory board may at some point be helpful with fund raising, fund raising should not be mixed with the proceedings of the advisory board. (1, p. 1)

The Ohio State MIS Advisory Board has selected four areas for specific assignments, namely architecture (defined as the review of college needs), curriculum, internship sources, and public relations. Members of the Board, which numbers fourteen, select one of these areas on which to concentrate their efforts.

To those venturing into the establishment of an advisory board, it is invaluable to listen to the experiences of one who has served successfully elsewhere. Not only does it serve to provide guidelines for operation, it also provides optimism for what can be accomplished through the successful management of an advisory board. If handled properly, however, the corporate advisory board experience can be beneficial to both the corporate executive and the academic entity.

The AACSB-AUBER report does an excellent job of summarizing guidelines for the development of a corporate advisory board, presents guidelines for conducting the meetings of the advisory board, and suggests possible subcommittees for the advisory board. A partial list of guidelines for developing the advisory board follows: (3, p. 15)

- Get top level administration involved
- Set objectives carefully
- Determine the decisions the corporate advisory board will influence

- Present challenging problems
- Communicate
- Select and motivate people who are interested in the program and the university
- Deal with meaningful projects
- Provide rapid turn-around by acting on advisory board results quickly

The following is a summary of meeting guidelines as presented in the AACSB-AUBER Report. (3, p. 14)

- Use effective, comfortable meeting room arrangements. Move to a clear workspace after the meal of a luncheon meeting.
- Make every effort to make the members feel welcome.
- Introduce any new members or faculty guests at the beginning of a meeting
- Begin on time and follow an agenda. Distribute a copy of the agenda at the beginning of the meeting or mail it with the meeting notice.
- Conduct the meeting so as to maintain interest and accomplish its purpose within the allotted time.
- Eliminate outside distractions.

Finally, the following list represents possible Corporate Advisory Board Subcommittees as presented by the AACSB-AUBER Report. (3, p. 13)

- Academic Standards
- Admissions/Recruitment
- Career Advisement
- Curriculum Development
- Executive-in-Residence
- Faculty Development
- Financial Planning
- Fund Raising
- Internship Programs
- Placement
- Public Relations
- Student Aid

Keep in mind that the above list represents compilations of responses from the deans of those schools and/or programs which have existing advisory boards. The lists therefore represent items designated as the result of the experiences or areas of concern of other advisory boards. The AACSB-AUBER Report is a very comprehensive summary of the survey results of those educational institutions experienced in managing the activities of an advisory board.

SUMMARY

This paper has summarized the overall benefits of a corporate advisory board. The importance of communication and collaboration between higher education and the corporate community is emphasized. Benefits to both the educational and the corporate entities from advisory board participation are enumerated. Next, this paper attempted to provide meaningful guidelines for those forming an advisory board. Finally, it reported on guidelines for conducting meetings and indicated possible subcommittee divisions for the advisory board. By following a few sounds suggestions from those with experience, the educational entity can make the advisory board a more rewarding experience for all concerned.

REFERENCES

1. Buckley, William P. "The MIS Advisory Board at Ohio State University," Address at Duquesne University School of Business and Administration MIS Advisory Board Meeting, February 3, 1988.

2. Cummings, Peter. "Vesting the Interest: Corporate Involvement Leads to Financial Commitment," Case Currents, November, 1981, pp. 32-35.

3. Gnuschke, John, Rene Elicano, and Marilyn Helms. "Business Advisory Councils: An Analysis of a Survey of AACSB Colleges of Business," A Survey Prepared by the Association for University Business and Economic Research (AUBER) for the American Assembly of Collegiate Schools of Business (AACSB), May, 1985.

4. Healy, Timothy I. "Higher Education and Business: A Proven Partnership." Educational Record, Summer 1979, pp. 272-281.

5. Sturgeon, C. Eugene, "The Advisory Council: It's Working for the College of Business" Illinois State University Business News and Views, Fall 1985, pp. 9-10.

SOFTWARE SOURCES
FOR THE SMALL COLLEGE WITH A SMALL BUDGET

Diane Murphey, CDP
Assistant Professor of CIS

Pamela Nelson
Assistant Professor of Mathematics
Panhandle State University
Goodwell, Oklahoma 73939

This paper reflects the efforts of two faculty members over a period of six years to acquire software necessary to keep an Information systems curriculum viable. Sources covered range from publishers and discount firms to donations by private sources and public domain software. Although there are undoubtedly additional resources not covered in this presentation, these are the ones that have been used to build a software library at Panhandle State University.

INTRODUCTION

Teaching CIS in a small University with a limited budget is a challenge anywhere, but if you add a remote rural location, it forces creativity on the part of the faculty and makes software piracy attractive, as well as seductive. In an effort to remain professional despite the tempting barrage of illegal copies, the small college with an ever-shrinking budget must find alternative means of legal acquisition of quality sofware. This paper reflects the efforts of two faculty members over a period of six years to acquire software necessary to keep an Information Systems curriculum viable. Although there are undoubtedly additional resources not covered in this presentation, these are the ones that have been used to build a software library at Panhandle State University.

SOFTWARE PUBLISHERS AND DISCOUNT FIRMS

Many publishers have educational prices and/or student versions of their software. Generally, student versions have been adequate for teaching purposes. Because they are limited in scope, the instructor must spend time testing the program functions in order to avoid the "OOPS Syndrome." A variety of textbook publishers provides educational versions at little or no cost. Making copies for lab use is authorized but is time consuming and adds the burden of security. Recently, publishers have been bundling software with each textbook, placing responsibility for its care with the student. This normally adds about ten dollars to the text price, a good value if the instructor's time is limited.

An attractive piece of demonstration software is EXSYS by Expert Systems Inc. For $15, a workable expert system which handles up to twenty-five rules gives students a taste of the real thing. The software comes with a manual and permission to copy as needed. The full system can be purchased at a discount if desired, but the demo gives a good "tip of the iceberg" exposure for undergraduate students.

An alternative to using student versions is to buy the package at a special educators' price. This gives the added advantage of full function packages at a fraction of the retail cost. One company with such a pricing structure is WordPerfect Corporation. They offer WordPerfect 4.2 to qualifying faculty and students at $125.00, as opposed to the retail price of $495.00. Site licenses are another method for providing quality software at a reduced cost. Site licenses allow the institution to buy multiple copies of the same piece of software or the right to copy with specific parameters at a cost less than purchasing individual copies. Companies offering discounts and/or site licenses are Apple Computer Corporation, Tandy Corporation, Egghead Discount Software, Microsoft Corporation, Campus Technology Products, Microrim, and others.

Companies will also offer upgrades of existing software to current owners, provided they have turned in the registration card that comes with the packages. Upgrade offers come only to registered owners and are more reasonable than purchasing the new version. Neglecting to mail the warranty card that comes with the software can be a costly oversight.

A few educationally minded companies are willing to donate copies of their software for classroom use. One such company is Index Technology. Through their educational grant program, colleges and universities with qualifying courses can be awarded copies of Excelerator, a program which automates many of the tedious tasks of systems analysis. This program, with a list price of $8,000.00, would be out of the reach of most software budgets for the next century. Although there are companies, such as AT&T, who have grants for large universities, Index Technology has made quality technology available to small schools.

In a small college environment, computer faculty are often utilized in teacher education classes. Panhandle State University has a continuing education program for elementary and secondary teachers during the summer months. Many of the short courses offered deal with computer literacy, computerized teaching tools, LOGO, word processing for teachers, and subject specific courses such as Using Computers in the Math/Social Studies/English/etc. Classroom. When teaching methods classes, there is a tremendous amount of software available for demonstration purposes at no cost or for a small transportation or workshop fee. These packages are sent on approval or loan and can be used to stretch the budget and to get a much better selection of teaching packages. The software can be used for two to four weeks and normally takes two to three weeks for delivery. You must plan ahead to coordinate your need for the software and its availability. However, the companies have been reliable on shipping dates and sending what was requested. Companies that have allowed us to use their software for demonstration purposes are Sunburst, K-12 Micromedia, Cambridge Development Laboratories, and Scholastic, Inc.

PRIVATE SOURCES

An often ignored source of software is the local community. Businesses and alumni interested in your program at the college are many times willing to donate their "pet" packages that they would like to see taught to prospective and current employees. Also, people in the local area may have software and hardware that are not being utilized. Many times this surplus can be used by the college and be a tax deduction for the donor. Although it is true that many people will attempt to give you junk, sometimes even junk can be useful. Panhandle State University accepted the donation of an older NCR minicomputer. The machine was not what we would have chosen but the donation included a COBOL code generator and a wide range of accounting and inventory applications used in a small business. This software made an obsolete machine useful for independent study projects by advanced

students. It is not maintained, and the donor understands that when it cannot be repaired, it will be hauled to the dump. In the meantime, we have a piece of equipment for our upper level students to use that allows them to gain operations experience as well as exposure to hardware and software that would otherwise not have been available. Relationships with local donors can be developed through Adult and Continuing Education classes as well as community projects.

Another source of funds is local, regional, and national foundations. In 1985 over 4,000 foundations existed with assets of $63.8 billion. These foundations gave close to $5 billion in grant money, 25% of which went to universities and schools (The Foundation Directory, 1985, Foundation Center, New York, New York). Foundation giving and assets are growing. For example, in the state of Oklahoma, foundation assets have grown 272% since 1972 while grants awarded have grown 182%. Some reference books that may aid you in finding information are Foundation Directory, Foundation Grants Index, Foundation Fundamentals, A Guide to Grantseekers, and Getting A Grant in the 80's. These may be obtained from The Foundation Center, New York, New York. Often the larger state universities will offer courses in grantsmanship that could be helpful in developing proposal writing skills. A good course for grant seekers is taught by Dr. Don Udell, University of Oklahoma, Norman, Oklahoma.

PUBLIC DOMAIN SOFTWARE

Although there exists a great store of public domain and user supported software, beware!! Many individuals are willing to give software that is worth just what was paid for it - nothing. Another problem with software anonymously acquired through such sources as bulletin boards is the "vandals of the 80's." Their twisted minds find delight in leaving attractive software which, when used, leaves your computer system infected with a virus capable of destroying programs and data.

One source of public domain software that is relatively safe is your state Department of Education. Many states maintain a library of software available free of charge to schools in the state. Although much of this software is designed to be used in elementary and secondary school settings, some is useful in college introductory classes. Much of the software could be utilized in teaching Computer Literacy and Methods courses for education majors at all levels. In addition to the state level, local school districts often have a collection of public domain software. As part of a grant received to improve computer teaching in local schools, Panhandle State University surveyed elementary and secondary schools within 100 miles collecting lists of all software owned by each school. The information was compiled into a catalog indexed by subject and machine type. The catalog also contains an entry for contact person, school, publisher, and public domain status. This catalog was then distributed to the area schools as a reference tool to enable them to share expertise with others who use the same software as well as a reference to consult when considering purchase of a particular piece of software.

Another avenue for finding public domain software is contacts through meetings and professional organizations. Become part of the "good ole boy/girl network." Summer computer institutes are an inexpensive, relatively painless way to become part of a network or start your own as well as improve your teaching skills. Some of the institutes our faculty have attended are sponsored by Boyd and Fraser Publishing at Memphis State University and by Interface Magazine and Mitchell Publishing at James Madison University in Williamsburg, Virginia and Central State University in Edmond, Oklahoma. Often area universities will have short courses that can be gleaned for software sources as well as teaching techniques. Interface, in

cooperation with regional universities, sponsors one day seminars that give an overview of emerging areas in IS curriculum. These, too, are a good opportunity to develop contacts. The key to a good network, however, is you. Don't be afraid to ask for information and advice from colleagues at other universities, the solution may be only a question away.

Combining departmental resources into one or two central micro labs is a way to stretch existing resources. Software can then be purchased for the lab thus allowing more than one department to utilize the same programs. At Panhandle State University one central lab is shared by IS, Agriculture, Chemistry, Physics, Education, and Business. This type of situation does take a reasonable amount of cooperation between the participants, but the key is one faculty person in charge of the lab operation and scheduling. The lab is open, but continually staffed by a trained IS major. Their job is the most crucial to the success of a shared lab environment. They must assist nontechnical users as well as the introductory level IS students. Professors in other departments utilize the lab primarily for computer assisted instruction. They give the lab a copy (never the only copy nor the original!) of the software they wish their students to run. The lab staff handles all instruction in how to use the software, maintains a sign-up sheet if requested by the instructor, and assists in obtaining any printouts required but is not expected to know the subject involved. Although it takes a bit of finesse, the increased funding available makes a shared lab environment well worth the effort.

CONCLUSION

By using every available source, scrounging in every corner, begging when necessary, and crying and gnashing teeth, it is possible to have software adequate to meet your requirements. Great perseverance and creativity are needed to keep your curriculum current in this ever changing and growing market. In this paper, we have shared some of the techniques and tricks we have used to find teaching materials and software while living under restricted budgets. Perhaps these will also work for you.

HEALTH CARE INFORMATION SYSTEMS: THEIR DEVELOPMENT AND IMPACT ON PATIENT CARE

by Michelle L. Walters* and George Fowler**

Texas A&M University, Department of Business Analysis and research, College Station, TX, 77843, * teaching assistant and ** professor, (409) 845-1616, and (409) 845-7946, respectively.

ABSTRACT

The health care industry faces severe challenges. Government and insurance requirements and the demands for higher quality patient services are all issues that have forced health care providers to look to information systems. Through an HIS, the industry can provide all required reporting and at the same time improve patient care.

This paper examines the development of HISs. The authors also looked at a current system in order to provide a perspective of where HISs are today. The authors then examined a proposed system. This system attempts to solve the shortcomings of prior systems. A descriptive analysis is presented along with a proposal for future investigation.

HIGH COST OF HEALTH CARE

Health care expenditures in the U.S.A. have increased to 322.4 billion dollars annually or 10.5 % of GNP (Thomas and Davis 1987). The health care industry spends 4.55 billion dollars on information systems alone. Of this, 4 billion dollars is spent on hospital information systems (HISs) and the remainder spent in the automation of physicians' offices, clinics, etc. (Kennedy 1987). Hospitals currently spend roughly 50% of their HIS dollars on patient accounting systems. Hospitals must compete for health care dollars and information systems will offer them a competitive advantage. This advantage may be realized in two ways: cost reduction and/or improved patient service.

THE NEED FOR CHANGE

Expenditures by hospitals on HISs are rising as old, inadequate systems are replaced by and improved new ones (Kennedy 1987). Much of the impetus behind these changes is a strong need to reduce costs as hospitals are forced to compete (Morris 1986). The health care industry is no longer a revered institution serving humanity, immune from scrutiny. Consumers are demanding more and better service and at a competitive price.

Insurance companies' and the Federal Government's demands for more and better records are also having an impact on hospitals. Most of these demands have developed since 1980, and cannot be handled by the old HISs which were designed for "operational fire fighting instead of support capability, cost effectiveness, and information availability and quality" (Morris 1986). Information is now being recognized as a vital resource in the fight for corporate survival. Systems that provide for information resource management, like HISs, will play a major role in health care.

H.I.S. PRIORITIES

The top priorities of HIS administrators today are: ease of database modification, quality of training provided by the vendors, ability to update and revise the database, and the ability to interface accounting data and systems with other financial and patient care systems (Kennedy 1987). Furthermore, the priorities include the ability to gather information from many different sources and to provide users with timely, accurate and tailor-made reports (Morris 1986) .

HIS administrators also face an increased demand by clinicians to provide support in new areas, such as networked information systems that connect clinics and doctors with their associated hospital. Currently, dissatisfaction among HIS personnel is high because their needs are not now being met. Current systems are not responsive to the user. They are typically not easy to use, particularly for the novice computer user.

One promising new system described in the literature is "the Medical Gopher", so named because it is capable of retrieving, organizing, collecting and reviewing data. This type of work has been termed "go for" work, and has occupied too much of physicians' time in the past (McDonald and Tierney 1986). The Gopher was designed to save the physicians' time and reduce oversights. So far, the system has been used by 200 physicians as a micro-computer based test ordering system. Each of eight IBM-compatible micro-computers were connected to a central hospital system. The physicians have used the system for two years and "find it acceptable, but not preferable, to manual ordering" (McDonald and Tierney 1986). The system was written mostly in R-BASIC and its characteristics include: mixing of free text and code; query and report writing facilities; word processing capabilities; a mouse to enter data; menus ; and fixed form orientation (McDonald and Tierney 1986).

THE "PAINT SCREEN" SYSTEM

The current work involves development of HIS software for the easy, accurate collection and manipulation of patient data by nurses. The software is easy to use; training takes less than an hour. The system designer is able to design or paint a data entry screen according to the exact specifications of the nurses, including the design and use of pop-up menus, different color combinations, use of mouse, keyboard or voice for data entry, size and location of data entry fields, choice of descriptive rather than cryptic variable names, choice of data entry types and pattern checks to ensure accurate entry.

The ease and flexibility of screen design is a major advantage of the proposed Paint Screen System. Ashby (1985) has shown that structured forms are not helpful and are even "antithetical to the goal of complete data capture". Furthermore, after data for each patient is collected using the flexible screens, it will be stored in a relational database. Storing the data in a relational data base provides for easier database maintenance and faster retrieval. Both of these are vital issues. But, probably more important is the fact that by reducing the time involved in data capture and retrieval, a nurse has more time for the patients. This increased nurse-patient time should lead to better health care service for the patient.

RESEARCH ISSUES

Any new HIS should not be accepted at face value. It must be tested to see if in fact it does improve the health care environment. In particular, the system must provide significant advantages over existing techniques and systems. Several research hypotheses have been identified by this research:1) Does the Paint Screen system enhance data capture? 2) Will the quality of patient service be improved? 3) Is the HIS improved because of Paint Screen? These hypotheses open up other questions that need to be studied. These questions deal with the issues of human factors. Is the system user friendly? Are the screen designs appropriate for their intended use - form filling, question/answer, menu or some combination? Does the system allow for both novice users and experienced users? Is the incidence of data errors reduced? Is the quality time of the health care professional increased? All the questions must be answered in order to truly judge the proposed system's benefits. However, a descriptive analysis of features can be accomplished. Figure 1 demonstrates that many of the comparisons fall more favorably on the side of the proposed system, Paint Screen.

The proposed system will be tested in two ways: 1) Health care professionals will be asked to use the system and answer a detailed questionnaire about it. This will be accomplished by testing in several hospitals and at a medical school (not at the same location). 2) A simulation model will be developed for a micro-computer that will use Paint Screen with a shell behind it. No actual patients will be involved. This simulation will provide the means for testing communication protocols - menus vs. question-answer vs. form filling. It will also provide the means for capturing error rates. These data can then be analyzed statistically and provide insights into the benefits of new HISs like Paint Screen.

SUMMARY

There is a strong need for fast, easy, accurate, flexible data entry software in the health care industry. Large sums of money have been spent and will be spent to try to meet the needs of health care professionals as they deal with patients, competition from within the field, increased government and insurance company requirements and rising costs. Current systems like the Medical Gopher were discussed to provide a basis for understanding the development and status of existing HISs.

A new system was examined. This system is intended to handle the data entry needs of nursing professionals today. This system includes many of the features the

health care professionals require and say are missing in current systems. An analysis of the system's features tend to support the perception of improvement. However, several research issues were presented. This issues must be answered statistically. This will be done with the ongoing research by the authors.

REFERENCES CITED

Ashby, R.O., 1985, Structured vs. unstructured encounter forms: a comparison, IEEE Proceedings of the Ninth Annual Symposium on Computer Applications in Medical Care, 7-11.

Kennedy, O.G., 1987, Information systems: a status report, Hospitals, 8, 61.

Korpman, R.A., 1985, Patient care information systems: looking to the future, Part 5. The integrated information system, December/January, 2, 56-63.

McDonald, C.J. & Tierney, W.M. 1986, The medical gopher - a microcomputer system to help find, organize and decide about patient data. The Western Journal of Medicine, 145(6), 823-829.

Morris, D.C., 1986, Information systems: the direction of things to come, Healthcare Financial Management, 6, 29-37.

Thomas, D.R., & Davis, K.M., 1987, Physician awareness of cost under prospective reimbursement systems, Medical Care, 25(3), 181-184.

Figure 1. Comparison of Paint Screen to Current Systems

CHARACTERISTICS	CURRENT SYSTEMS	PAINT SCREEN
TRAINING	POOR	GOOD
DATA BASE:		
- DEVELOPMENT	COMPLEX	EASY
- MAINTENANCE	COMPLEX	EASY
- MODIFICATION	RELATIVELY EASY	EASY
INTERFACE WITH OTHER SYS.	POORLY	NOT TESTED
TAILOR MADE REPORTS	NOT PROVIDED	PROVIDED
USER FRIENDLY/EASY TO USE	FAIR	GOOD
ERROR RATES	BENCHMARK TO BE SET	NOT TESTED
SCREEN DESIGN:		
- FLEXIBILITY	NONE	HIGH
- EASE OF	POOR	GOOD TO EXCELLENT
- MENU	SOME	YES
- FORM FILL	MOST USED	YES
- QUESTION-ANSWER	SOME	YES
DESCRIPTIVE DATA NAMES	POOR	GOOD
ERROR CHECKING ON SCREEN	NO	YES

THE CHANGING PROFILE OF THE STUDENT ENROLLED IN THE INTRODUCTORY COMPUTER (LITERACY) COURSE

Jean Buddington Martin, Assistant Dean
Computer and Office Systems and Engineering Technology
Florida Community College at Jacksonville
Jacksonville, Florida 32202-4030

Kenneth E. Martin, Director
Division of Computer and Information Sciences
The University of North Florida
Jacksonville, Florida 32216

Abstract

Having noted the diversity of skills of the entering students in the introductory computing course the authors developed and administered a survey to identify certain characteristics of students (Martin & Martin, 1986). It was hoped that the results might provide useful data to be factored into the design and implementation of the course. Two years later, at the beginning of the 1987 Fall term, the authors again administered the same survey. The intent was to identify any changes in student characteristics that had taken place since the original administration of the instrument (Martin & Martin, 1987). The purpose of this paper is to isolate differences and similarities of the entering introductory students of 1985 and 1987. An analysis of the results of the two surveys indicates that possible areas of concern include: a leveling off of the number of students entering the course who have had a previous computer course; the development of a "gender gap" regarding the use of computers; and the challenge of meeting the needs of the diverse student population in the course.

Computers are now entrenched in almost all aspects of American life. This transition has taken place at a faster pace than has any other technological advancement in this nation's history (Adams & Fuchs, 1986). This rapid evolution carries with it tremendous challenges for people administering and teaching almost all computing courses (Fritz, 1985). The introductory service course--often referred to as the "literacy" course--has probably undergone more changes in a shorter period of time than any other component of the computing curriculum. The increase in student familiarity with the technology prior to entering the course has provided another major impetus to the challenge of planning and implementing this course (Turner, 1987). Many of the students have had previous computing experience (Gilbert & Green, 1986) and have definite opinions about the preferred content of this course (Martin & Martin, 1986).

Many facets of the course have been controversial. Perhaps nothing has received more attention than the practice of referring to it as a computer literacy course. Because a substantial change that has taken place in the content is the inclusion of instruction in the three major genres of software, word processing, electronic spreadsheet, and data base management, it can be argued that the term literacy no longer applies. The content has been enlarged greatly beyond that which is implied by literacy. The term computer competent suggests a higher gradation that may provide a more accurate definition of the course: "To be computer competent implies being able to use the computer as a tool professionally" (Arden, 1986, p. 27). Certainly the computer is a tool that can be used to extend our mental capabilities (Merrill et al., 1986). The applications of the tool are now a major emphasis of the course. This distinction is important because it reflects the emerging profile of increasing competence of many of the students entering the course.

It is particularly critical that progress in this course be monitored because frequently this will be the only computing course that students will be exposed to in their college careers. The design and implementation of the course content must keep pace

with societal and technological changes in order to remain a viable component of the curriculum. In order to assess the characteristics of the students entering the introductory course, the authors designed and administered a survey at the beginning of the fall semesters of 1985 and 1987.

The survey was administered to all students in the introductory course at Jacksonville University and the University of North Florida. The authors decided to use the term traditional student to refer to those who were twenty-one years of age or younger. Everyone else was considered to be a nontraditional student. The following are the results (1985-1987).

PREVIOUS COURSES

1. Fifty percent of the groups, both in 1987 and 1985, who were 21 years of age or younger (traditional) had previously taken a computer course either in high school, college, or industry. Of the 1987 group who were 22 years of age or older (nontraditional) 37.6% had previously taken such a computer course compared to 31% (21.3% growth rate) in the 1985 classes (not significant).

2. The phenomenon observed in number 1 was not surprising, but the authors then attempted to push the analysis one step further to determine if students 17 or 18 years of age had more classroom experience with computers than their counterparts in 1985. Of the students who are 17 or 18 years of age 59 to 46% had previously taken a computer course either in high school, college, or industry, versus about 50 (in each year) for 19 and 20 year olds.

PREVIOUS PROGRAMMING EXPERIENCE

3. Between 46 and 43% of traditional students have previously programmed in some language (usually BASIC), versus 35 to 32% of nontraditional students (both are slight decreases).

PREVIOUS WORD PROCESSING EXPERIENCE

4. Of the 1987 traditional students 51.4% had previously used a word processing package while 37% of their 1985 counterparts had done so (38.9% growth rate and a significant difference). Of the 1987 nontraditional students 47.4% had used word processing while 32% of the 1985 nontraditional students had done so (48.1% growth rate and a significant difference).

5. Of the 1987 females 50.2% (includes both traditional and nontraditional) had previously used a word processing package while 39% of their 1985 counterparts had done so (28.7% growth rate and a significant difference). Of the 1987 males 49.2% (includes both traditional and nontraditional) had used word processing while 30% of their 1985 counterparts had done so (64% growth rate and a significant difference).

6. Between 45 and 53% of the female, nontraditional students had previously used a word processing package, versus corresponding figures of 31 and 43% for male nontraditional students (neither difference is significant). Note also that 55.4% of the 1987 traditional males had previously used a word processing package while 30% of their 1985 counterparts had done so (84.7% growth rate and a significant difference). Also 48.4% of the 1987 traditional females had used word processing while 34% of their 1985 counterparts had done so (42.4% growth rate and a significant difference).

PREVIOUS SPREADSHEET EXPERIENCE

7. Of the 1987 nontraditional students 26% had previously used a spreadsheet package while 27% of their 1985 counterparts had done so. The difference is of course not significant. However 16.7% of the 1987 traditional students had used a spreadsheet while 7% of their 1985 counterparts had done so (138.6% growth rate and a significant difference).

8. Between 19% and 25% of males had previously used a spreadsheet, versus 14 to 17% of females (neither difference is significant).

9. Between 14 and 28% of female, nontraditional students had previously used a spreadsheet, versus 28 to 24% of male nontraditional students (neither difference is significant).

PREVIOUS DATABASE EXPERIENCE

10. Between 25 and 18.5% of the nontraditional students had previously used a database package, versus 16 to 21% of the traditional students (neither difference is significant).

11. Between 22 and 26% of males have previously used a database, versus 19 to 15% of the females (neither difference is significant).

12. Between 24 and 28% of female, non-traditional students and male, non-traditional have used a spreadsheet (both in 1987 and 1985).

RETENTION FROM PREVIOUS COURSEWORK

The authors next explored whether prior computer courses have meant exposure, and hence, retention of certain ideas.

13. "I recognize the difference between micro computer (PC) peripheral equipment (disk drive, printer, etc.) and mainframe (large computer) peripheral equipment--yes or no ? About 70% of students who have taken computer courses retain this knowledge, while about 40% of the others know the difference.

14. "I am familiar with the difference in processing speed and ability of mainframe computer versus micro computers--yes or no ?" About 45% of students who have taken computer courses recognize the difference, while about 20-25% of the others recognize the difference.

15. "I know the difference between primary (main) memory and secondary (auxiliary) memory--yes or no ? Between 60 and 51% of the students who had previously taken a computing course recognized the difference (not significant), versus figures of 30 and 6.4% for students who had not taken a previous course (significant).

The next series of statements dealt with attitudes and expectations from the course the students were about to take. Students were asked to circle one of five choices for each statement: strongly agree, agree, not certain, disagree, or strongly disagree.

ATTITUDES AND EXPECTATIONS

16. "I am comfortable using a computer." Between 52 and 49% of the nontraditional students agreed or strongly agreed, versus 44 to 46% of the traditional students.

17. "I think it is important to learn how to write a computer program." Of the 1987 traditional students 80.6% agreed or strongly agreed, while 89% of the 1985 traditional students felt the same (significant). Of the 1987 nontraditional students 76.3% agreed or strongly agreed while 80% of their 1985 counterparts felt the same (not significant).

18. "I think it is important to learn how to use packaged software (e.g. spreadsheets)." About 85-90% of students agree with this statement (1987 and 1985).

19. "I think 'hands on' experience is desirable in a course like this one." Over 90% of 1987 and 1985 students agree with this statement.

20. "I think it is important to learn the history of computers." Of the 1987 traditional students 37.4% agreed or strongly agreed while 50% of their 1985 counterparts felt the same (25.2% decline rate and a significant difference). Of the 1987 nontraditional students 48.6% agreed or strongly agreed while 56% of their 1985 counterparts felt the same (not significant).

21. "I think it is important to learn computing terminology." Over 90% of 1987 and 1985 students agree with this statement.

22. "I think computers are/will be important in my field of employment." Between 85% and 95% of 1985 and 1987 students agree with this statement.

23. "I think computers are/will be important in my personal (home) life." Between 65 and 57% of the traditional students agreed or strongly agreed (significant at .10 level), versus 56 to 71% of the nontraditional students (significant at .05 level).

OVERALL CHARACTERISTICS OF STUDENTS

24. Of all 1987 students 49.6% had used word processing previously while 34% of their 1985 counterparts had done so (45.9% growth rate and significant).

25. Of all 1987 students 20.8% had used spreadsheets previously while 17% of their 1985 counterparts had done so (22.4% growth rate and significant at the .10 level).

26. Of all 1987 students 20% had used a database previously while 21% of their 1985 counterparts had done so (not significant).

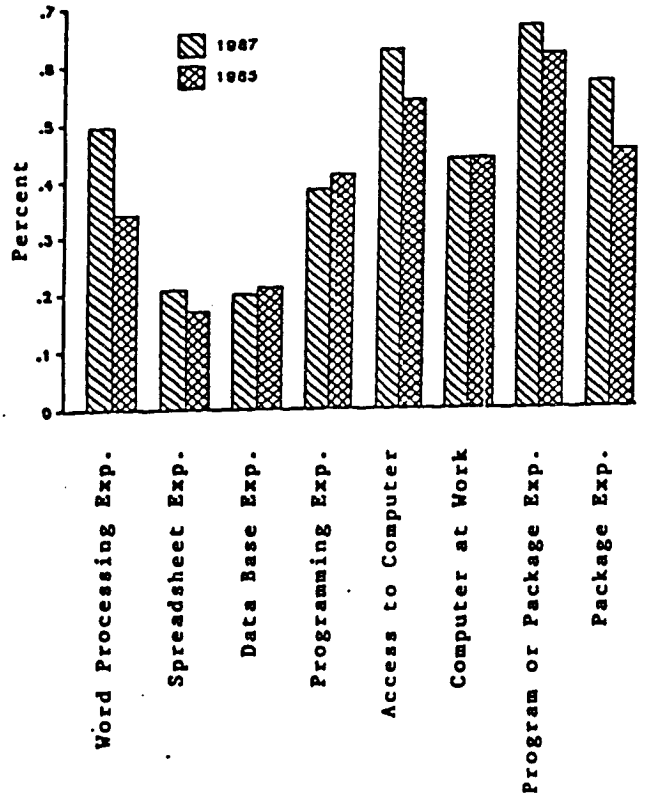
27. Of all 1987 students 38.5% had programmed previously while 41% of their 1985 counterparts had done so (not significant).

28. Of all 1987 students 62.8% have a computer in their home or at their place of employment while 54% of their 1985 counterparts had one (16.3% growth rate and significant).

29. Of all 1987 students 43.8% have a computer at their place of employment while 44% of their 1985 counterparts had one (not significant).

30. Of the entire 1987 sample 66.8% HAVE programmed or used a package previously while 62% of the 1985 sample had done so (significant).

31. Of the entire 1987 sample 57% HAVE previously used one of word processing, spreadsheet, or a database while 45% of the 1985 sample had done so (26.7% growth rate and significant).



IMPLICATIONS

Because the sample in this study may not be typical of all college students and based upon self-report, caution should be exercised when drawing conclusions from the data. However, certain areas of concern emerged as the results of the survey were analyzed.

First, the percentage of students who have been exposed to a previous computer course has not increased. In fact, the 17 or 18 year old students in 1987 seem to have had less exposure than their counterparts of two years earlier. This finding seems particulary inconsistent with the increase of computer labs in the secondary schools. The question arises as to whether this is merely an aberrant condition existing only in the sample or does this finding suggest that a plateau has been reached. It should be noted that the percentage of 1985 students reporting prior computer instruction was consistent with the results reported by Gilbert and Green (1986). The authors have pondered about this question but feel it would be inappropriate to do more than present the idea of a leveling off of computing experience as a possibility.

Second, what may be interpreted as an extremely disturbing condition may exist. In the 1985 study, a greater percentage of females than males had previously used a word processing program. In 1987 that difference disappeared. However, a new phenomenon seems to have materialized. Approximately 41% more males than females have previously used a spreadsheet and approximately 71% more males than females have used a data base. Spreadsheet and data base programs can be important tools for administrative decision making and can impact upon professional competence (Fleischer & Morell, 1985). Thus, to be competitive in many career choices it is critical to be well grounded in the fundamentals of the use of these tools. There is considerable evidence that a gender gap has developed regarding the use of computers among children and it matters for several reasons. It matters because there is tremendous educational value in the use of computers. It matters because a certain amount of knowledge about computers is required in our role as citizens. Advances in technology have been accompanied by concerns over privacy rights, computer crime, and changing employment patterns. Students who have an aversion to the technology are likely to become adults who are the most fearful and view the computer as a threat rather than a tool. Finally, it matters because of the occupational consequences. It has been documented that girls are not taking advantage of opportunities to use computers available to them in school. This has been particularly notable in middle and high school grades (Sanders, 1986).

> And boys are learning about and using computers more in the schools than girls. Once again, the schools are becoming a breeding ground for a pattern of opportunity based on sex, race and class rather than on abilities. For the young women of tomorrow's 'high tech' world, missed opportunities carry a potentially double threat-for them and for their children (Lipkin & McCormick, p. 14).

Other studies support the finding that there is a gender difference between students (Sproull, Zubrow, Kiesler, 1986; Pinto, 1986) and discuss the impact of computer aversion (Meier, 1985; Fritz, 1985). Obviously these are important considerations and this study seems to support a trend towards a greater split between males and females regarding the use of computers.

Third, the diversity of the student population adds substantially to the challenge of teaching the introductory course effectively. Approximately 50% of the students have had prior computing experience. However, that means about half of the students will have had only limited exposure to computing. Further, the students enter the course with varying levels of apprehension and interest in certain components of the course--history of computing for example.

The professor, through instructional innovation, is frequently the primary reason that so many students learn how to use computers and emerge with a positive attitude about the technology (Adams & Fuchs). Too, teachers often have a tremendous effect on students because they often serve as role models. "Research on women professionals in science and mathematics careers has found that nearly all of them pinpointed a close relationship with a supportive teacher as the most crucial turning point in their student careers" (Lipkin & McCormick, 1985, p. 16).

Stuckman (1985) developed a model for teaching computer literacy. His model recognizes the need to incorporate techniques for addressing student anxiety and assess the person's readiness to learn. Clearly, to meet the challenges inherent in the course one must not only posses the requisite knowledge of the course material, but also be able to apply strategies appropriate for a highly diverse group of students.

References

Arden, E. (1986, Spring). Beyond computer literacy. The College Board Review, 27-28.

Adams, D. M., & Fuchs, M. (1986). Educational computing. Springfield, Il: Charles C. Thomas.

Fritz, J. M. (1985). Rethinking computer literacy. In K. A. Duncan & D. L. Harris (Eds.), Computers in education (pp. 681 - 685). Norfolk, VA: Proceedings of the IFIP TC 3 4th World Conference on Computers in Education.

Gilbert, S. W., & Green, K. C. (1986, May/June). New Computing in higher education. Change, 33-45.

Lipkin, J. & McCormick, L. M. (1985). Sex bias at the computer terminal-how schools program girls. The Monitor The Association for Educational Data Systems, 24, 18-20, 26.

Martin, J. B., & Martin, K. E. (1986). A profile of today's computer literacy student. SIGCSE Bulletin, 18, 27-33.

Martin, J. B., & Martin, K. E. (1988). A profile of today's computer literacy students: an update. SIGCSE Bulletin, 20, 235-239.

Meier, S. T. (1985). Computer aversion. Computers in Human Behavior, 1, 171-178.

Merrill, P. F., Tolman, M. N., Christensen, L., Hammons, K., Vincent, B. R., & Reynolds, P.L. (1986). Computers in education. Englewood Cliffs, NJ: Prentice-Hall.

Sanders, J. (1986, January/February). The computer gender gap: Close it while there's time. The Monitor The Association for Educational Data Systems, 24, 18-20, 26.

Sproull, L., Zubrow, D., & Kiesler, S. (1986). Cultural socialization to computing in college. Computers in Human Behavior 2, 257-274.

Stuckman, R. (1985). A model to develop computer literacy in education. In K. A. Duncan & D. L. Harris (Eds.), Computers in education (pp. 681 - 685). Norfolk, VA: Proceedings of the IFIP TC 3 4th World Conference on Computers in Education.

Turner, J. A. (1987, July 22). Familiarity with new technology breeds changes in computer-literacy courses. The Chronicle of Higher Education, pp. 9, 12.

Maintaining Pass Rates and Academic Standards in the Face of Declining Student Ability in an Introductory Database Course

G. Joy Teague
Division of Computing and Mathematics
Deakin University
Geelong
Victoria, 3217
Australia

Recent decline in pass rates in Information Systems units, which was attributed to high school graduates not being well prepared for tertiary study, necessitated modification of teaching methods. A number of new and revised techniques were implemented concurrently with particular emphasis on increased use of advance organizers, on examples which would result in the generation of "mental models" of the material and on having the students rephrase the material in their own words.

INTRODUCTION

The Division of Computing and Mathematics at Deakin University offers majors in Information Systems, Computer Science, Mathematics and Statistics. In the years 1984 to 1986 there was a noticeable decline in the performance of students enrolled within the Division, evidenced by low pass rates in most first and second year units. During this period neither the staff teaching these units nor the content of the units changed substantially so it was concluded that the quality of the students had declined. An analysis of student backgrounds and academic performance (1) clearly indicates that student performance declined in the years 1984 to 1986. Table 1, taken from the report, shows the proportion of students passing more than half of the full-time student load of eight units in the years 1984 to 1986.

	Proportion of Students Passing		
	8 units	7 or 8 units	> 4 units
1984	.47	.55	.78
1985	.34	.51	.67
1986	.28	.29	.47

Table 1

In the 1986 end of year examinations the pass rate in almost all of the first and second year Computing and Mathematics units was around 50%. In previous years pass rates had been in the range 60% to 75%.

Suggested Reasons for Declining Standards

There has been some public discussion in newspapers about a perceived decline in secondary school standards, however no formal study of student standards at secondary schools has been undertaken. A number of reasons for the decline have been advanced.

(1) The secondary school curriculum has been broadened to include more "soft options", intended to encourage more students to complete secondary schooling
(2) there has been a corresponding reduction in the number of students selecting both mathematics subjects at year 12. Mathematics background in year 12 has been shown to be an accurate predictor of performance in the first year of the degree program (1).
(3) Students no longer learn how to study.
(4) many of the students entering university are there simply because they can't get a job.

How to Improve Pass Rates?

The question of how to improve pass rates without lowering standards became of paramount importance. Lowering standards to pass more students would eventually result in a reduced

demand for graduates from the program and consequently a further reduction in the quality of the students entering the program as the better students chose to go elsewhere. One solution, not accepted by all staff, was to "teach less and teach it better".

THE INFORMATION SYSTEMS STREAM AND THE INTRODUCTORY DATABASE UNIT

The Information Systems stream consists of a one semester unit of COBOL programming in first year, followed in second year by a File Structures unit and the introductory Database unit. (A unit is one quarter of a semester's workload.) In both of the second year units all assignments are written in COBOL. The intention is to use the assignments to progress the students as far through the chain of cognitive accomplishments (2, 3, 4) as possible. In particularly to contribute to the progress of students towards the third link in the chain, the development of "techniques for using generalized templates and procedural skills for one language when learning a new formal system" (4 (p. 55)) by making them very proficient in one language, currently COBOL. It is considered that the best method of achieving this is to have students write as many programs as possible. That is, "the best way to learn about algorithms, procedures, and programming . . . is to have hands-on experience" (2 (p. 33)).

The theoretical material in the introductory database course consists of four sections:

(1) an introduction to Codasyl databases, the basis for assignment work
(2) a comparison of the data and file structures of five commercial databases, viz. Adabas, Total, System 2000, IDMS and IMS
(3) a consideration of factors relating to database recovery, integrity, concurrency and security
(4) an introduction to relational database concepts.

Approximately thirteen class contact hours are spent on each of the four sections. The relational database section is not discussed in this paper.

Changes to Teaching Methods

Changes to teaching methods in the database unit were based substantially on Mayer's recommendations for increasing the novice's understanding of computers and computer programming (5). The first two, providing a concrete model and having the learner restate the new material in his/her own words were used extensively. The third technique, to "assess the learner's existing intuitions about computer operation and try to build on them, or modify them, as needed" (5 (p. 129)) was difficult to achieve with a class of seventy students. Frequent short answer tests were given to try to determine where material had not been understood and, where identified, this material was revised. Mayer's fourth and fifth recommendations, providing methods for grouping statements into "meaningful chunks" and decomposing statements into substatements, were not considered appropriate in this class.

Less Material Presented

The first step was to reduce the quantity of material presented thus extending the time available for reinforcement of those concepts which were presented. To this end, material which was considered to be peripheral to the concepts being taught, which was unlikely to be remembered by the students, and which was unsuitable for examination purposes was omitted from lectures. One hour of the three lecture hours each week was devoted to tutorial type presentation of examples and for class tests.

Programming Tutorials

The students in the Database class were Computer Science and/or Information Systems majors who had completed five programming courses. They should have spent approximately three hundred hours programming for these classes and achieved the lower end of what might be considered to be the "expert" range (6). While this is considerably less than the periods of up to 100 hours per week which many experts put in to gain their expertise (7), it is considerably more than most novices have spent programming. Despite this, a number of students had expressed their concern before the beginning of the semester that, when required to write a program for an assignment, they had no

idea of how to begin. To assist these students, a one hour tutorial was held each week.

Approximately twelve of the class of seventy students attended regularly throughout the semester, most students having "better" things to do. Of the 28% of the class who failed the programming test conducted towards the end of the semester, 60% had attended no tutorials and 27% had attended only one or two tutorials. The terms "movers" and "stoppers" coined by Perkins et. al. (8) to describe the different approaches taken by novice programmers when they encounter difficulties is typified by the attitudes of these students to tutorials.

Sample Programs

During the Codasyl section of the course some of the additional time made available by reducing the material presented was used to work through additional program examples. Normally in lectures, when a new statement was introduced, a brief example would be given. In the additional time longer examples were presented and explained, particularly examples which incorporated a number of Codasyl commands in their solutions.

Mental Images

Much of the additional time made available by reducing lecture material was spent in providing examples which could be used to build mental images. In the Codasyl section, as described above, this was done by providing examples and schema diagrams. The building of mental images was emphasized heavily in the commercial systems section by providing diagrams of file structures and demonstrating the addition of new records into the various databases. Limited time in the section on recovery, integrity and concurrency meant that the inclusion of material to create mental images was limited also.

Advance Organizers

In all three sections, advance organizers were used to introduce new material whenever possible. The Assimilation Encoding Theory proposed by Mayer predicts that appropriate use of an advance organizer will result in the transfer of "anchoring material" already in long term memory into working memory, where it will be integrated with the new material during learning, and the integrated knowledge will be transferred back to long term memory (9). Generally, the types of advance organizers used were related to material covered in the File Structures course in the previous semester, to assignments done in that course and to the preceding material in the Database course.

Tests

A number of short-answer tests were given during the semester. The time allowed for a test was approximately thirty minutes. There were a number of objectives in administering regular tests. Mayer advocates that the learner should be encouraged to "actively restate the new technical information in his or her own words" (5 (p. 129)). Answering test questions forced students to describe some of the more important concepts in their own words. The second objective was to encourage students to revise lecture material shortly after the time it was presented to assist in the process of transferring lecture material into long term memory. Finally, these tests provided pointers to the topics which had not been well understood and additional time was spent on these topics.

Summarizing Lecture Material

Overhead transparencies which contained a summary of the lecture material were used in all lectures. Most examples and various other points were written on a chalk board. It was common practice for students to write down anything which they saw written, but not to make notes independently. For some of the more important points no transparency was presented and the students were specifically directed to write their own notes, thus forcing them to summarize the material using their own words.

RESULTS

Table 2 shows Information Systems pass rates under the old teaching methods and the 1987 Database pass rate after the change in teaching methods. The Statistics pass rates have been included in the table for comparative purposes.

	Pass rates		
	End '86	Mid '87	End '87
Info Systems	49%	49%	71%
Statistics	47%	52%	52%

Table 2

Final Examination

The final examination appeared to the examiner to be more difficult than the corresponding examination at the end of the previous year, however feedback from students indicated that they found the examination fairly easy. Whereas the mid-year File Structures examination answers had highlighted fundamental misconceptions and lack of knowledge, the Database examination answers generally indicated understanding of basic concepts.

CONCLUSIONS

The pass rate in the introductory database class increased from less than fifty percent in 1986 to 71% in 1987. There are a number of factors, other than improvements to the methods of teaching which might have contributed to the increased pass rate. However, it seems probable that the revised teaching methods contributed to the better understanding of the subject material by the students in the class resulting in a much higher pass rate in the unit. In particular, the techniques used to build mental images and to encourage their transfer into long term memory appear to have resulted in the students being better prepared for their examination. The additional work done by students who regularly attended tutorials resulted in better grades in programming tests for almost all of those students.

REFERENCES

(1) Ridgeway, G., Unpublished report on first-year Computing/Mathematics student pass rates 1984 to 1986, Deakin University, Geelong, Victoria, 1987

(2) Pea, R. D. and Kurland, D. M., *On the Cognitive Prerequisites of Learning Computer Programming*, (Technical Report No. 18), Bank Street College of Education, New York, NY. Centre for Children and Technology, 1983

(3) Mandinach, E. B. and Linn, M. C., *The Cognitive Effects of Computer Learning Environments*, Journal of Educational Computing Research, Vol. 2(4), pp. 411-427, 1986

(4) Mandinach, E. B. and Linn, M. C., *Cognitive Consequences of Programming: Achievements of Experienced and Talented Programmers*, Journal of Educational Computing Research, Vol. 3(1), pp. 53-72, 1987

(5) Mayer, R. E., *Contributions of Cognitive Science and Related Research in Learning to the Design of Computer Literacy Curricula,* in Computer Literacy, R. Seidel, R. Anderson and B. Hunter (eds.) Academic Press, New York, pp. 129-159, 1982

(6) Soloway, E. and Ehrlich, K., *Empirical Studies of Programming Knowledge*, IEEE Transactions on Software Engineering, Vol. SE-10, No. 5, pp. 595-609, 1984

(7) Kurland, D. Midian, Mawby, Ronald, and Cahir, Nancy, *Development Studies of Computer Programming Skills*, presented at the annual meeting of the American Educational Research Association, New Orleans, LA, April, 1984 (Also Technical Report No. 29, Bank Street College of Education, Center for Children and Technology, New York, October 1984)

(8) Perkins, D. N., Hancock, C., Hobbs, R., Martin, F. and Simmins, R., *Conditions of Learning in Novice Programmers*, Journal of Educational Computing Research, Vol. 2(1), pp. 37-55, 1986

(9) Mayer, R. E., *Can Advance Organizers Influence Meaningful Learning?*, Review of Educational Research, Vol. 49, No. 2, pp. 371-383, 1979

DESIGNING AN ETHICS COURSE FOR THE CIS CURRICULUM

Paul J. Will
Department of Computer Science
SUNY College at Oswego
Oswego, New York

ABSTRACT

A new field of inquiry, ethics in computing, is emerging amid concerns about computer crime and surveillance. Some professional societies recommend inclusion of such a component in the curriculum but little is being done to implement this suggestion.

Research reveals a range of concerns in terms of topics to be covered and positions to be examined. Offering a course on computer ethics involves a variety of teaching strategies, including how directive the instructor should be in forming attitudes.

Integrating humanistic perspectives into a technically-oriented discipline presents a unique opportunity to develop lines of student inquiry atypical of the usual computer science instructional mode.

BACKGROUND

Many sources cite the need for ethics education in the training of computer professionals, yet little is being done to implement such a component in the computer science/information science curricula.

An attempt in that direction is being made at the State University of New York College at Oswego. The Department of Computer Science has approximately 240 undergraduate majors, and a special topics course, CS 490 - Ethics in Computing, was offered in the Spring 1988 semester for upperclassmen. Of the 22 students who elected the class, 18 were computer science majors, 3 were management science, and 1 was an applied mathematical economics degree student. Many of the CS students were in the Information Science option of the Computer Science degree program. This suggests that such a course would appeal to both CS and MIS majors. Two-thirds (14) of the 22 students had completed several business courses and about an equal number (13) had at least one philosophy course in their background. Only one student had a previous ethics course.

The course was promoted as being germane to those considering career options as MIS director, data processing manager, or data base administrator. The students expressed interest in expanding their knowledge of ethics in computing and, throughout the

course, were amazed at the scope of its impact. In some sense, this initial class was a prototype of a new course, InSci 300 - Current Problems in Information Science, offered in Fall 1988 as a regular part of a new Information Science degree program. Portions of the content covered in CS 490 are incorporated into InSci 300.

CONTENT

The overall approach is to present the extent and variety of ethical issues associated with computing before examining the specific responses of professional societies or legal bodies to the problems. Therefore, the first half of the course examines computer crime, in all its various guises; privacy concerns; social and economic dislocation; and defense contracting.

Computer crime is a good place to begin because of its intrinsic interest for students and the abundance of information relating to it. Various techniques for commiting computer crime from trojan horses, logic bombs, trap doors to viruses are discussed. The profile of typical computer criminals and their motives are noted. Financial crimes and fraud are looked at as well as hackers and software piracy. The differences between copyright, shareware and public domain software and policies related to these categories provide the focus for several class sessions. The types of warranties and legal liabilities that exist are outlined.

Computer surveillance and the invasion of individual privacy are the basis for a second unit of study. The present uses of data bases and the possible misuse of the electronic funds transfer system for investigatory purposes are detailed. Discussion of employee monitoring and matching records across information systems shows the potential for harassment. Displacement and other social and economic consequences of the Information Age are not neglected, although they form a smaller area of attention.

The sensitive area of defense contracting and its relationship to computer science along with questions about the propriety of the Strategic Defense Initiative led to a vigorous class discussion of ethical positions. A range of opinions on such issues are evaluated in the context of actual pronouncements by professionals on these questions.

Only after this wide-ranging examination of computer-related situations with ethical consequences is a serious look taken at various professional codes (DPMA, ACM, and IEEE) and legislation (Privacy Act of 1974, Computer Fraud and Abuse Act of 1986, Computer Security Act of 1987, and various state laws, including New York's).

The DPMA position statements related to these areas as well as its Model Computer Crime Act are useful adjunct information to these documents. These are available from Joseph Collins, Governmental Affairs Manager, at DPMA national headquarters. This coverage is important because a graduate of this course will be viewed as somewhat of an expert in the area and needs to be cognizant not only of present positions but the sources of continuing information on these matters.

The intent of this particular curricular design is initially to focus the student's attention on the extent and variety of computer-

related ethical problems and later to examine reasoned responses to these issues. At the beginning of the course no attempt is made to judge the student's personal ethical position, rather a very open-ended approach is taken on purpose. This stimulates discussion. Later efforts are made through readings and in-class presentations to refine and sharpen student response. Upon leaving the class, students are conversant with a range of responsible positions on the sensitive ethical issues that relate to our profession.

RESOURCES

Given the contemporary nature of the topic, ethics in computing is best analyzed through a variety of sources. An anthology of relevant articles from professional and popular publications provides a current focus to any such study. While the textbook Ethical Issues in the Use of Computers, itself such an anthology, is assigned, it is supplemented extensively by other readings. In addition, a number of audio-visual resources, as the Epicot film Ethics in the Computer Age, are employed to address particular topics. Guest speakers are utilized, such as the Director of the campus Instructional Computing Center, who discusses site license provisions and software use policies, and an ethics professor, who talks about the theoretical basis for ethical decision-making.

The students serve as resource persons in that they write several reviews of articles and then research a related topic of interest to them, write up the research results, and present them to the class as a whole.

Professional societies as DPMA and ACM and their respective publications are good sources of information as well as more specialized groups such as Computer Professionals for Social Responsibility. Copies of actual legislation related to these issues are readily available from political representatives on the state and national levels.

Teaching a computer ethics course provides a venue for the computer scientist to introduce humanistic perspectives and concerns into the traditionally technical curriculum. This is an exciting venture and offers a professional challenge. At the same time, it sensitizes future leaders in the computer field to a whole area of concern and responses they have not been exposed to before. Issues, I might add, that will have an ever increasing importance in society.

BIBLIOGRAPHY

(1) Bellin, David and Gary Chapman, eds., Computers in Battle--Will They Work?, Boston, Harcourt Brace Jovanovich, 1987.

(2) Forester, Tom, ed., The Information Technology Revolution, Cambridge, Massachusetts, The MIT Press, 1985.

(3) Forester, Tom, ed., The Microelectronics Revolution, Cambridge, Massachusetts, The MIT Press, 1983.

(4) Johnson, Deborah G., Computer Ethics, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1985.

(5) Johnson, Deborah G. and John W. Snapper, Ethical Issues in the Use of Computers, Belmont,

California, Wadsworth Publishing, 1985.

(6) Logsdon, Tom, Computers and Social Controversy, Rockville, Maryland, Computer Science Press, Inc., 1980.

(7) Parker, Donn B., Ethical Conflicts in Computer Science and Technology, Reston, Virginia, AFIPS Press, n.d.

(8) Parnas, David Lorge, "Why I Won't Work on SDI: One View of Professional Responsibility," a paper available from Computer Professionals for Social Responsibility, Inc., PO Box 717, Palo Alto, California.

(9) Rosenberg, Ronni, "Privacy in the Computer Age," a paper available from Computer Professionals for Social Responsibility, Inc., PO Box 717, Palo Alto, California.

(10) Sieber, Ulrich, The International Handbook on Computer Crime, New York, John Wiley and Sons, 1986.

PC BASED COBOL INSTRUCTION

Harry C. Benham, Ph.D.
University of Oklahoma

This paper takes the position that Personal Computers can provide an environment better suited to learning COBOL than the environments associated with the traditional mainframe or minicomputer based COBOL instruction. The advantages of PC based COBOL instruction over mainframe based instruction are the debugging facilities of PC COBOL, the editors available for PC COBOL, a less intimidating machine, the screen painting/code generation capabilities, and the similarity of PC COBOL programming to the use of state-of-the-art programmer workstations. Disadvantages of PC based COBOL instruction include the student being responsible for file maintenance and a lack of experience with COBOL in a mainframe environment.

INTRODUCTION

COBOL, the most widely known and used language, is very good at processing information in the real world. So in this age of Fourth Generation Languages, Object Oriented Languages, and Database Query languages, COBOL remains the primary business data processing language. Not only are the vast majority of existing applications written in COBOL but well over half of all new application development uses COBOL (4). Because COBOL is so central to business data processing, COBOL instruction is explicitly included in the model curriculum suggested by DPMA and ACM. Consequently COBOL instruction is a vital component of the Information Systems curriculum and the question of how to provide COBOL instruction arises.

This paper takes the position that Personal Computers can provide an environment better suited to learning COBOL than the environments associated with the traditional mainframe or minicomputer based COBOL instruction. Since the PC has established itself as a serious addition to business data processing hardware, it seems only natural that the most widely known and used language would also migrate to the PC. PC's have established themselves by enhancing the productivity of their users. With increasing numbers of Information Systems students and an expanding curriculum, enhanced productivity in COBOL programming instruction is essential. If tomorrow's student can master COBOL in less time, more time will be available for instruction in the newly expanded areas of 4th Generation Languages, Telecommunications, Decision Support Systems, and Database to name just a few.

The remainder of this paper supports COBOL instruction using PC's. The next section reviews the capabilities of COBOL Compilers available for the PC. It is followed by a section which presents the major advantages of PC based COBOL instruction. The third section lists the disadvantages.

Concluding remarks are contained in the final section.

COBOL ON THE PC

A number of COBOL compilers are available for the IBM PC family of microcomputers. Vendors offering one or more COBOL compilers for the PC include IBM, Micro Focus, Microsoft, Realia, Ryan McFarland, MBP, Nevada, and Waterloo. At the low end, a PC COBOL compiler can be obtained for well under $100. At the top end, prices are on the order of $3500 per copy. Varying with price are the "quality" of the compiler and features provided. One indication of compiler "quality" is the level of certification obtained from the General Services Administration's (GSA) Federal Compiler Testing Center. PC COBOL compilers are available which are not GSA certified, certified for ANSI '74 at the low-intermediate level, certified for ANSI '74 high, and ANSI '85 high. Additionally, PC compilers are available which support the language extension found in IBM OS/VS COBOL and IBM VS COBOL II. Top line compilers provide the option of selecting the language variant.

Features provided with the compiler include editors, debugging tools beyond the standard ANSI COBOL DEBUG option, Screen or Output Form painting, and code generation. Those compilers which include editors tend to provide very functional full screen editors. Some provide syntax checking. Block moves and copies are typically supported as are search and replace functions. Line numbering is either automatic and invisible or not required. At least one compiler accepts standard ASCII files. Thus any text editor capable of writing an ASCII file may be used to write Code.

Debugging tools included with PC compilers typically allow interactive source level debugging. For example, the Micro Focus Animator debugging tool allows one to view source statements as they are executed (at a variable speed), halt execution to query or set data values, resume execution at any point in the program, and toggle to displayed screen output.

The most expensive of the PC compilers provide screen and output form painting utilities coupled with code generation facilities. Rather than layout a form on paper, count spaces, and then proceed to laboriously code DATA DIVISION statements, these packages allow the programmer to place headers and data items on the screen (representing an output form) in their desired location and then automatically generate the DATA DIVISION statements required to print the form. Occasionally, minor editing of the generated code is desirable.

There are two areas of incompatibility between PC COBOL and mainframe COBOL that should be noted. First PC COBOL may not support the same computational data types as mainframe COBOL. For example, COMPUTATIONAL-3 packed fields, a popular format on IBM mainframes, is alien to the 8088 processors in the PC family. Some PC COBOL compilers do nominally support packed fields through conversion to other data types. Screen I/O is the other area of incompatibility. All certified PC compilers provide basic ACCEPT and DISPLAY statements. Most have also implemented some form of screen I/O language extension based on PC screen addressing modes. At least one vendor has screen-mapping translators in the wings which will allow code for CICS and IMS screen handling to be implemented on PCs.

One final dimension of COBOL on the PC is execution speed. Mirecki (6) provides benchmark comparisons between several PC COBOL compilers. In comparison to mainframe COBOL, identical applications were found to run in approximately the same elapsed time on an IBM AT and a normally loaded IBM 3081 (5). Obviously, a floppy disk based PC application with considerable I/O will be slow. But for instruction purposes, execution speed is not paramount. And

even a slow execution on a PC may be several minutes less than the turn-around times on an over-worked academic mainframe.

INSTRUCTIONAL ADVANTAGES

The advantages of PC based COBOL instruction over mainframe based instruction are the debugging facilities of PC COBOL, the editors available for PC COBOL, a less intimidating machine, the screen painting/code generation capabilities, and the similarity of PC COBOL programming to the use of state-of-the-art programmer workstations. Each advantage listed is discussed below in turn.

The source code level interactive debugging tools available with PC COBOL facilitate program walkthroughs and the detection of logical errors. Common problems such as a misplaced period in an IF statement can be located and diagnosed very quickly in an interactive source level environment. The student, rather than spending hours staring at a source listing and wondering why his program doesn't work, can "watch" the program get "stuck." Being able to "see" the difference a period makes seems to internalize the concept better than being "told" many times. Errors in the DATA DIVISION's PICTURE clauses can be detected by querying data values during execution. Missing or misused End-of-File flags and their effects can be detected by watching program flow and querying data values when the flow diverges from the student's intention. In short, the effects of common student logic errors are visible in a source level interactive debugging environment. By viewing the problem and receiving immediate feedback, the student locates logic errors and learns not to repeat them.

The Editor's provided with PC COBOL tend to be superior to editors available on mainframes. There is, of course, considerable individual variation in preferences concerning editors. Yet the editors supplied with PC COBOL are well adapted to writing COBOL code. Those familiar with either mainframe editors or PC word processors find adapting to PC COBOL editors relatively straightforward and intuitive. One important area in which PC COBOL editors are superior to mainframe editors is in response time. The PC's dedicated processor can provide easier access to the code.

It can be argued that the PC provides a less threatening or intimidating environment especially for the beginning student than a mainframe. With a PC, the student has complete and total control over the hardware. If all else fails, the power switch is within reach. No mystical "computer" in another room or across the campus is doing strange things to the student's program and no operator messages will be displayed. In the self contained environment of the PC, the student is in total control and can do no real damage.

COBOL is an acknowledgedly verbose language. Once the basics of coding a DATA DIVISION have been mastered, repeated DATA DIVISION coding becomes tedious. For the student who has mastered the techniques of coding DATA DIVISION entries, the screen and output form painting facilities coupled with code generation capabilites can substantially reduce the amount of tedious coding. Code generation is no substitute for knowledge of how to write the code. In fact knowledge is required to use code generation effectively as minor editing of the generated code is often advisable. A knowledgeable student can, with screen painting and code generation tools, produce all the code required for a sophisticated screen display in a few minutes.

Finally, PC COBOL provides a reasonable simulation of state-of-the-art programmer workstations. In training Information System specialists, it is advisable to instruct them in the tools and techniques employed in the industry.

In fact, there seems to be an emerging trend toward offloading COBOL application development and perhaps even maintenance to the PC (2,3).

INSTRUCTIONAL DISADVANTAGES

The major disadvantage of PC COBOL is that the student must assume the responsibility for file maintenance. In a mainframe environment, file backup services are automatically provided for the student. On the PC, files or entire disks have been damaged resulting in the loss of the student's work. On the positive side, this rarely occurs to the same student twice indicating that a valuable although unintended lesson is learned.

The other disadvantages of PC COBOL are that the student does not gain experience with COBOL in a mainframe environment. That is, the student is not immediately exposed to Job Control Language (JCL), the problems of interfacing their program with the Operating System, or using system CALLs. Nor is the student exposed to Core Dumps. As good as the interactive debuggers are, they are not particularly well suited for diagnosing technical problems associated with invoking other language subroutines or interfacing device drivers. For these types of technical problems, one needs the tools provided in a mainframe environment.

CONCLUSIONS

There are advantages and disadvantages to providing COBOL instruction on the PC. For the beginning student, it is easily arguable that the PC with its superior logic error detection facilities, better editors, and less intimidating hardware provides an environment better suited to learning COBOL. As the student progresses on the PC, screen painting and code generation partially automate the more tedious aspects of COBOL allowing the student to focus attention on the logic and design of their programs. The PC alone provides a platform for mastering standard COBOL and would ideally position a student to write PC based applications which are beginning to appear (1).

As an instructional tool, the PC is less well suited for the most advanced students. Students at this level need debugging tools typically found only in mainframe environments. Perhaps the best strategy for the most advanced student, a strategy which seems to parallel emerging industry practice, would be to do initial development on a PC and then migrate to a mainframe environment. Such a technique would tend to foster the partitioning of operating environment specific aspects of programs from standard COBOL and exploit the strengths of each environment.

REFERENCES

(1) Angus, Jeff. "Info Center Managers Increase PC COBOL Use: Compiler Enhances Productivity." InfoWorld, Vol. 9 No. 7 (Feb. 16, 1987), pg. 33.

(2) ComputerWorld, "COBOL tool out for PC's", ComputerWorld, (December 7, 1987).

(3) DeWolf, Mary. "COBOL in a PC Setting," PC Tech Journal, Vol. 5 No. 1 (January, 1988), pp. 130 - 141.

(4) Doke, E. Reed and Lewis A. Myers. "The 4GL: On its way to becoming an Industry Standard?" Data Management, (August 1987), pp.10-12.

(5) Jalics, Paul J. "COBOL on a PC: A New Perspective on a Language and Its Performance," Communications of the ACM, Vol. 30 No. 2, (Feb.1987), pp. 142 - 154.

(6) Mirecki, Ted. "COBOL Performs," PC Tech Journal, Vol. 3 Nos. 6-8 (June, July, August, 1985), pp. 58-75,111-131, 107-136.

SQL PUTS 4GL POWER INTO COBOL

Alden C. Lorents
Northern Arizona University (15066)
Flagstaff AZ, 86001

ABSTRACT:

SQL is rapidly becoming a standard as a data manipulation language (DML) to process data using relational tables. New applications and re-engineering of old applications are gravatating toward more use of relational databases. COBOL had been, and continues to be the primary lanaguage supporting the development of information systems. SQL provides COBOL with the DML to access DB2 databases, and some of the programming power of a fourth generation language. This paper reviews some of the basics of using SQL in COBOL, and illustrates some of its power.

INTRODUCTION

A recent article in one of the trade journals started out with a picture of two information gerus talking: "What will the programming language of the year 2000 look like?" the man asked, "I don't know " his friend said, "but it will be called COBOL". The point is very clear, COBOL is going to be around for a long time. The reasons are as follows:

1. It is estimated that the current inventory of COBOL code is around 80 billion lines, which is growing at a 10% annual rate.

2. Most of the CASE technology is migrating toward generators that produce COBOL code. Examples of this include Auther Anderson's INSTALL/1, Pansophics TELON, and Software International's UFO and INTLECT.

3. There is a large pool of programmers with COBOL as their primary language.

4. A lot of investment is going into producing software tools to support COBOL based systems.

Another point is quite clear. COBOL is going to continue to change. Factors influencing this change include:

1. Screen input/output including ISPF dialogs.
2. Application development supporting communications.
3. Application development for the PC environment.
4. Development supported by relational databases.
5. Report generation.

These changes will continue to put new demands on curriculum updates and resource plans. The movement into the DB2 environment is going to happen, and it will have some major impacts in a number of areas. This paper is an overview of using SQL in COBOL, and illustrates some of its power.

SQL AND THE DATA DIVISION

The data division is used as an area to define and store data as the data is processed and transfered from one medium to another. The entries in the data division are not much different for DB2, conceptually, than they are for other databases such as IMS or IDMS. The primary definitions that go into the data division for DB2 are as follows:

A. The SQL communications area (SQLCA).

Every database system has its communications area for passing information between the program and the database system. The most used component is the error or status codes that communicate to the program the success or non-success of specific operations on the database.

ILLUSTRATION 1.

```
*****EXEC SQL INCLUDE SQLCA  END-EXEC.
  01 SQLCA.
     05 SQLCAID      PIC X(8).
     05 SQLCABC      PIC S9(9) COMP-4.
     05 SQLCODE      PIC S9(9) COMP-4.
```

The precompiler comments the original source statement. Other variables are generated, but are not shown here due to space.

B. The DB2 record (row) descriptions:

All of the DB2 tables that will be used in the program need to be defined to the program just like any other file or database. This can be done manually or by using the DB2 data declaration generator (DCLGEN). This processor will generate all of the COBOL code from the definitions stored in the DB2 catalogs. The resulting COBOL definition will be stored in a library to be picked up by the DB2 precompiler.

The declaration generator produces the following which is included in the COBOL program by the pre-compiler with the source statement.

EXEC SQL INCLUDE ITEMDEC END-EXEC.

ILLUSTRATION 2.

```
************************************************************************
* DCLGEN TABLE(AL_ITEM)                                                *
*        LIBRARY('D00992.SMMDB2.COBOL(ITEMDEC)')                       *
*        QUOTE                                                         *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS         *
************************************************************************
     EXEC SQL DECLARE AL_ITEM TABLE
     ( AL_ITEM_ID                    INTEGER,
       AL_ITEM_DESC                  VARCHAR(50),
       AL_ITEM_DEPT                  CHAR(4),
       AL_ITEM_AMT                   DECIMAL(7, 2)
     ) END-EXEC.
************************************************************************
* COBOL DECLARATION FOR TABLE AL_ITEM                                  *
************************************************************************
 01  DCLAL-ITEM.
     10 AL-ITEM-ID           PIC S9(9) USAGE COMP.
     10 AL-ITEM-DESC.
        49 AL-ITEM-DESC-LEN  PIC S9(4) USAGE COMP.
        49 AL-ITEM-DESC-TEXT PIC X(50).
     10 AL-ITEM-DEPT         PIC X(4).
     10 AL-ITEM-AMT          PIC S99999V99 USAGE COMP-3.
************************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 4             *
************************************************************************
```

Some items worth noting on this declaration are as follows:

1. The names in DB2 used the underscore character. They were converted to the hyphen for COBOL.
2. The item-id field was an integer field in DB2 which was converted to a signed computational field.
3. The item-description field was a variable field in DB2, and it is converted to a group of two fields, one for the length and one to store the content. The picture clause for the content will be the maximum. The length is a half word integer.
4. All decimal fields are stored as signed packed decimal fields.

C. Cursor Definitions:

The cursor concept provides the COBOL programmer with the ability to set up a logical file made up of a subset of data from a set of DB2 tables. This may be a subset of rows and/or columns from one table, or may be the result of a join between two tables. The programmer has all the power of the SQL select command with joins and subselects in creating this logical file.

ILLUSTRATION 3.

This illustration sets up a logical file for all vendor prices on a specific part number in sequence by vendor number.

```
EXEC SQL
   DECLARE  VNDRCOST CURSOR FOR
   SELECT   VNDR_CD, APS_PART_NUMBER,
            VNDR_CURR_COST, VNDR_BID_COST
   FROM     NTAD_VENDOR_COST
   WHERE    APS_PART_NUMBER = :V-PARTNO
   ORDER BY VNDR_CD
END-EXEC.
```

SQL AND THE PROCEDURE DIVISION

The use of SQL in the procedure division covers a lot of areas depending on the application and the type of program. This section will cover some of the basic examples of maintaining and retrieving data using DB2 tables.

Typical data maintenance includes adding, changing(updating) and deleting data.

ILLUSTRATION 4.

Inserting data into a DB2 table.

```
*EXEC SQL
*INSERT INTO AL_TRAN
*       (AL_TRAN_NO, AL_MEM_ID, AL_ITEM_ID, AL_TRAN_DATE,
*        AL_TRAN_TYPE, AL_TRAN_DESC, AL_TRAN_QTY, AL_TRAN_AMT)
*VALUES (:SQL-TRAN-NO, :SQL-MEM-ID-T, :SQL-ITEM-ID-T,
*        :SQL-TRAN-DATE,:SQL-TRAN-TYPE,:SQL-TRAN-DESC,
*        :SQL-TRAN-QTY, :SQL-TRAN-AMT)
*END-EXEC.
 PERFORM SQL-INITIAL UNTIL SQL-INIT-DONE
 CALL "DSNHLI" USING SQL-PLIST9
 IF SQLCODE < 0 GO TO S20-SQL-ERROR ELSE
 MOVE 1 TO SQL-INIT-FLAG.
 DISPLAY "TRANS INSERT COMPLETE".
```

Notice the use of the colon in front of each COBOL (Host) variable. The values can be variables or literals and the variable list must correspond with the DB2 column names.

This example also shows the generated code by the precomiler. Notice the use of GO TO logic. All SQL errors less than zero are serious DB2 errors that exit to the program's main SQL error routine. Processing can resume with the use of a continue.

ILLUSTRATION 5.

Updating a specific row in a table.

```
 S150-UPDATE-ITEM.
*****EXEC SQL
*****    UPDATE AL_ITEM
*****    SET AL_ITEM_DESC = :SQL-ITEM-DESC,
*****        AL_ITEM_DEPT = :SQL-ITEM-DEPT,
*****        AL_ITEM_AMT  = :SQL-ITEM-AMT
*****    WHERE AL_ITEM_ID = :SQL-ITEM-ID
*****END-EXEC.
      PERFORM SQL-INITIAL UNTIL SQL-INIT-DONE
      CALL "DSNHLI" USING SQL-PLIST18
      IF SQLCODE < 0 GO TO S20-SQL-ERROR ELSE
      MOVE 1 TO SQL-INIT-FLAG.


 S140-GET-ITEM.
*****EXEC SQL
*****    SELECT AL_ITEM_ID, AL_ITEM_DESC, AL_ITEM_DEPT,
*****           AL_ITEM_AMT
*****    INTO   :SQL-ITEM-REC
*****    FROM   AL_ITEM
*****    WHERE  AL_ITEM_ID = :SQL-ITEM-ID
*****END-EXEC.
      PERFORM SQL-INITIAL UNTIL SQL-INIT-DONE
      CALL "DSNHLI" USING SQL-PLIST15
      IF SQLCODE < 0 GO TO S20-SQL-ERROR ELSE
      MOVE 1 TO SQL-INIT-FLAG.
      MOVE SQLCODE TO 300-SQLCODE.
```

GET-ITEM illustrates retrieving data from the table before it is updated. This illustration also shows that the data is going into a record area called SQL-ITEM-REC, as opposed to specifying each field. After the program changes the values that need to be changed, based on input from some screen, UPDATE-ITEM would change the values in the DB2 table.

ILLUSTRATION 6.

Updating a set of rows in the database.

```
*****EXEC SQL
*****    DECLARE A-TRAN   CURSOR FOR
*****    SELECT * FROM AL_TRAN
*****    FOR UPDATE OF AL_TRAN_AMT
*****END-EXEC.


 S150-UPDATE-TRAN-AMT.
*****EXEC SQL
*****    UPDATE AL_TRAN
*****    SET AL_TRAN_AMT  =
*****        :SQL-TRAN-QTY / 2000 * :SQL-MEM-SALARY
*****    WHERE CURRENT OF A-TRAN
*****END-EXEC.
      PERFORM SQL-INITIAL UNTIL SQL-INIT-DONE
      CALL "DSNHLI" USING SQL-PLIST19
      IF SQLCODE < 0 GO TO S20-SQL-ERROR ELSE
      MOVE 1 TO SQL-INIT-FLAG.
```

This illustration shows the use of cursor logic to process a logical file using lookups to another table. Like any file the cursor needs to be opened, fetched and closed. Note that the cursor has been opened for update. The * means to select all fields. Notice the use of an equation in the UPDATE-TRAN-AMT routine. Also, the update is being made to the current cursor position for that table.

ILLUSTRATION 7.

Deleting the one to many situation maintaining referential integrity.

```
S300-DELETE-VNDRCOST.
     EXEC SQL
       DELETE FROM NTAD.VENDOR_COST
       WHERE APS_PART_NUMBER = :D-PARTNO
     END-EXEC.
     MOVE SQLCODE TO W-SQLCODE.

S300-DELETE-PART.
     EXEC SQL
       DELETE FROM NTAD.PART_DESC
       WHERE APS_PART_NUMBER = :D-PARTNO
     END-EXEC.
     MOVE SQLCODE TO W-SQLCODE.

S900-COMMIT.
     EXEC SQL  COMMIT   END-EXEC.
     MOVE SQLCODE TO W-SQLCODE.
```

This illustration uses a parts table and a vendor cost table with prices from different vendors for each part. When the part is deleted, all prices in the vendor cost table must also be deleted. Notice that a delete is done on each table where the part number is equal to the desired part. The delete routines are the same. However, in the case of the cost table, multiple rows will be deleted if there are multiple rows for that part. Notice also, the use of the commit. The commit is done after both deletes have been done. If something were to happen during the delete process, the database will be rolled back to its original status, maintaining its integrity.

SOME EXAMPLES THAT SHOW THE POWER OF SQL IN COBOL

The power of SQL comes in its ability to perform large amounts of file processing without much programming. SQL is not really an end user language as the requirements become more complex.

ILLUSTRATION 8.

Processing using a Join

```
*EXEC SQL
*    DECLARE A-MEMTRAN CURSOR FOR
*    SELECT AL_MEM.AL_MEM_ID, AL_MEM_NAME, AL_MEM_DEPT,
*           AL_MEM_SALARY, AL_TRAN_DATE, AL_TRAN_DESC,
*           AL_TRAN_QTY, AL_TRAN_AMT
*    FROM AL_MEM, AL_TRAN
*    WHERE AL_MEM.AL_MEM_ID = AL_TRAN.AL_MEM_ID
*END-EXEC.
*EXEC SQL
*    FETCH A-MEMTRAN
*    INTO
*    :SQL-MEM-ID, :SQL-MEM-NAME, :SQL-MEM-DEPT,
*    :SQL-MEM-SALARY, :SQL-TRAN-DATE, :SQL-TRAN-DESC,
*    :SQL-TRAN-QTY, :SQL-TRAN-AMT
*END-EXEC.
```

This illustration processes data from two tables (AL_MEM, and AL_TRAN) joined on member ID. Each time the program calls on the fetch routine, another row is obtained from the view created by the join of the member table to each transaction in the tran table.

ILLUSTRATION 9.

Processing using a combination of a join, function, and a correlated subquery with "group by" option.

```
EXEC SQL
   DECLARE MINCOST CURSOR FOR
      SELECT DISTINCT P.APS_PART_NUMBER,
                      P.APS_PART_DESC,
                      P.STATUS_CD,
                      P.NUMBER_AVAILABLE - P.STOCK_LEVEL,
                      C.VNDR_CURR_COST,
                      C.VNDR_CD
      FROM  NTAD.PART_DESC P,
            NTAD.VENDOR_COST C
      WHERE P.TYPE_CD LIKE :W-TYPECD
            AND P.APS_PART_NUMBER = C.APS_PART_NUMBER
            AND C.VNDR_CURR_COST = ANY
                (SELECT  MIN(C.VNDR_CURR_COST)
                 FROM NTAD.VENDOR_COST C
                 WHERE P.APS_PART_NUMBER = C.APS_PART_NUMBER
                 GROUP BY C.APS_PART_NUMBER)
      ORDER BY P.APS_PART_NUMBER
END-EXEC.
```

This SQL command is used to select the minimum bidder for each part number, within a product type, from a group of vendors who have bid on that part. The part data is in one table and the cost data is in another table in a one to many relationship.

The subselect selects out the minimum bid for each part within a part group. This narrows the field down to one row per part from the vendor cost table. The join is done between the part table and the logical minimum cost table on part number for all parts with a type-code that has a search string like "CRT". This search string would have been loaded into W-TYPECD from an online panel. The final result is retrieved in order by part number.

SUMMARY

The use of SQL within COBOL will grow as more applications are developed around DB2 type databases. SQL will provide some productivity gains to the programmer, due to the power of the language. There are however, a number of areas to consider regarding the use of SQL in COBOL.

1. Standards relative to structuring SQL within COBOL.
2. Standard approaches to solving typical problems with SQL.
3. Programmer training.
4. Curriculum approaches.
5. Alternative development languages and tools.

GENERAL REFERENCES

1. C. J. Date, Guide to the SQL Standard, Addison Wesley, 1987.
2. C. J. Date, and Colin J. White, A Guide to DB2, Addison Wesley, 1988.
3. IBM, IBM DATABASE 2 Application Programming Guide, (SC26-4293-0) May, 1987.
4. G. Wiorkowski, and David Kull, B2 Design and Development Guide, Addison Wesley, 1988.

A DATABASE APPROACH TO PROGRAM DESIGN

John E. Boggess, CDP
Lloyd R. Weaver, CDP
Clayton R. Molinari

Department of Computer Technology
Purdue University
West Lafayette, IN 47907

ABSTRACT

This paper presents techniques for generating data oriented program designs from a database model and logical access maps with pseudocode for procedural languages or action diagrams for fourth generation languages (4 G/L's). As part of this processing, guidelines for validating related records and checking the status of database operations are included, both for retrieval and update (add, change, delete) processing. The similarities between control break processing and database "set" processing (e.g. looking at all orders for a customer), and how many of the same basic principles can be applied will be reviewed. Also included are guidelines for translating generalized processing requirements from pseudocode into third-generation languages (COBOL, PL/I) or from action diagrams into fourth-generation languages (IDEAL, ADS-OnLine).

INTRODUCTION

Program design techniques including modularization, structure diagrams such as hierarchy charts, flowcharts, structured programming, and the like have advanced greatly over the years. They now provide a proven, stable foundation on which we can build as we enter the era of database, fourth generation language and CASE-driven program development. Specifically, there are tools and techniques which can be tailored to these environments to attain a very high level of programmer productivity and produce structured code which is data oriented, highly stable and highly maintainable.

By combining logical access maps (to show the sequence of accesses to database records or tables), action diagrams (to structure the actual processing of these tables or records), and optionally pseudo-code (to show specific processes for each record type or table), we can *quickly* produce programs which are not only highly structured but are also highly data-oriented.

DATABASE APPROACH

Before proceeding, it is important to define a database approach to program design, then show how it focuses on data requirements. This is different from a traditional approach, where we tend to concentrate on functional processes and only address data as needed.

In very simple terms, a database approach involves carefully defining data in a way that is both understandable to users and efficient to update and access. An approach usually followed as part of this process is called *normalization*, which exclusively associates data values (called elements, fields, or attributes) with the persons, places and things (defined

as records or entities) which they best describe. In this way, information about a customer need only be defined in one place which means changes to customer information need only be updated once. Likewise, customer information can be used by whatever systems and programs require it. These concepts correspond to *minimizing data redundancy* and *maximizing data sharing*, two key goals of a database approach. One end result of this database design and definition effort will be a conceptual database such as shown in Figure 1. In this figure, all of the arrows represent one-to-many relationships. For example, one customer has many orders but each order belongs to only one customer. An ordered product may be backordered and therefore be represented by more than one shipment line.

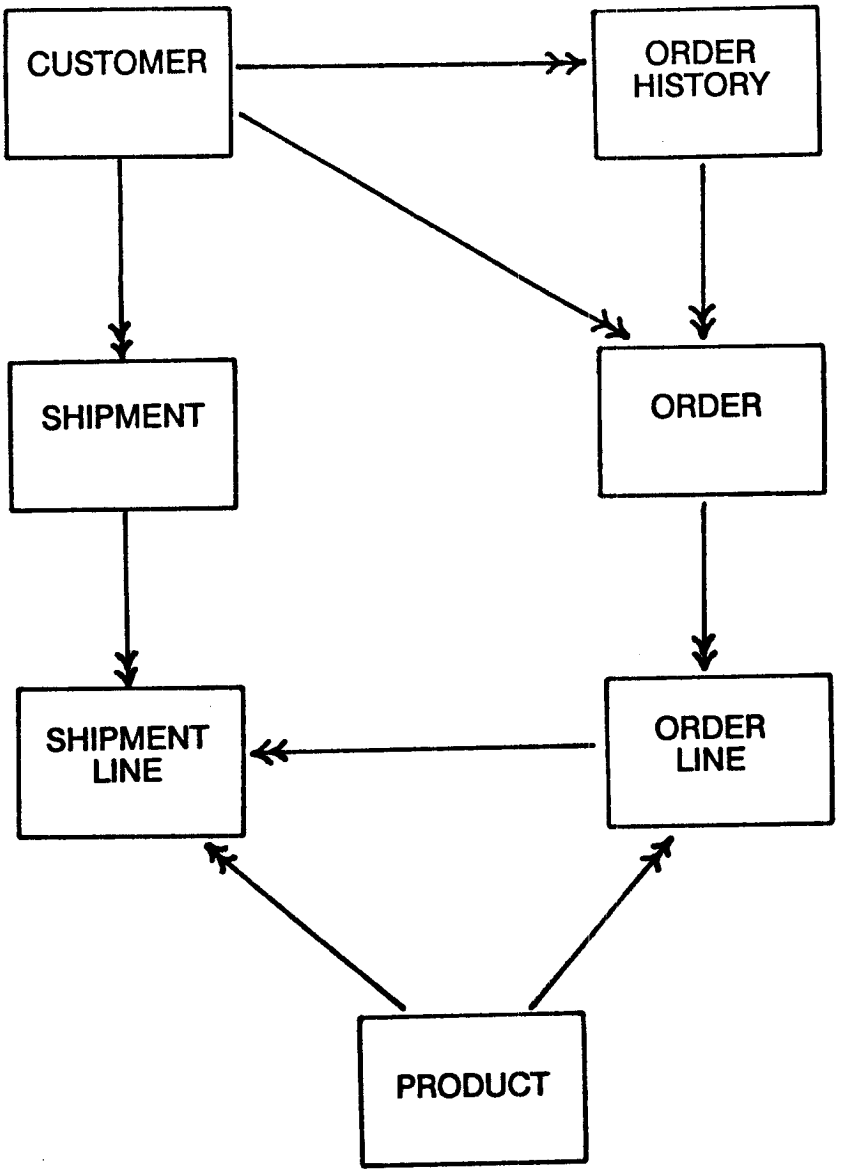


Figure 1. Conceptual Database for Order Processing

When applying a database approach to programming, we use the conceptual model of a database which already exists (or is being implemented in parallel with program development) to drive our program design. We use a *logical access map* (discussed below) to derive the structure of our program. By following such an approach, we can very quickly produce a structured program design showing which modules must be included, and in what hierarchical order.

Once we have our basic database program design, we can take advantage of many facilities provided by database management systems (DBMSs) which would otherwise have to be provided by the program. These include the ability to use record and field descriptions stored in data dictionaries, security features built into the DBMS, standard lookup tables and data validation modules, plus a number of related tools (such as query languages and fourth-generation languages) which can enhance programmer productivity considerably.

Although the use of database management systems is becoming increasingly popular in a number of organizations, it is important to emphasize that simply using a DBMS instead of VSAM or some other access method is not enough. At the end of the day, a database management system is no more than another (very sophisticated) access method, and will do nothing magical to help your program logic. To get the full productivity and maintainability gains anticipated, we must also follow proper database design (i.e. normalization) and structured database programming techniques.

FEATURES OF DATABASE-ORIENTED PROGRAM DESIGN

Traditionally, we focused on the functions or processes we wanted a program to perform, and data assumed a subordinate role. If data were available to match our processing requirements, we simply used it. If not, we had to input new data, sort existing data, and/or transform existing data in some way such as merging data from two or more files. Usually data was changed to match the program, rather than the program designed to match the data, even if this meant duplicating data.

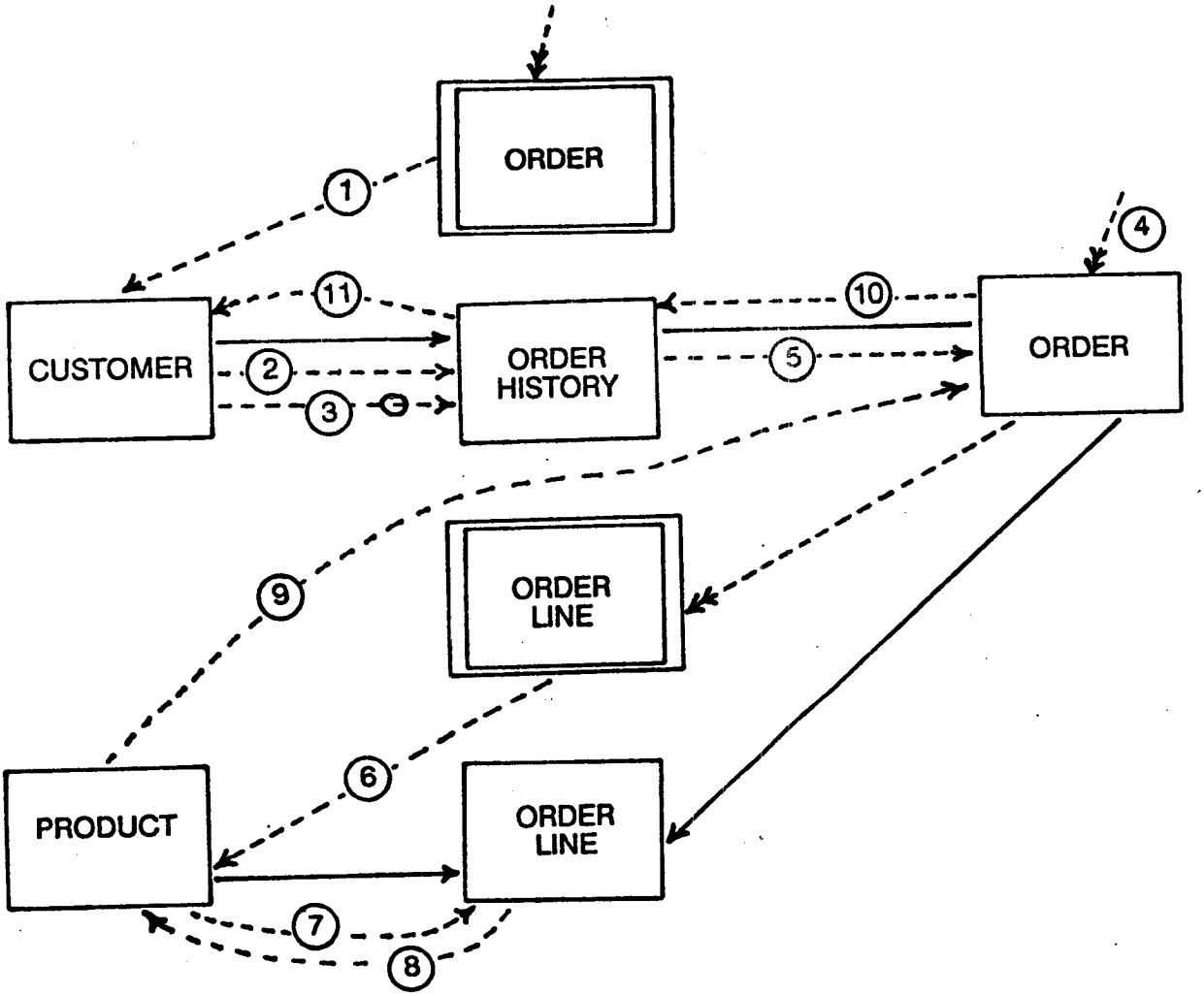
With a database approach to program design, the focus is always on data (i.e. the structure of the program should mirror the structure of the data). (2) If records and files have been properly defined with all information about a given entity (customer for example) stored in one place, (i.e. in one record or group of records). Changes to customer information are then easy to make; there are no cases where the information is changed in some places but not in others. Likewise, indexes can be included fairly easily with most database management systems so that all records can be accessed in different orders without having to first sort them and produce redundant copies of data.

Such an approach forces all the processing for a particular table or record type to be performed in a single module. This module can in turn invoke processing of related records in lower level modules. Processing can also be oriented toward either single occurrences of a record or sets of related records.

PHASES OF DATABASE ORIENTED PROGRAM DESIGN

Following a database approach, we want the focus of our program design efforts to be on *data*, and the functionality will follow naturally. In this section updating the database with customer order information will be used as an example. This involves the following four phases:

1. Extract all tables/record types and relationships needed by the program from the database diagram (which may be an entity-relationship (E-R) diagram, a network Bachman diagram, or some other tool). "Invert" this diagram so that the first record type accessed is at the top, and all related records are arranged hierarchically below. This inverted database is represented by the boxes and solid lines (relationships) in Figure 2.



Read INPUT TRANSACTION
Perform ORDER SUB-OBJECT processing for all transactions

ORDER-SUB-OBJECT
1. Validate CUSTOMER
2. Validate ORDER HISTORY
3. If NO ORDER HISTORY
   Create default ORDER HISTORY
4. Assign ORDER NUMBER
5. Store ORDER
   Perform ORDER LINE SUB-OBJECT processing until NEW ORDER
9. Update ORDER
10. Update ORDER HISTORY
11. Update CUSTOMER

ORDER LINE SUB-OBJECT
6. Validate PRODUCT
7. Store ORDER LINE
8. Update PRODUCT
   Read next INPUT TRANSACTION

Figure 2. Logical Access Map

2. Use dashed lines to show the flow of processing between different tables or record types. If multiple record occurrences for a particular record type are to be accessed (such as processing multiple orders), the dashed line should end with a double arrow head. If only a single (related) record occurrence is to be accessed (such as customer information for an order), show this with a single arrow head. Finally, if an access is conditional (such as adding an order history record only if one does not already exist), show this by placing an "O" for optional on the dashed line near the arrow head. The resulting diagram is a logical access map (LAM) (6), which shows the order of record access and the basic program structure required. We will then number each access and list specific functions which are performed. In our list we want to clearly identify what type of record processing is being performed (i.e. read, add, modify, delete).

   Note that different types of access for the same record or table type (e.g. reading an order record initially to assign the next order number, then subsequently adding an order record) are shown as two types of accesses on the logical access map. Note also that relationships do not have to exist between records shown on the LAM to have accesses between them; we show processing logically flowing from placing an order to shipment processing, even though there is no defined relationship directly between the order and the shipment record types.

3. When using procedural languages, develop pseudocode or Structured English for modules corresponding to each record access on the logical access map. When using non-procedural languages the pseudocode is more appropriately replaced by an action diagram. The most important thing to remember at this stage is that *we want all functions performed against a given record type to be initiated from or done in a single module.* For example "order processing" involves storing order header information, processing order lines, and shipping an order in lower level modules. As long as this module coordinates all phases of order processing and none of the subordinate modules is processing more than one record type, our modularization objectives are met. Figures 3 and 4 illustrate the pseudocode and action diagram which correspond to the logical access map in figure 2.

4. Translate the pseudocode to a programming language (COBOL, PL/I, etc.) or the action diagram to a fourth generation language.

```
Begin ADD ORDER PROCESSING module
        Input FIRST TRANSACTION record
        Repeat for ALL ORDERS
                Input CUSTOMER record
                Process CUSTOMER module
                If CUSTOMER RECORD FOUND
                        Input ORDER record using MIN ORDER NUMBER
                        Repeat until ORDER RECORD NOT FOUND or
                                        ORDER NUMBER > MAX ORDER NUMBER
                                Process CHECK ORDER module
                        End repeat
                        Store ORDER
                        Repeat for ALL ORDERED ITEMS
                                Process ORDER LINE module
                        End repeat
                        Find current ORDER record
                        Process UPDATE ORDER module
                        Find current ORDER HISTORY record
                        Process ORDER HISTORY UPDATE module
                        Find current CUSTOMER record
                        Process UPDATE CUSTOMER module
                Endif
        End repeat
End ADD ORDER PROCESSING module


Begin CUSTOMER module
        If CUSTOMER RECORD NOT FOUND
                Print "Invalid customer"
                Input NEXT TRANSACTION record
         Else
                Input ORDER HISTORY record
                Process ORDER HISTORY module
        Endif
End CUSTOMER module


Begin ORDER HISTORY module
        If ORDER HISTORY RECORD NOT FOUND
                Assign ORDER HISTORY default values
                Store ORDER HISTORY record
        Endif
End ORDER HISTORY module


Begin CHECK ORDER module
        Add 1 to ORDER NUMBER
        Input ORDER record using ORDER NUMBER
End CHECK ORDER module
```

Figure 3. Pseudocode

```
Begin ORDER LINE module
        Input PRODUCT record
        Process PRODUCT module
        If PRODUCT RECORD FOUND
                Store ORDER LINE
                Add 1 to WS OPEN LINES
                Add ORDER QTY * ORDER PRICE to WS ORDER VALUE
                Find current PRODUCT record
                Process UPDATE PRODUCT
        Endif
        Input NEXT TRANSACTION record
End ORDER LINE module


Begin PRODUCT module
        If PRODUCT RECORD NOT FOUND
                Print "Invalid product"
        Endif
End PRODUCT module


Begin UPDATE PRODUCT module
        Add ORDERED QUANTITY to QUANTITY DUE OUT
        Modify PRODUCT record
End UPDATE PRODUCT module


Begin UPDATE ORDER module
        Assign WS OPEN LINES to OPEN LINES
        Modify ORDER record
End UPDATE ORDER module


Begin UPDATE ORDER HISTORY module
        Add 1 to NUMBER ORDERS
        Add WS ORDER VALUE to TOTAL ORDERS VALUE
        Modify ORDER HISTORY record
End UPDATE ORDER HISTORY module


Begin UPDATE CUSTOMER module
        Add ORDER TOTAL to ACCOUNT BALANCE
        Modify CUSTOMER record
End UPDATE CUSTOMER module
```

Figure 3. Pseudocode (cont.)

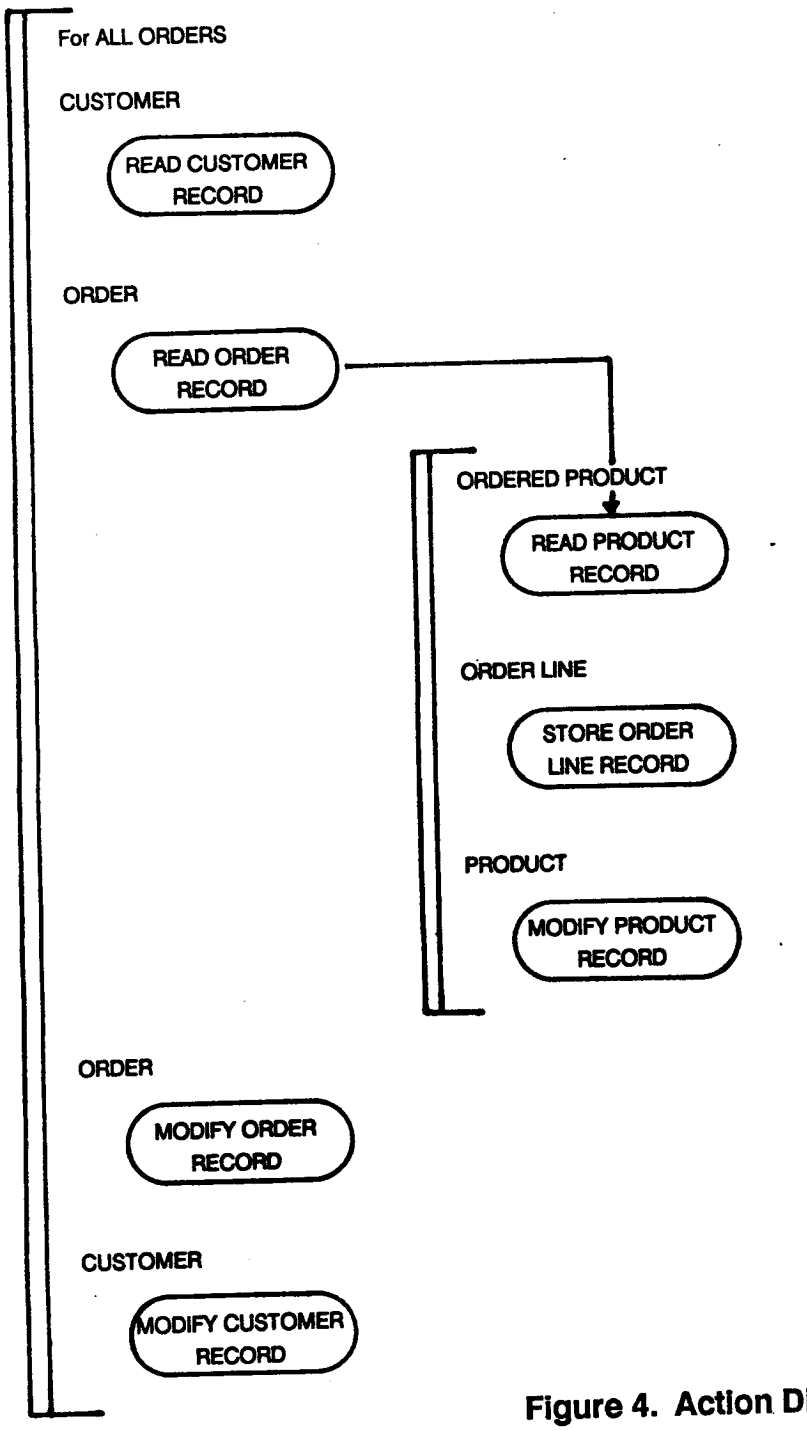


Figure 4. Action Diagram

effort. Knowledge concerning the organization requesting the project and what the project involved also varied widely among project leaders. Those having more knowledge about the project and the organization requesting the system were able to command more respect from their team members. Those project leaders who were unfamiliar with the project and organization found it difficult to manage the team effectively. Even though an inadequate knowledge base on the part of project leaders did lead to problems, this is not altogether uncharacteristic of industry. Therefore, it is debatable whether attempts to overcome this problem should be pursued or not.

Not surprisingly, one substantial problem was encountered in the graduate student method that had not been encountered in previous methods. This problem was the lack of coordination and communication among the project managers, their team members, the instructors in the systems development courses, and the instructor in the graduate ISM class. Project leaders and team members did not meet except at called meetings. Since the project manager's role was only part of one of a number of classes the graduate students were taking and since they did not attend the undergraduate classes, it was difficult to arrange meeting times. This also made it difficult for the graduate students to coordinate what the instructors in the development courses wanted from their students. A possible solution to this problem in the future would be to a conduct periodic team meeting in which the project manager and the course instructor attended. Another possibility would be to have a team liaison that would have weekly meetings with both the project leader and course instructor. These possible solutions should lead to greater coordination and communication between the project group members, the project manager, and the course instructor.

Conclusions

Considerable problems are faced by instructors in simulating the project manager's role in systems development classes. The methods most often used by instructors to recreate the project manager role are:

(1) Instructor selects a student to act as a project manager (instructor selection).
(2) Instructor allows the project manager to emerge naturally from the student group (student selection).
(3) Instructor plays the role of project manager in addition to the teaching role (instructor as project manager).

Many of the problems encountered with the traditional methods were overcome by using graduate students as project managers. The instructors of both the systems development courses and the graduate course were very satisfied with the approach. Using graduate students had many advantages including:

(1) Project managers were trained in project management.
(2) Project managers did not feel any peer pressure or resentment for being chosen to lead.
(3) Instructors time in the systems development courses was reduced.
(4) An exciting learning environment was provided for the project managers (graduate students).

One problem remained with this method -- some of the graduate students were ineffective managers. A new problem that emerged with this method was the lack of coordination and communication between the project managers and the team members.

EXPLORING THE 'ELECTRONIC TEXTBOOK': COMPUTERIZED LECTURE NOTES FOR STUDENT ACCESS

Bruce L. McManis, Ph. D.
L. Wayne Shell, Ph. D.
Nicholls State University

ABSTRACT

This research presents the concept of on-line lecture notes as an instructional aid for students. The concept is explored by relating the experiences of two instructors who pioneered their use within a College of Business Administration. The system of lecture notes, help files and other instructional aids is in place in a mini-computer time-sharing system operated by the College of Business Administration. Approximately two hundred students use these aids each semester. Advantages and disadvantages of the lecture note system are discussed. Differences between the two instructors' approaches to the concept are discussed. Possible changes resulting from widespread use of micro-computers are analyzed.

INTRODUCTION

"Today, many educational institutions experiment with diverse instructional methods using a variety of technological tools to provide the learner with flexible continuing educational services. . . .[M]uch of the effort in using technology in education has shifted from procuring hardware and software to helping personnel use and integrate equipment effectively in the curriculum.

"Also being addressed is the need to restructure educational goals in a growing information/service economy.

"Though the business world is rapidly acquiring and/or using networks to accommodate the millions of electronic devices in the total work environment, the educational arena is moving much slower, perhaps due to a lack of technical information and understanding of this new and growing field.

"Educators cannot ignore the increased use of different technical services in current use which constitutes an instructional system.

"An instructional system must provide for a cohesive, cost-effective and meaningful learning environment." (1)

At Nicholls State University, the teaching of management science and of operations management are heavily supplemented with computer aids. Naturally, the management science and operations management courses make substantial use of decision support packages--software for problem solving. These are by no means the only courses with computer-enhanced learning; AACSB has been quite pleased with the overall level of computer

involvement in our College of Business Administration. In addition to these expected computer uses, these courses use a very different type of computer aid. These computer supplements include lecture note summaries, outlines, homework problem and answer files, solved problem examples, course syllabi and glossaries. Their scope has recently been extended to the teaching of financial statement analysis. These supplements are extensive enough to be considered 'software textbooks.'

This presentation covers the origins of these computer aids, a detailing of some of the problems encountered and solved, and plans for modifying and improving upon the present systems of computer help in use. It relates the personal experiences of two faculty members who have initiated the use of the computerized lecture note system.

ORIGINS OF THE PROJECT

This project originated with one instructor's desire to build a set of lecture notes that would be easy to enlarge and update. That requirement was easily met by the use of a word processor or text editor. Each segment of the notes began as an outline, and grew to full-fledged prose and worked problems. While the original motive for creating the notes was personal, the instructor realized that students could also benefit from access to these notes, and began to make them available to his students. The innovation in this process is that the notes were not distributed in the standard form of mimeographed handouts, but in the form of electronic text files, which students could read from their terminal screens or dump to a printer for reading at a later date. The result is neither a set of handouts nor a student workbook, but a useful set of outlines, glossaries, notes and worked problems--an electronic text.

DESCRIPTION OF THE LECTURE NOTE SYSTEM

Making the notes available to students in this manner was simple, since the notes were composed with the text editor of a time-sharing system on which the instructor and the students had accounts. All that was required was to establish a 'library' in which the notes would reside, and to establish a security system that would allow the students to retrieve the notes, but not to alter them.

Until very recently, the time-sharing system was a DEC 11/60 with 28 megabytes of disk storage and about 30 terminals, four of which were printing terminals (LA36's). The system now consists of a DEC 11/70 with 282 megabytes of disk storage and about 50 user terminals. The terminals are located in one large student laboratory and in various faculty offices. The time-sharing system is the property of, and is for exclusive use by faculty and students of Nicholls' College of Business Administration.

The files are currently stored in a 'library account,' accessible by all students whose account numbers allow them access to that library. The structure of accounts is such that all of students of a given instructor have accounts which give them access to the same library, regardless of which course they are registered for. The files in the library are established with protection codes that allow students to read copies of those files, but cannot write onto them. This prevents students from accidentally or deliberately erasing or altering file contents. The instructor, as owner of the library account, is the only person authorized or able to write into those files; thus he can update these files as appropriate. This system of lecture notes and other computer aids has been in use for over four years with no known security problems.

The lecture note files were constructed using an old-fashioned text editor, EDT. It is not a modern word processor, but it possesses several valuable features not found in all modern word processors, such as an indexing feature and a mechanism for allowing for footnotes. EDT will not by itself produce the finished text file--it does not perform in the 'what you see is what you get' mode. Rather, the EDT file is embedded with 'dot commands' to control margins, spacing, pagination, etc. The EDT file is used as the source file to the RUNOFF utility, which translates the text and embedded commands into the finished text file. The instructor constructed the lecture note files in his faculty account on the time-sharing system, then used RUNOFF to send finished text files to the student library account. At the same time, the library versions of the files are given a read-only protection code. Students retrieve copies of files by use of the Peripheral Interchange Program (PIP) utility. As discussed in later sections, modifications have recently been made in both file creation and file retrieval techniques.

DR. SHELL'S SYSTEM

Current utilization of computerized help for the Operations Management course consists of 11 lecture note files totaling over 150 pages; a general HELP file, a course syllabus file, a series of homework answer/hint files, and a set of homework problem files which supplement the required textbook's problems. The lecture note files cover the following topics: Introduction to Operations Management, Linear Programming Concepts, Linear Programming Applications, Forecasting, Product and Process Design, Capacity and Location Decisions, Transportation Models, Layout Decisions, Inventory Management, Material Requirements Planning, and Quality Assurance. The individual lecture note files range in length from four to over twenty pages of single-spaced text.

In Management Science, there are six lecture note files, plus Help, Syllabus and Answer files. The lecture note files cover the topics of Introduction to Management Science, Decision Analysis, Forecasting, Linear Programming, Transportation Algorithm and Inventory Analysis.

In Business Research Methods, there are nine lecture note files, plus Help, Syllabus, and Research Topic files. The lecture note files cover the following areas: Introduction to Business Research, Use of Scientific Method and Modeling, Vocabulary of Research, Research Design, Measurement and Scaling, Instrument Design and Administration, Sampling Procedures, Statistical Methods, and Report Writing.

In all cases above, the Syllabus file is an electronic version of a printed handout furnished on the first day of classes. The Help file contains a listing and brief description of each of the files for student use, and is posted on bulletin boards in the computer labs.

These supplements have been successful enough to warrant their extension into other course areas, such as finance.

DR. MCMANIS' SYSTEM

Dr. McManis has converted a number of WordStar micro-computer files to the DEC time-sharing system. These fifteen files outline a course in Financial Statement Analysis, and were originally intended to become a hardbound, published textbook. Dr. McManis, concerned about student overuse of printers, wrote a BASIC language Menu program that simplified file retrieval procedures, and that restricted prin-

ting at the same time.

Dr. McManis became interested in the lecture note system as a result of the obvious success that Dr. Shell and his students had experienced. Dr. McManis has drafted most of a textbook in financial statement analysis, but has found it difficult to share his information with students. The need to share information was significant because his approach was very different from that used by any other author. In order to evaluate the validity of the new approach, Dr. McManis has used it in his classes since shortly after a preliminary outline was developed. Each semester the outline has been revised and additional text written in those areas that worked smoothly the prior semester.

His text materials were too large to cost effectively reproduce in hard copy (printed) form for the students, particularly in light of the frequency of updates to the material. As a first attempt to provide students with access to the information in a computer readable format, Dr. McManis made available floppy disks containing ASCII text files that could be displayed on any MS-DOS micro-computer. But rather than read text files on their monitors, most students quickly chose to print their diskette contents. This resulted in a less efficient reproduction method than if the material had been reproduced by conventional means.

This provided the impetus for Dr. McManis to explore a method of making the text available on the DEC 11/70 system where printing could be controlled. He also saw a second advantage of using the time-sharing system: statistics could be accumulated on student usage.

The original text was written using WordStar on a micro-computer. It was "printed" to a disk file to produce an ASCII file that was then uploaded to the DEC system and stored in a library account that is accessible only to students in the instructor's user group. A menu program was written to provide students with an easy method to access the text files. The menu program also provides control over student access. It does not permit printing of the files or access to the files from a printing terminal. It also provides pagination of the files while they are being displayed.

An advantage of using the time-sharing approach with restricted printing is the ability to control copy that is preliminary and therefore subject to error and/or lack of documentation. It also protects the author's ability to publish the work at a later date.

ADVANTAGES OF THE CURRENT SYSTEM

The advantages of this computer aids system can be divided into those benefits which derive from the content of the materials and their generic computerization, and those which arise from use in a time-sharing system. In a time-sharing environment, each file on disk can support many users simultaneously. Students can use the system at any time (even after hours with a modem on a home computer). This may prove the greatest advantage in relieving the instructor of some student visits to the office for "what did I miss in class today."

The instructor's lecture notes do not need to be complete at time of initial issue: portions of notes that are incomplete can be shown in outline form, to be augmented in later versions or in class. To accommodate updates and revisions, the notes should show date and version number (such as "Latest version number is x.xx, date mm/dd/yy).

One of the finest advantages of the computerized lecture notes is the ability to incorporate computer-generated output into text files. Students frequently use such programs as LINPRO, TRNPRT, DECMAK, and STAT. The first three of these packages are unpublished programs written by and/or modified by the faculty and staff at Nicholls State; these programs cover linear programming, the transportation algorithm and decision tables/decision analysis. STAT is the DEC-supported statistical analysis package. As discussed in a later section, students also make use of a wide variety of micro-computer software, including TWIN (a LOTUS-123 clone) and MICROSOM (a McGraw-Hill decision support package). Inclusion of micro-computer sessions into the text files is covered in that section. The note files can include verbatim terminal sessions illustrating their use. In the time-share system, this is done with the help of the DEC 'pseudo-keyboard' utility, ATPK, which captures entire terminal sessions--user inputs, computer prompts and outputs--into text files. These text files can later be edited to remove typographical errors and unwanted, repetitive or extraneous material. The computer sessions can be further edited to include explanatory text as desired. The results can be stored as text files, printed as lecture notes or converted to transparencies for class use.

Currently, there is substantial use of transparencies built from the computer-solved problems contained in the lecture notes. An alternative to transparencies is live, large-screen computer demonstrations. The College of Business has this technology and uses it frequently, but this is the subject of another paper. The use of transparencies is 'static,' in that it does not permit interactive changes in the problem being demonstrated. Use of live computer demonstrations allows for such changes and is therefore more 'dynamic.'

It is desirable to break lecture notes into manageable sections of perhaps fifteen pages or less. Such files should be short enough to encourage reading at the terminal, not printing them for later reading. Another advantage of the shorter file length lies in the ease of updating. The instructor has shorter files to work with, reducing the amount of computer time devoted to recalling and saving files. The student has only to check those files which have changed, and not bother with those files which have not been recently updated.

Students obtain several benefits from the computer supplements that have been described. Most instructors deviate from the material contained in the required textbook. If a specific deviation is important, the faculty member may distribute an appropriate handout to supplement the lecture. On the other hand, if the deviation is minor or consists of an alternative way of explaining text material, the instructor may likely leave such changes to the students' ability to take their own lecture notes of good quality. When this approach is taken two problems are created. First, not all students are good note takers. Any errors they make in taking notes become a part of their study materials. Second, if a student misses class it becomes difficult for him to obtain accurate versions of the material missed. The student must then rely on the notes of other students (which may suffer the same inaccuracies as his own, had he been present) or must make a special visit to discuss the matter with the instructor. The computer-based set of lecture notes corrects both of these problems. Students have a convenient method of checking the accuracy of their own notes; they have an additional source of information for notes on missed classes.

DISADVANTAGES OF THE CURRENT SYSTEM

One disadvantage of the time-share approach is the inability to print a partial file in time-share system; this disadvantage does not exist in a micro-computer environment because of the ability to print a partial file in micro based system through the Print-screen function or the LIST6 (or similar) utility. This disadvantage may disappear with the use of a menu program to control the amount of information available at each student's call.

Some aspects of lecture cannot be easily contained in the computerized notes; graphs are one example, complex equations are another (since EDT does not support subscripts and fancy graphics). Where this occurs, simply insert into the notes "This material will be covered only in the classroom." Perhaps a brief summary or outline can be used to hint at the missing contents. This occurs commonly where special graphics and symbols cannot be captured by the text file for inclusion in the lecture notes.

Currently, students in Dr. Shell's classes can either view note files on their terminal screen, or send note files to a printer. This uses a lot of paper, and is not a permanently desirable way to use the system. This is not a very efficient way to distribute printed materials. It is also costly, using large quantities of paper and ribbons; there are currently no lab fees for computer use, and the budget has been quite limited. Again, a menu program can insulate the system against widespread printing of files. So far, the DEC operating system cannot prohibit specific uses of the PIP command, so that knowledgeable students--hackers--can still print files by appropriate use of PIP, bypassing the Menu program. To do so requires only that they know the names of the lecture note files; this list is available through the general CAT (Catalog) or DI (Directory) commands. The amount of unwanted printing is therefore currently controlled by not actively disseminating knowledge about the use of the PIP utility or names and locations of the specific note files.

Under both lecture note systems that have been described, the student can view the note files from top to bottom only. The current design does not permit 'backing-up' to previous pages or sections of a file.

MODIFICATIONS AND IMPROVEMENTS

There have been several developments during the four years of field-testing these materials. These developments have led to changes in the manner in which the computerized course aids have been implemented. First, the university has acquired significant numbers of micro-computers, operating in a largely separate environment from the time-sharing system. Student problem-solving software is now largely micro-computer based, using McGraw-Hill's MICROSOM in management science, and using MICROSOM and TWIN in operations management. Some time-sharing software remains in use.

Two issues arose with the arrival of MS-DOS micro-computers in sufficient quantity for general student use. First, could (or should) the notes be transferred to the micro-computer environment, and how would the management of the micro based system differ from management of the time-sharing system? What are the advantages and disadvantages of each environment? For example, updates on a micro-computer based system might be more cumbersome than in a time-sharing environment. Also, there is practically no upper limit on file contents in the time-sharing environment, but there might be such restrictions in a micro-computer environment. Alternatively, the notes can be

contained in a system much like the sequence of help screens of Lotus or Twin. This may not be practical, as a spreadsheet screen may hold too little information per screen. Such a note system might eliminate the current disadvantage not being able to back up to review a previous section of a file. Second, could micro-computer outputs be uploaded to the DEC time-sharing system for inclusion into the electronic notes?

The ease of updating files might be mimicked by a network of micros linked to a file server. One update from the master account (library account) serves all users--no need for each student to write a fresh copy to his or her disk.

The ability to capture terminal or computer sessions is also possible in a micro-computer environment, but to a lesser degree. It is possible to use the printscreen function to send screen images to file, rather than to printer. This is obviously not quite the same as sending entire terminal sessions to file. In a micro-computer environment for the lecture aids, graphics characters could normally be included in the files; but in a system where micro-computer outputs are uploaded to a time-sharing system, capturing graphics might be impossible.

Discussions have been held on potentially allowing students to print certain files or to print portions of files using a maximum page count per session approach. These ideas have not been implemented at this point.

CONCLUSIONS AND RECOMMENDATIONS

The systems of lecture note files and other computer-enhanced learning aids work quite well, although they continue to evolve. The experiences related here suggest that such systems could easily be installed in a wide variety of hardware/software combinations, and would make only modest demands on the technical staff of the computing center. Students do in fact learn more and faster with these systems in place. Construction of lecture note files and other educational aids should encourage authorship by faculty.

REFERENCES

1. Charp, Sylvia. Editorial, T.H.E. Journal. September 1987, p. 10

2. Duckworth, Edwin A., James Kelley and Stephen Wilson. "AI Goes to School." Academic Computing, November 1987, pp. 6-10, 38-43, 62-63.

3. Reisman, Sorel. "Selecting Educational Software for Use with Textbooks." T.H.E. Journal. January 1987, pp. 80-84.

4. Smith, David. "How the Microcomputer Boom Changed British Schools." T.H.E. Journal, Dec-Jan 1987/88, pp. 81-84.

5. Smith, James H. "Coursework Development in a Framework II Environment." Academic Computing, November 1987, pp. 22-25, 43-45.

6. van Weert, Tom J. "Information Technology in Education: The Situation in the Netherlands." T.H.E. Journal, Dec-Jan 1987/88, pp. 77-80.

IMPLEMENTATION EXAMPLES

To illustrate the required phases of the database oriented approach in program design, we will show three examples. Figures 5 and 6 show partial implementation of the pseudocode using COBOL with IDMS and COBOL with SQL respectively. Figure 7 illustrates partial implementation of the action diagram using IDEAL.

```
MAIN.
        Read INPUT FILE at end move "y" to WS-END-OF-FILE-FLAG.
        Perform ORDER until END-OF-FILE.
        STOP RUN.

ORDER.
        Obtain calc CUSTOMER using INPUT-CUST-NUMBER.
        Perform CUSTOMER.
        If CUSTOMER-FOUND
                Obtain ORDER within ORDER-NUMBER-SET using MIN-ORDER-NUMBER
                Perform CHECK-ORDER until DB-REC-NOT-FOUND
                Store ORDER
                Move zeros to ORDER-TOTAL
                Move zeros to WS-OPEN-LINES
                Perform ORDER LINE until CONTROL-BREAK OR END-OF-FILE
                Obtain ORDER within ORDERED-PRODUCT-SET
                Perform UPDATE-ORDER
                Obtain CUSTOMER within CUSTOMER-ORDER-SET
                Perform UPDATE-CUSTOMER

CUSTOMER.
        IF DB-REC-NOT-FOUND
                Generate INVALID-CUST-DETAIL
                Read INPUT-FILE at end move "y" to WS-END-OF-FILE-FLAG

CHECK-ORDER.
        Add 1 to ORDER-NUMBER.
        Obtain calc ORDER using ORDER-NUMBER.

ORDER-LINE.
        Obtain calc PRODUCT using INPUT-PROD-NUMBER.
        Perform PRODUCT.
        If PRODUCT-FOUND
                Compute EXTENDED-PRICE = (PRICE*ORDERED-QUANTITY)
                Add EXTENDED-PRICE to ORDER-TOTAL
                Add 1 to WS-OPEN-LINES
                Store ORDER-LINE
                Obtain PRODUCT within PRODUCT-ORDER-SET
                Perform UPDATE-PRODUCT

        Read INPUT-FILE at end move "y" to WS-END-OF-FILE-FLAG.
```

Figure 5. Partial COBOL/IDMS Program from Pseudocode

```
MAIN.
    Read INPUT FILE at end move "y" to WS-END-OF-FILE-FLAG.
    Perform ORDER until END-OF-FILE.
    STOP RUN.

ORDER.
    Obtain calc CUSTOMER using INPUT-CUST-NUMBER.
    Perform CUSTOMER.
    If CUSTOMER-FOUND
        Obtain ORDER within ORDER-NUMBER-SET using MIN-ORDER-NUMBER
        Perform CHECK-ORDER until DB-REC-NOT-FOUND
        Store ORDER
        Move zeros to ORDER-TOTAL
        Move zeros to WS-OPEN-LINES
        Perform ORDER LINE until CONTROL-BREAK OR END-OF-FILE
        Obtain ORDER within ORDERED-PRODUCT-SET
        Perform UPDATE-ORDER
        Obtain CUSTOMER within CUSTOMER-ORDER-SET
        Perform UPDATE-CUSTOMER

CUSTOMER.
    IF DB-REC-NOT-FOUND
        Generate INVALID-CUST-DETAIL
        Read INPUT-FILE at end move "y" to WS-END-OF-FILE-FLAG

CHECK-ORDER.
    Add 1 to ORDER-NUMBER.
    Obtain calc ORDER using ORDER-NUMBER.

ORDER-LINE.
    Obtain calc PRODUCT using INPUT-PROD-NUMBER.
    Perform PRODUCT.
    If PRODUCT-FOUND
        Compute EXTENDED-PRICE = (PRICE*ORDERED-QUANTITY)
        Add EXTENDED-PRICE to ORDER-TOTAL
        Add 1 to WS-OPEN-LINES
        Store ORDER-LINE
        Obtain PRODUCT within PRODUCT-ORDER-SET
        Perform UPDATE-PRODUCT

    Read INPUT-FILE at end move "y" to WS-END-OF-FILE-FLAG.
```

Figure 5. Partial COBOL/IDMS Program from Pseudocode

```
PRODUCT.
    If DB-REC-NOT-FOUND
        Generate INVALID-PROD-DETAIL

UPDATE-PRODUCT.
    Add ORDERED-QUANTITY to QUANTITY-DUE-OUT.
    Modify PRODUCT.

UPDATE-ORDER.
    Move WS-OPEN-LINES to OPEN-LINES.
    Modify ORDER.

UPDATE-CUSTOMER.
    Add ORDER-TOTAL to ACCOUNT-BALANCE.
    Modify CUSTOMER.
```

Figure 5. Partial COBOL/IDMS Program from Pseudocode (cont.)

```
MAIN.
    Read INPUT FILE at end move "y" to WS-END-OF-FILE-FLAG.
    Perform ORDER until END-OF-FILE.
    STOP RUN.

ORDER.
    Exec SQL select CUSTOMER where :CUSTOMER# = INPUT-CUSTOMER-NUMBER;
    Perform CUSTOMER.
    If CUSTOMER-FOUND
        Exec SQL select ORDER where :ORDER# = MIN-ORDER-NUMBER;
        Perform CHECK-ORDER until DB-REC-NOT-FOUND
        Exec SQL insert
            into ORDER (ORDER#)
            values (:ORDER-NUMBER);
        Move zeros to ORDER-TOTAL
        Move zeros to WS-OPEN-LINES
        Perform ORDER LINE until CONTROL-BREAK OR END-OF-FILE
        Exec SQL select ORDER where :ORDER# = ORDER-NUMBER;
        Perform UPDATE-ORDER
        Exec SQL select CUSTOMER where :CUSTOMER# = CUSTOMER-NUMBER;
        Perform UPDATE-CUSTOMER
```

Figure 6. Partial COBOL/SQL Program from Pseudocode

```
CUSTOMER.
    If SQLCODE = 100
        Generate INVALID-CUST-DETAIL
        Read INPUT-FILE at end move "y" to WS-END-OF-FILE-FLAG

CHECK-ORDER.
    Add 1 to ORDER-NUMBER.
    Exec SQL select ORDER where :ORDER# = ORDER-NUMBER;

ORDER-LINE.
    Exec SQL select PRODUCT where :PRODUCT# = INPUT-PRODUCT-NUMBER;
    Perform PRODUCT.
    If PRODUCT-FOUND
        Compute EXTENDED-PRICE = (PRICE*ORDERED-QUANTITY)
        Add EXTENDED-PRICE to ORDER-TOTAL
        Add 1 to WS-OPEN-LINES
        Exec SQL insert
            into ORDER-LINE (ORDERED-PRICE)
            values (:PRICE);
        Exec SQL select PRODUCT where :PRODUCT# = INPUT-PRODUCT-NUMBER;
        Perform UPDATE-PRODUCT

    Read INPUT-FILE at end move "y" to WS-END-OF-FILE-FLAG.

PRODUCT.
    If SQLCODE = 100
        Generate INVALID-PROD-DETAIL

UPDATE-PRODUCT.
    Exec SQL update PRODUCT set :QUANTITY-DUE-OUT = ORDERED-QUANTITY;

UPDATE-ORDER.
    Exec SQL update ORDER set :OPEN-LINES = WS-OPEN-LINES;

UPDATE-CUSTOMER.
    Exec SQL update CUSTOMER set :ACCOUNT-BALANCE = ORDER-TOTAL;
```

Figure 6. Partial COBOL/SQL Program from Pseudocode (cont)

```
ADD-ORDER Procedure

    Transmit ORDER-TRANSACTION-ENTRY-SCREEN

    For each CUSTOMER-DATAVIEW where CUST-NO = SCREEN-CUST-NO
        Add 1 to SCREEN-NUMBER-ORDERS
        Set NUMBER-ORDERS to SCREEN-NUMBER-ORDERS
    EndFor

    For new ORDER-DATAVIEW
        If VALID-ADD
            Set ORDER-DATAVIEW by name
        Else
            Quit ADD-ORDER
        EndIf
    EndFor

    For each ORDER-LINE-DATAVIEW
        Do PROCESS-ORDER-LINE
    EndFor

EndProc

PROCESS-ORDER-LINE Procedure

    For new ORDER-LINE-DATAVIEW
        where ORDER-NUMBER = SCREEN-ORDER-NUMBER
            Set ORDER-LINE-DATAVIEW by name
            Add 1 to SCREEN-OPEN-ORDER-LINES
            Set OPEN-ORDER-LINES = SCREEN-OPEN-ORDER-LINES
    EndFor

EndProc
```

Figure 7. Partial IDEAL Program from Action Diagram

When implementing pseudocode with a procedural programming language there are some specific guidelines which are recommended. Within these guidelines, specific conventions vary depending on whether processing is for retrieving, adding, updating or deleting, or simply positioning to records. Each case is discussed separately below:

a. Retrieving records. When performing database retrieval processing, we want to issue a command to obtain a record or table occurrence, check the database status to make sure no unexpected error codes were encountered, *then* process the record itself and any related records. Although we are now accessing a database, we will still be doing control break processing where related sets of record values determine our control break levels. For example, for each customer in the customer set, all orders for that customer must be processed before accessing the next customer. In addition, all order lines from the order line set that match the given order would be accessed before processing the next order. This is clearly control break processing with the major break being customer, intermediate break on order and minor break on order line. In

COBOL we implement this as follows:

(1) Get the first (or only) desired record occurrence for a table or record type. In figure 5 this was done with an "OBTAIN CALC ..." for customer, and an "OBTAIN FIRST (or LAST) record WITHIN set" command. In the SQL example, an "EXEC SQL SELECT ..." command was used to access a single record occurrence An SQL *cursor* command would permit access to records from a set one at a time.

(2) Use an "IF" statement to check for any expected error conditions (i.e. a record not found, or no records in an expected set of records). Error conditions are handled within the "THEN" statements. If there are no errors, PERFORM a lower-level module (until end of set) to process that record type as part of the "ELSE" clause. In the examples, this was used to terminate processing if no product was found and to create a new order history if none was found.

(3) IMPORTANT: In the module PERFORMed to process that record type, *check the database status before doing any other processing!* (In IDMS, you do so with a "PERFORM IDMS-STATUS" statement; in SQL, you check the value of SQLCODE.) You can then do any processing related to this record type (which may include getting record occurrences for lower level record types on the logical access map and PERFORMing modules to to process those records).

(4) When processing a set of related record occurrences, get the next related record of that type as the last statement of the module. In IDMS, this would be an "OBTAIN NEXT/PRIOR record WITHIN set" command. For SQL, it would be an "EXEC SQL FETCH X INTO ..."

command.

b. Adding records. For modules which add new record occurrences, the approach is much different. We are again doing control break processing, but this time based on a transaction file. To better understand what is involved, it may be helpful to talk in terms of an *order object* (3), which is a meaningful entity to a user. Object entities consist of several database entities (record types / tables). *Objects* can then be broken into *sub-objects*, which also are meaningful units. For an order object we will define a *customer sub-object* (consisting of customer and order history information and stored as CUSTOMER and ORDER-HISTORY records), an *order header sub-object* (using the ORDER record) and an *order line sub-object* made up of ORDER-LINE and PRODUCT records. Control break processing is then done at three levels for the above example, where each level may require processing of several database tables or record types corresponding to the sub-objects defined. In COBOL, we will implement this processing as follows:

(1) At each transaction control break, follow a "top down" approach to accessing and validating sub-object record/table occurrences. For order level processing, we first access the customer record, then the order history record associated with the order. We validate each record to make sure corresponding occurrences are in the database before continuing.

(2) Once all higher level occurrences have been validated, we assign the key value (i.e. the next order number, within a range of permitted values, for a new order) and store the record.

(3) We then perform the next lower level of processing until a control break occurs for the current level (i.e. order processing until there is a

change of customer or order number, order line processing until there is a change of order number, etc.). We get the next input transaction record at the lowest level of processing.

(4) When a control break occurs, we "close out" processing of the current level. For example, before going on to the next new order to be processed the current order must be closed out. This involves updating that order, order history and customer to reflect any changes made as a result of lower-level processing, plus re-initializing any default values required for the new order occurrence to be added.

c. Changing or deleting records. Changing or deleting record occurrences will be done "bottom up". If we delete an order line, we want to update order information to reflect one fewer order lines. This may in turn result in an order which has no order lines (which we may also want to delete), etc. Processing updates upward within an object properly accounts for the cascading effect of such updates. This is in contrast to adding records where we work "top down" so that we can validate higher level records and thus maintain referential integrity. (1)

d. Database Positioning. This a variation on retrieval processing, where the usual purpose is to continue accessing a database from a particular position after processing was interrupted for some aside processing operation. With IDMS, this positioning is established with a "FIND CURRENT record" or like command. In SQL, such positioning is normally used in conjunction with cursors for UPDATE or DELETE operations on individual records within a set.

SUMMARY

This paper presented a data oriented approach to program design as an alternative to the more traditional functional application oriented approach. This approach forces the data to be the focal point of the program design subordinating the application. Emphasis was placed on the Logical Access Map (LAM) as the basis of the design. Pseudocode or action diagrams were shown as logical extensions of the LAM to implement the design using procedural or non-procedural languages. A set of guidelines for this implementation were proposed.

REFERENCES

(1) Date, C.J., *An Introduction To Database Systems*, vol. I, 4th ed., Addison-Wesley, 1986.

(2) Jackson, M.A., *Principles of Program Design*, Academic Press, 1975.

(3) Kroenke, D. and K. Dolan, *Database Processing: Fundamentals, Design, Implementation*, 3rd ed., Science Research Associates, 1988.

(4) Martin, J. and C. McClure, *Diagraming Techniques For Analysts and Programmers*, Prentice-Hall, 1985.

(5) Martin, J. and C. McClure, *Action Diagrams: Clearly Structured Program Design*, Prentice-Hall, 1985.

(6) McFadden, F. and J. Hoffer, *Data Base Management*, 2nd ed, Benjamin-Cummings, 1988.

(7) Orr, K., *Structured Systems Development*, Yourdon Press, 1977.

AN INNOVATIVE METHOD OF PROJECT MANAGEMENT IN SYSTEM DEVELOPMENT COURSES

Reagan M. Ramsower, Baylor University
James R King, Jr., Baylor University

Abstract

An interesting situation facing instructors teaching classes which utilize case projects to teach systems development (i.e., system analysis, system design, and senior project courses) is the simulation of a project management environment. This paper clarifies the problem, recognizes its causes, and discusses the results of an approach which used MBA students as project managers to overcome previously encountered problems.

Introduction

Instructors in information system classes face considerable problems in simulating the project manager's role when information systems projects are employed to teach development concepts and knowledge. Courses facing this problem include Systems Analysis, Systems Design, and the Senior IS Project course. The variety of methods used by instructors to simulate the project manager role are:

(1) Instructor selects a student to act as a project manager (instructor selection).
(2) Instructor allows the project manager to emerge naturally from the student group (student selection).
(3) Instructor plays the role of project manager in addition to the teaching role (instructor as project manager).

Each of these methods has inherent problems which will be discussed in the following sections.

Instructor Selection

Instructor selection of individuals to serve as project leaders normally causes some degree of resentment towards the leader by the other students. Resentment typically occurs because team members feel they have been given subordinate roles in the project for reasons they do not understand. Animosity is aggravated when peer appointed leaders manage in either an authoritative or cavalier style.

Instructor selection of student project managers also puts the student project leaders in management roles for which they generally do not have the necessary project management experience and training to function effectively. This naivete can be acute in areas such as determining which team tasks should be performed by project management and which should be performed by team members. Additionally, peer group students serving as project managers are often incapable of motivating the team, dealing professionally with team members on a one-to-one basis, and commanding the respect of team members.

Student Selection

A second method employed by some instructors is to let the manager's role evolve naturally within the group. This method assumes that the individual on the team which manifests the best leadership capabilities will emerge via student selection as the designated project manager. The problems associated with peer students serving as project managers mentioned previously can be more acute when this method is used since a democratic process is often used by the students for selection.

Two intrinsic problems with this method occur when more than one qualified student emerges as a potential project manager, or no qualified students emerge as potential project mangers. The former problem results in alienating students who were eager to be selected as project managers. The latter problem places either the students or the instructor in the position of selecting a reluctant pupil as project leader. As a result, project management is reduced to consensus decision-making or leaderless inaction.

Instructor as Project Manager

A third method of group project management involves having the instructor serve as an informal manager for all project groups. This method places an incredible time burden on the instructor if it is done correctly. Additionally, the instructor is burdened in that he must deal with the problem of privileged information. It is up to the instructor to decide what information acquired through interactions with one group should be passed along to the other groups. Finally, the instructor has to deal with the desire to provide too much help to a struggling group. This can become unfair to other groups and even to the members of the group being helped. An inevitable problem that arises in all three of the methods but is especially prevalent in the last approach concerns the instructor's dual role as project advisor and evaluator (grader) for all projects. The instructor finds it necessary to instruct and advise project groups on methods and techniques which enhance the probability of successful development. A difficult problem is created when the instructor must assign grades to the projects and finds some groups applied all the methods and techniques taught but had poor performance on the project, and other groups ignored the methods and techniques taught but developed outstanding projects. The instructor must also face the dilemma of providing feedback during the project. For instance, is an instructor "giving" a group a grade by telling them when they deviate from desired performance levels during intermediate stages, or is the instructor "letting" the group fail by not pointing out inadequacies when they are detected by the instructor?

Due to the problems encountered with the three approaches presented, an alternative using M.B.A. students in an Information Systems Productivity class as project managers was developed.

Solution

Students in a graduate course on Managing Information Systems Productivity were selected to serve as project managers for the group projects in three undergraduate courses mentioned above. These students were selected because of their technical and managerial Information Systems education, their project management training obtained in the ISM class, and their general management education through the M.B.A. program. The graduate students were assigned to only one group in only one of the three undergraduate courses. The project managers were directly responsible to the instructor in the graduate class. The graduate instructor acted as the Director of Systems Development and the boss of the project managers. The team members were directly responsible to their project manager. There was no formal, direct relationship between the course instructors and team members except as students in their classes.

It was hoped that the problems encountered with other methods of project management in undergraduate systems development classes would be eliminated by creating a project management environment in this method. The following section discusses the results of using the ISM graduate students as project managers for the systems development classes.

Results

Using graduate students to serve as project managers in undergraduate system development courses helped overcome many of the problems encountered with the other methods. A problem almost completely surmounted by the new technique was the lack of project management training. Even though many of the graduate students had no previous experience as project managers, the graduate program appeared to have given them a working knowledge of group management techniques, interpersonal communications, and a better understanding of the responsibilities of management and those of team members.

A second problem that was overcome was the resentment of peer project managers by team members. The team members did not display any resentment towards the graduate project managers and did not view them as peers. The problems of no leader emerging within a group or too many potential leaders emerging when the manager selection decision was left up to an evolutionary process was overcome since the project leaders were appointed by their boss, the graduate class instructor.

Using the graduate students as project leaders definitely relieved many of the problems of the instructors in the systems development courses. Since team members were required to take problems to the graduate leader, the instructor was called upon only when the graduate student was incapable of providing a solution or when a possible solution was out of the control of the graduate student. Even though this occurred numerous times, the amount of time the instructors had to actively participate in the projects was radically reduced. The instructors were able to keep personally removed from the projects because of this reduced interaction. This meant there was less of a chance to try and help those teams that were struggling. It also meant that there was less privileged information to be passed along and it was easier to keep from passing along that which was known by the instructors.

The instructor in the graduate class found this approach created an outstanding environment to teach project management techniques. Project managers were able to test in a real development environment project management concepts and productivity techniques. The students learned from their mistakes. Class discussions were vivid, realistic, and meaningful. The students often craved answers to questions of:

(1) How to motivate?
(2) How to give directions which would be carried out?
(3) How detailed should instructions to team members be?
(4) What style of management is appropriate?

When the instructor taught this class without the aforementioned development setting, the students often found discussions of motivation, communication, and management style boring and tedious.

Even though most of the problems with previous methods were overcome or mitigated by using graduate students, one important deficiency still existed. Some of the graduate project leaders were ineffective managers. For many different reasons, they failed to provide the necessary inspiration to their team members to successfully accomplish the project.

The two most important reasons for this lack of management success related to the knowledge base of the project leaders. The technical skills of the project leaders varied widely. Inadequate technical support by a project manager meant that the instructor had to fill the gap and this often caused the team members to lose respect for their project leader. This was particularly prevalent in the system design and applied software development courses where programming was a significant amount of the development

USING DEBATE IN THE CLASSROOM TO ENHANCE IS LEARNING

Dr. Melvyn Weisel
School of Computer Science and Information Systems
Pace University, NY

ABSTRACT

This paper represents the author's efforts to utilize debates as a mechanism for improving instruction in the information systems classroom. Specifically, it describes this implementation in regard to a graduate course entitled "Policy and Practice in Information Systems". The author details the methodology used for such an implementation and describes the objectives for such an implementation, from both the instructor's and the student's perspective.

While the paper does present student opinion as to the values of debate in the classroom, it does not attempt to measure the effects of such an implementation on such variables as student achievement, retention of information, or overall conceptual learning. The author recognizes that this would require an analysis of data that are not yet available for this purpose. Nonetheless, the author presents a very positive perspective based on his own perceptions and on the feedback that has been generated from the students involved in this effort.

RATIONALE

I really don't recall my initial exposure to the concepts of a Socratic approach to education. It seems like I've always had an affinity for the use of questions as a means of leading students to "discover" principles on their own. I imagine that many lecturers in Systems Analysis share this feeling, since it really is just another form of careful systems development. After all, just as the output must be decided first and then used to help determine the structure of the system, so the objective(s) of the lesson must first be clarified in the instructor's mind. From there the instructor then works backwards by developing those questions that will cause the students to proceed from their present state of awareness and skill towards that objective. Yet, when I think back on my own educational experiences it is sad to think how few of my instructors have taken the time and trouble to plan in this manner.

Let's face it! Most of us have gone through our entire university career with our classroom experiences entirely confined to a structured delivery system in which the professor lectures and the students take notes. And, even worse, some of these sessions were downright boring. Small wonder since many of the lectures were delivered with the instructor simply reading

from a prepared text or sheaf of notes. The minor deviations from this theme, such as having one of the students deliver a report or, on even less frequent occasions, using a media presentation, simply served to confirm the tedium that existed in so many classrooms. Even in those classes where questions were frequently used to encourage student participation, or where exercises and/or homework were reviewed as a means of enforcing the development of new skills, many classes quickly fell into a dull routine continuing day after day, interrupted only by an occasional quiz or exam. And now we, the former students, have become the professors and, according to Steven Cahn in his recent commentary on "Ethics in Academia", many of us are continuing the same laborious routines established by our predecessors. It would certainly seem that a search for alternatives to the "traditional" instructional modes might be fruitful.

An opportunity to explore such an alternative presented itself to me while in my current position as Associate Professor of Computer Information Systems at Pace University in New York. As part of a new graduate program, an MS in Information Systems, a new course was developed entitled "Policy and Practice in Information Systems". This was envisioned as a "capstone" course for those students near completion of the degree requirements so that they could utilize the learnings gained from courses previously taken. As can be gathered from the title, the course content deals with developing an understanding of those principles from which one can develop a coherent set of policies and practices for an Information Systems department or division.
Obviously, a course of this nature will be most effective if it focuses on those issues that are the subjects of concern for today's journals and periodicals. These issues, in turn, frequently contain materials that are somewhat controversial and lend themselves most easily to some form of debate.

STRUCTURING THE "POLICY AND PRACTICE" COURSE

The class in "Policy and Practice" began in the Spring semester of 1988. It was scheduled for one three-hour session per week throughout the fifteen week semester. It was decided to hold eleven debates over the course of the semester, with the other four sessions being devoted to introductory work, testing and general lectures.. Twenty-three were students enrolled in the course. I had planned each debate team to consist of five or six participants, meaning that each student should be an active team member for five debates over the course of the semester. In addition, in

order to provide identifiable leadership for the team effort, every debating would be headed by a captain. The captain would be responsible for coordinating the individual efforts of the team members. Since there were to be eleven debates, twenty-two of the twenty-three participants would be able to serve as team captain for one of their five debate sessions.

As one might surmise, not every topic in any given course curriculum may be suitable for a debate. In fact, the course content of "Policy and Practice" included several topics which could be better handled in a more traditional manner. For example, much of the material related to subjects such as Planning or Risk Analysis would be better suited to a lecture and open discussion that to a two-sided debate session. The problem was, therefore, to develop a course structure that would allow for presentation of these topics in an other-than-debate mode, while still facilitating the debates for those topics most suitable. It was decided that an appropriate solution would be to divide the typical class session into three components: an opening lecture, team meetings and a debate. This would allow for presentation of materials that were basically inimical to the debate format and would also break up the three consecutive hours of instruction into more palatable hour-long components. It also must be realized that, in practice, the actual time divisions are only loosely maintained. Should a specific lecture or debate require more than the single hour allotment, this would be achieved by shortening the time allocated for team meetings.

During this middle hour (the team meeting) session, the class splits up into groups in accordance with a debate schedule (distributed early in the semester). This time is used to allow the team to review the periodicals and articles which have been gathered by individual team members and to generates a strategy for the debate, itself.

The debates all follow the same basic format. This format was presented to the students at one of the introductory sessions, in which it was also carefully explained that the overall objective of the debate was to review the materials germane to the topic, not to try and emerge victorious. The format allowed each team to make a brief opening presentation which supported their position. after which each team would alternate in asking a question of their opponents. These questions were prepared during the team meeting sessions. It was during these questioning periods that some very lively discussions actually took place. In a debate about the merits of "charging out" the

costs of information systems to the users, for example, questioning about the ethical and moral convictions of the opposing teams brought some heated, but good humored, responses. Throughout all the debates, in fact, an overall tone of camaraderie and good spirit was always present. I imagine that everyone realized that today's opponent might very well be tomorrow's partners on another topic.

In my desire to avoid dominating any discussion or prejudicing any issue, I tried to remain quiet for at least the first half of the debate session (although sometimes without success). Since every team had been asked to provide me with a written outline of the opening position statement and a list of questions that they were prepared to ask of their opponents, I could sense the direction that the debate would take in advance and was able to insure that the discussion remained on the "right track" with a minimum of actual intervention during the debate.

The overall format worked very well, and the only complaints voiced by the students were related to the shortness of the debate portion of the session. (They would undoubtedly have preferred to dispense completely with the opening lectures). As it is, on many occasions, several of the students would remain in the classroom after the regular three hour session had ended and continued to voice their positions on the issue that had just been discussed.

Every debate was also reviewed during the opening part of the following class session. At this time I would generate feedback from the non-team members in the audience and solicit their feelings and opinions about the subject debated during the previous session.

OBJECTIVES OF DEBATE - INSTRUCTOR'S PERSPECTIVE

It is important to recognize that, in addition to structuring the work the student must do in preparing for a debate topic, the debates provided a strong motivational force. Nonetheless, I was concerned about those times that the student was part of the "audience" and not a member of one of the two teams at the front of the room. I was very much aware that these students must be encouraged to actively participate in all of the sessions, even those in which the student is not an official team member. Since each student will be an actual debater only once every two to three weeks, this participation during the "off weeks" is significant.

One must recognize that the motivation provided by the debates can be a fleeting phenomenon in that it is

sometimes built on the fragile foundation of success. As such it is incumbent upon the instructor to insure that the actual class sessions are not allowed to cause a student to leave with a feeling of having "lost" the debate. The instructor must do everything in his or her power to make certain that the students fully understand that the success of a session is based on how thoroughly a topic has been explored and not on whether one side or another has prevailed. In fact, it is the obligation of the instructor to insure that no session ever ends with an obvious advantage being held by one team or the other. In this regard, those topics that do not allow for a full exploration and give and take by two different positions are not really suitable for these debates and should best be dealt with by lecture or other type of lesson.

Another concern was that the debates must stimulate the students to research, conduct literature reviews and develop knowledge in those areas which are being debated. To help the students become aware of the resources available for this purpose, one of the opening lectures was devoted to a review of the Pace (and surrounding area) facilities available for this purpose (see Appendix A) and to proper research methods. In addition, I distributed folders that I had prepared in advance, containing articles and sources appropriate to each topic, to the appropriate team captains. These were designed to serve as "seed" documentation about the subject at hand.

Finally, the debates should create a feeling for team effort and cooperation among team members. In this sense the debates are viewed as preparing the students for those future occasions in the business environment when team cooperation is critical to the success of a project.

OBJECTIVES OF DEBATE - STUCENT PERSPECTIVE

There should be no misconceptions about the fact that every student wants to end each session as a winner. However, for the most part, this desire is really an extension of the student's need to demonstrate his or her abilities and skills to the instructor (and to the class). Most students will look at the debates as an opportunity to do so and, provided that the ground rules are presented at the very beginning of the semester, will have no difficulty in accepting the "no winner" concept.

An important concern for almost all the students related to the grading system for the course. This concern was expressed early in the semester by some students, who felt that the hours spent in researching and preparing for the debate were not adequately displayed

because of the finite time allotted to the debate, itself. This concern was alleviated by allowing the students to submit written documentation substantiating their efforts along with the basic debate materials.

In general, the students looked forward to the debates as a means of providing more interesting learning experiences than the typical classroom approaches. They are also interested in discussing those issues that have relevance to the world of work and to the experiences that they have gained outside the university environment.

SELECTING THE DEBATE TOPICS

Obviously much time and effort must be spent in selecting topics that are appropriate for debate. Most of this work was done in the six months that preceded the opening of class. There were several rules of thumb that were used in reviewing potential topics.

First, was the topic integral to the subject at hand? In this specific case, did the topic relate to the "Policy and Practice" curriculum? In this regard, I was quite flexible in that an issue such as "Star Wars" was included as a viable topic and, eventually, debated. It is interesting to note that most of the ensuing discussion centered around the definition of "fail safe" and degree to which any program or system could be made fail safe, not on moral or ethical issues.

Second, were there clearly two different, yet defensible, points of view that emerged on examination of the topic? Were these roughly equivalent positions in that one could feel comfortable taking either side? It should be noted that, in many instances, two teams were simply assigned to a topic and the team captains were allowed to determine, jointly, which team would take the affirmative and which team would take the negative position. This emphasized the fact that the debates were being judged more on a review of the issues than on the presentation and defense of a position.
In this sense, the debate on centralization versus decentralization of the IS function was an excellent choice in that it provided for two relatively equal positions.

Third, was the topic researchable in that articles and literature about the subject were available in the university library or a nearby facility?

Fourth, was the scope of the topic sufficiently broad enough for the students to expend time and energy in research and yet narrow enough to be covered in the time allowed for a single debate? Obviously, this rule of thumb

was very difficult to enforce. In point of fact, many of the topics would have warranted additional debate time, should some have been available.

THE ROLE OF THE INSTRUCTOR

One of the first tasks that I faced was to develop a meaningful schedule and distribute the schedule early enough for the students to plan their overall semester's research. This presented somewhat of a problem in that the schedule obviously depended on the actual class roster which, experience had taught, was not reliable until after the first class session. This meant that I had to develop the schedule between the first and second schedules. This was further complicated by my desire to ensure that the makeup of the debate teams was changed after every debate, rather than keeping the team rosters frozen throughout the semester. This constantly changing mix of participants hopefully would generate maximum interaction during the team meeting sessions and prevent undesirable fixed alliances from forming. Despite the time pressure, the schedule was available by the second week and debates actually began by the fourth week of the semester.

As mentioned earlier, one of my major concerns was to minimize my influence during the debate, itself, by remaining quiet for at least half the session. Nonetheless, I was careful to come to each debate prepared with questions that I could use to make certain that both positions were equally meritorious and that all the issues were adequately presented. These questions had to be used on several different occasions. For example, during the debate focused on our ability to control the future development of automation, most of the students felt naturally comfortable taking the affirmative position and support for the negative side had to be carefully built up to maintain proper balance.

In addition, for those debates that were proceeding without enough controversy or notable enthusiasm, I allowed myself the luxury of playing "devils advocate" and stirring things up a bit. This happened only once or twice and the participants generally needed very little prodding to get involved in the spirit of the debate.

STUDENT FEEDBACK

Student input was solicited at several times throughout the semester. At the very first class meeting, for example, the class was presented with the list of potential topics (see Appendix B) for debate asked to respond to them. This proved both useful and motivational.

After the ninth week of

the semester (halfway through the debates) the students were asked to express their opinions about the debates. These were uniformly positive, although the atmosphere in the classroom was quite open and I actually encouraged critical feedback so that it could be used to improve future class sessions.

Aside from the earlier mentioned concerns about grading, the students had very little critical commentary to make throughout the semester. One suggestion, however, about the size of the debate team seems worth considering. Several students felt that, since the time was limited, smaller teams (maximum of four members per team) would have provided the participants with a better opportunity to express themselves individually. This constraint is logical and, should future class size allow, worth implementing.

SUMMARY

Not only has the response of the participants to the debate format been overwhelmingly positive, but their responsiveness, in terms of effort and achievement, has been most gratifying. The amount of preparation for these debates is obvious, both in time spent by the individuals involved and in the results that can be seen from the degree of professionalism and competence level displayed during every session.

It is recognized that, pending an evaluation based on statistics gathered over several years of applying the debate format in the classroom, any conclusions drawn about the effectiveness of the format is premature. These data are being saved and the statistics will be presented in the future. I am confident that they will support the overwhelmingly positive verbal appraisals that have been in evidence thus far.

EVALUATING THE INDIVIDUAL FROM WITHIN THE SYSTEMS DEVELOPMENT PROJECT TEAM

Thomas W. Dillon
Department of Computer Science
Richard A. Henson School of Science and Technology
Salisbury State University
Salisbury, MD 21801

ABSTRACT

The DPMA model curriculum recommends that CIS/86-8 Systems Development Project be a capstone course with an emphasis on project development using project teams.[1] The concept of group or team evaluation calls for a different form of evaluation than the traditional evaluation methods applied to individuals. Most of all, it is difficult to evaluate an individual performance from within a group project team. This paper outlines an evaluation method that discusses the division of group tasks for each individual and provides for evaluation of individual members within the systems development project team. Though this method is designed for CIS/86-8 Systems Development Project, its methodology may be applied to other group- or team-oriented courses.

INTRODUCTION

Being a systems development project instructor caused me great frustration when I would sit by and witness a project team at work. It always appeared that one member of the group would excel above the others, and a second member of the group would be academically "carried" by the group. Frustration came in two ways. First, the excelling student could not be graded above the project team, and second, the student being carried by the others could not be graded below the project team. To prevent my frustration and to better provide the students with a more effective evaluation, the following individual evaluation method was developed. The basis for this method is a combination of the contract grading system and the mastery approach of learning.[2, p. 298-304] In addition, self-evaluation plays an important key.

A PROJECT AND COURSE DESCRIPTION

This evaluation method is applied to a traditional second semester course in the systems development process. Students are usually seniors from the Information Systems Management sequence of the Business Administration degree or seniors in the Computer Information Systems track offered by the Department of Computer Science.

The systems development project is initially distributed to the students in the first semester course; Information Systems Analysis and Design (CIS/86-5). The case study project, drawn from the instructor's consulting experience, is analyzed in the first course individually by each of the students. In the second course (CIS/86-8) the students are grouped in teams of four or five, depending on enrollment figures. The design and implementation of the

project is performed in one of two microcomputer data base languages available on the college campus.

The student project teams meet with the instructor approximately every two to three weeks. In this way the instructor remains informed of group and individual progress.

THE EVALUATION TOOL

In the second week of classes, the students are placed into project teams. Teams are united in a random method by alphabetical listing to prevent faculty or student bias in project team development and organization.

The first assignment given to each project team is to further analyze the case study and to divide the project into modules. Each module is then further analyzed and divided into tasks. These tasks are listed by the members of the project team. Although each project team is analyzing the same case study, each arrives at slightly different task lists.

The second assignment, distributed in the third week, is the first model for the evaluation tool to be developed by the students. The tasks previously listed by the students in the module-division, task-list analysis are now reworked and written into job descriptions. The members of the project team are directed to divide the tasks and to develop a job description for each member. The number of job descriptions will be determined by the size of the project team. Each individual, with the input and consent of the project team, writes a job description listing tasks that he/she will complete as a part of the total project. Initial job descriptions are collected and reviewed by the instructor. Before returning the job descriptions the instructor attempts to evenly balance tasks between team members, and to mark and comment on tasks that may be too broad or not written with enough detail.

In the seventh week, with systems projects progressing on schedule, the project teams are again asked to submit job descriptions with detailed task divisions. Students are instructed in the expected changes in group roles [4] and the expected changes in job descriptions. They are reminded that tasks on job descriptions are changed only with consent of the group. The instructor again collects and reviews the tasks listed for each job description. Written comments are recorded and attempts are made to evenly distribute tasks between members of the project team.

The final job descriptions, to be submitted by each project team, are received by the instructor in the tenth week. Team members are again instructed that tasks on job descriptions may change and the roles of members in the group may be changing also. The students are directed to follow the guidelines listed below for the final job description. Guidelines are provided to simplify the instructor's paper work and to maintain consistency in evaluation format.

Job Description Guidelines

1. Job Title - State the title assigned individually to this team member.

2. Brief Description - In paragraph form give a brief description (not more than three sentences) for the previously-stated job title.

3. Task List - List approximately ten to fifteen tasks to be performed. Include at the end of the task list these additional evaluation criteria:

a. Attended all scheduled project team meetings.[3]
b. Was prepared for all project team meetings.
c. Participated actively in project team meetings.

The task list is to be double spaced on one sheet of paper. (Examples will be distributed)

The job descriptions and task lists can be photocopied and used as evaluation tools. Each member of the project team (including the person being evaluated) and the instructor receives a copy of the job description and task list. Using a rating system from one to five, with five being the highest score, team members, instructor, and the individual being evaluated, rate team member performances based on the task criteria listed.

A sample weighting for evaluation of individual members within the team may be 25% each for (1) overall project grade, (2) self evaluation, (3) team members average evaluation, and (4) individual evaluation of tasks within the project team by the instructor. To further enhance the self-evaluation premise, each student should meet with the instructor, after all evaluation forms have been collected, for a discussion on the task completion and team interaction.

ROOTS OF THE METHODOLOGY

This evaluation process is a combination of the mastery approach for learning and the contract method of evaluation. [2, p. 298-304] Each student lists the tasks that will ultimately be the method of evaluation. In the mastery approach to learning the instructor, not the student, lists the objectives to be completed to receive a passing grade. The method described in this paper permits the student to go one step further in the learning process by listing his/her own evaluation criteria. Since the student develops and actually creates the job description and the task list to be submitted to the instructor, the student is in turn creating the evaluation contract and the goals that must be reached for adequate evaluation. This method is a mixture that will hopefully provide all of the advantages of each form of evaluation.

THE OUTCOMES

The outcomes from this method of evaluation, in my case, were very enriching. I was able to observe three distinct positive results. First of all (the motivation behind this method), I was able to evaluate individuals in the team environment fairly and without frustration. Those who were carried by the project team could not meet the tasks listed

in the job description, and those who excelled above the others in the project team were easily identified.

The second result concerns the concept of self-evaluation. The student is actively involved in establishing the framework for his/her evaluation, and since the student is permitted to share in this process, three key outcomes are evident. [2, p. 296]

1. The student gains experience in evaluating individual strengths and weaknesses.

2. The student is encouraged to pursue more responsibility in the project team.

3. Students who participate in self-evaluation tend to be fair and objective and often are harder on themselves than an instructor would be.

The third and final result of this method of evaluation is that each student is provided with an opportunity to evaluate peers with a clearly-defined and objective evaluation tool. This is a vital experience for all students as they prepare to graduate and enter the working word.

CONCLUSION

This method of evaluation provides two benefits; team members are permitted to objectively evaluate each other in a near-business setting and students in a systems development project course are provided with a fair evaluation procedure. Objective evaluation of peers in a systems project team is applicable to a real-world situation that students will soon confront. Providing them with the tools and practice for this soon-to-be-realized encounter will enable them to deal better with this experience. Evaluation, one of the most important aspects of educating college students, must be impartial. This method provides for the most objective technique I have encountered, fairly evaluating all members of the project team.

BIBLIOGRAPHY

[1] DPMA. DPMA Model Curriculum for Undergraduate Computer Information Systems, Second Edition, Park Ridge, Illinois, July, 1986.

[2] Kirschenbaum, H., Napier, R., and Simon S. B., Wad-Ja-Get? The Grading Game in American Education, New York: Hart Publishing Company, Inc. 1971.

[3] Overbey, John T., "Student Project Guide for Systems Analysis and Design Courses," Proceedings from the Sixth Annual Information Systems Education Conference, San Francisco, 1987, pp. 181-184.

[4] White, K. and R. Leifer, "Information Systems Development Success: Perspectives from Project Team Participants," MIS Quarterly, September, 1986, pp. 215-223.

IMPLEMENTING TEAM PROJECTS IN AN ADVANCED PROGRAMMING CLASS

Prof. Susan K. Lisack, CDP
Computer Technology Department
Purdue University
W. Lafayette, IN 47907

ABSTRACT: Employers look for "team players". Although we as educators may discuss various team organizations for implementing large computer systems in our classes, the very nature of giving individual grades, grading on the curve, and threatening dire consequences for academic dishonesty may discourage true team work in educational settings. In our CIS curriculum at Purdue University, we decided to have the students work in teams in the advanced programming class. This gives them a chance to experience some of the benefits and frustrations of team interaction. At the same time, they are able to implement more programs as a team than each could do individually during a semester. This paper discusses several aspects of course preparation and course management which should be considered before instructing such a class.

BACKGROUND

In the Computer Technology (CPT) Department at Purdue University, our students currently take four programming courses. The first course emphasizes the development of structured programs, the use of modules and the importance of logic design before coding. In addition, students are learning commands for using a new computer system, new editor, and possibly a new programming language (BASIC). In the second course, students are again learning a new computer system, a new editor and a new programming language (COBOL), but can immediately proceed to implementing more complex programs due to their experience in logic design. Finally in the third course, our students stay on the same computer, using the same editor and programming language (COBOL), but the course objectives include teaching file structures and data structures (with accompanying COBOL additions), which are demanding topics for most students. Over the course of the years, our faculty has discussed and experimented with team projects in a variety of computer classes; they have been implemented in the upper level systems analysis and design classes. However, team projects did not work well in the beginning programming course, probably due to lack of experience, and large variances in student backgrounds, abilities, and motivation to work. We have found that the first programming class generally weeds out those students who can't succeed in the curriculum, or who don't like the demands of computer programming. In addition, as noted above, each of the first 3 courses teaches significant amounts of new material, which leaves little time to devote to team projects.

This left our fourth programming class, Advanced Topics in COBOL. In fall, 1987, as

the result of a curriculum update, the specific content of this course was flexible. Its initial objectives were to teach any COBOL topics not covered in previous courses (e.g. using and calling subprograms, using ReportWriter) and to prepare the students for the upcoming data base courses by reviewing and comparing different file structures, working with pointers and linked lists, and presenting additional data structures topics such as B-trees. It seemed like a good setting for introducing team programming projects. With their previous experience, the students could work on a system containing multiple files, each file having a different data or file structure. They would already have the necessary background to read and understand the system requirements, assign projects to individual team members, and begin work.

COURSE PREPARATION

One of the first implications of team implementation of a system of programs is that the course instructor must have documentation for the entire system of programs very early in the semester. In addition, to get the course off to a good start, decisions must be made about how teams will be assigned, how they will interact with the course instructor, how the teams will be graded, and who will prepare the needed data files. Each of these aspects will now be discussed in more detail.

PREPARING PROJECT SPECIFICATIONS

Creating a system of programs for the class is one of the most important, but potentially time-consuming, tasks in preparing for the team project course. If at all possible, the students should receive their project specifications during the first week of classes. These specifications should let the students know the specific contents and record layout of each file, the processing performed by each program, and the layout of any reports produced. However, since specifications are rarely perfect in "the real world", it is desirable to have a few inconsistencies or omissions in the project specifications. Some intentional errors might include a report field whose length is inconsistent with the field length from the data file, a missing calculation, no mention of the order of storing records in a file or list, etc.

When preparing project specifications, I look for a system of programs that:

(1) is a business application
(2) contains four or more inter-related data files, representing a variety of file and data structures, such as:
- a random access (or relative) file accessed via a hashing algorithm
- a random access (or relative) file accessed via pointers
- an indexed file (where the index is either maintained by the system, or by the application program)
- a sequential file

(3) contains nine to twelve programs, each of which require accessing or updating multiple files in the system.

CHOOSING DATA FILES

The advanced COBOL text that we use (Advanced Structured COBOL by T. Welburn) appropriately points out that direct (or random) access files are seldom used in business data processing, but they are used as the underlying file structures in a number of data base management systems. Since most of our students proceed on to a required sequence of data base courses, first-hand experience in accessing and performing various updates to different file types helps them with upcoming data base concepts.

Inter-relating the files also helps the student in future data base design projects, where there is a potential for scores of inter-related files. Try to set up files that are interconnected in different ways. For example, given a vendor file, and a product file, you might find all the products supplied by the same vendor by following a pointer field from the vendor record through multiple product records until the pointer field is zero. But to find all the vendors for the same product, instead of pointers, there may be a list of vendor identifiers stored in the product record.

Forcing the files in the class projects to contain a variety of file and data structures may cause your system of programs to be unrealistic, or less efficient than it could be. Use this to generate class discussions later in the semester about alternative file and data structures. Discuss whether they are better because they would be more efficient in processing or because they would result in simpler program logic for certain applications, like maintaining the files.

SELECTING COMPUTER PROGRAMS

Since this system of projects is being implemented in an educational setting, I like to make sure that each student is getting similar opportunities to design, code, and test programs. This affects which programs are included in the system of projects. For example, we want the students to update files (add, delete and modify records), and/or validate the update files used as input. So I write specifications for three or four separate programs that each update (or validate) primarily one file. If in the process of updating, other files are affected, so much the better. For example, we have used an inventory purchasing system with a vendor, product and order file. If a vendor is being removed from the vendor file, then that vendor must also be removed from the products which list that vendor, and there must not be any orders to that vendor still in the order file.

The next set of programs that I like to include involves a series of programs that need to be performed in a certain sequence, where one program produces a data file required as input for the next program, etc. This forces the students to evaluate the input and output requirements of each program to determine the best order of implementation (especially if the instructor does not provide test versions of these data files). To continue the vendor/product/ order example, one program might check all products to see whether they need to be reordered, and create a Products-to-Order file. Another program could then use this file as input, sort it by vendor, print the orders for each vendor, and add records to the order file. A third program could process received product data, and modify or delete records in the order file.

Depending on how many programs the students can complete in the allotted time, you can also specify report or query programs, each of which accesses two or three files. Again from the vendor/product/order example, reports might include all vendors and the products they supply, all products and the vendors who supply them, all vendors with any orders outstanding, all products which are currently on order, etc.

Another consideration in working with a new system of programs is determining where the data will come from to test the programs. The students need to recognize and gain experience with several of the possibilities. For example, the data file may already exist in the appropriate format, but it is too large to use for initial test purposes. Another scenario is that the data exists, but not in the correct format for the new system. It may currently be on paper, in a file that needs to be rewritten to a new format, or in parts of two or more different data files that must be combined. Part of the team's effort may be to write one or more quick file load (and possibly file dump) programs, or to learn how to use a utility program to load files with data. The specifics of this will depend on the system you are using, and the types of files involved. I generally have the students create at least one file load program, and then I provide programs to load and dump the other files.

TEAM ASSIGNMENTS

The size of the programming teams affects the number of programs needed in the system specifications, if you want each team member to be working on a different program. I have found that three or four person teams work well. If the teams get any larger, it is difficult for them to find times that the team can meet together.

Even though scheduling meetings in the business world is not always easy, at least on the job people usually work from 8:00am to 5:00pm. In the classroom, you will be faced with students having different class schedules, some students working during the day, others

working nights or weekends, others having young children at home that require babysitters, some commuting to campus from out of town, etc. Since scheduling regular team meetings may be a problem for some students, we responded by scheduling one of the class periods as a two-hour lab, which is primarily for the purpose of team interaction. Team members are also encouraged to find another hour during the week to meet together.

How to assign the teams must also be decided. Several options are:

(1) assign the teams randomly,
(2) take requests from class members, along with any special considerations
(3) look at grades from previous programming classes, and make "fair" team assignments (e.g. don't put all 'A' students on one team, and 'C' or 'D' students together on another team)

I have used a combination of the last two methods to assign teams. The first day of class, I ask each student to submit any requests they have for team mates (or class mates they would rather not team with), and the reasons for the request. I do not promise to fulfill all requests. At the end of the week, I assign teams by first trying to honor the requests submitted, and then grouping the remaining students "fairly", based on any previous grade information.

Since these team assignments are made so early in the semester, you must have a policy to deal with team members who fail to participate to any great extent in the course in general, or on the team. It is not fair to punish an entire team when a student has, in essence, dropped the class. If the student formally drops the class early in the semester, I may reassign the remaining team members to other teams, or combine two small teams. If the student is still enrolled in the class, I encourage the active team members to keep calling and leaving messages for their wayward team-mate. They also document the lack of participation in weekly memos to me. Then if there is still no activity from this student at mid-semester, I group 2 teams for system testing purposes.

GRADING POLICIES

How well the students work together as a team is dependent on both the individual students' personalities and how much their grade depends on the team completing the entire project. If the entire grade is determined by the end products of the team as a whole, then you will find some of the good students doing the work of other team members, just to ensure their own good grade. On the other hand, if the team members are only graded on their individual programs, then they have no incentive for assisting their team mates. Hence, I have tried to incorporate a combination of individual scores along with team ratings in determining a student's grade.

My method for assigning individual grades has been to have each person on the team responsible for implementing a specific set of programs. The team sets due dates for each program. Then each person's project is graded individually according to the established standards for written documentation, coding style, and completeness of testing.

The team grades are determined from one or more written team evaluations, and how well the entire system of programs works at the end of the semester. Team evaluations ask each student to confidentially rank each team mate on items such as regular meeting attendance, completing work on time, willingness to assist team mates, and familiarity with the project. In practice, students are not always honest on these team evaluations because they don't want to hurt the grade of a team mate.

The requirement to complete the entire system of programs is a much better incentive for continued team interaction. However, if this is a requirement, it is important that the programs assigned be small enough that they can easily be completed within the semester's time. The teams should have a week or more at the end of the semester to thoroughly test their programs. If practical, the instructor should set up a final system test using instructor provided data files and test procedure.

Another team incentive which I would try in the future, is to require ongoing submission of program logic designs and test plans, checked and approved by the other team members. Then if the instructor discovers serious errors or omissions, the entire team would lose points.

COURSE MANAGEMENT

It is important for the instructor to realize that he or she must immediately be familiar with the whole system of programs, and not just one program at a time, as may be true in other programming classes. This is because the teams may choose to implement the programs in different orders, and each team member is working on a different program. In order to be consistent between teams, I create a file of "Specification Updates" which I update each time I clarify or correct the project specifications for one team. This file is made available to all class members.

The course instructor needs to decide what role to play in the course. One role is that of "user requesting the system", in which the instructor assigns the project to the class, assigns the teams, asks for the system by the end of the semester, and then is present to clarify project specifications. My experience with this approach has been that, without intermediate deadlines, most students will get caught with the project not completed at the end of the semester.

Another role, which seems to work better, is that of senior project manager. Start by assigning the teams, distributing the project specifications and computer accounts, and discussing how the teams should work together. Remind them:

(1) to review, discuss, and look for errors in the project specifications,
(2) to look for programs that are dependent on data produced by other programs,
(3) to plan on designing logic before coding,
(4) to develop a test plan, and
(5) to perform team logic walkthroughs.

Then request regular updates on the team's progress in the form of a short weekly memo (which includes recent progress, questions, and requests for assistance), and submission of evidence of work completed.

My first request from each team, after they review the project specifications, is for a list of the team member's program assignments, and a schedule of delivery dates of program logic, program test plan, coded program (merely typed in), and completed program. If these schedules are not complete and reasonable, I ask for changes. (e.g. Is the set of programs that must be run in sequence being implemented in the proper order? Are they waiting too long to complete the logic for the first program?) The final schedule is kept as a reference for determining whether individual deliverables are late. There should be some penalty for each item that is late.

As course instructor, I try to review the submitted logic for errors, and return it for resubmission if necessary. In the future, I would also assess some penalty to the team, as well as to the individual, for failure to catch obvious mistakes or major omissions in the logic.

FINAL SYSTEM TESTING

In our first programming courses, we find it easier to grade projects that all have the same output, so we tend to provide one or more sets of test data which the students must use to create output before submitting their programs. Some students then introduce data dependencies into their programs to make them work with the test data provided. Hence, it is important to check the students' abilities to completely test a system of programs. At the same time, the course instructor would like to know that the teams' programs are producing correct output in a variety of situations, without having to perform a different set of calculations for each team.

My approach is to have the individuals perform their own testing during the semester when submitting individual programs. I ask them to add handwritten notes to their output indicating where various situations have been tested. Then at the end of the semester, I provide a set of test files and a test procedure which the team must follow in a live

demonstration of their programs' capabilities. This permits the final team evaluations to be uniform.

CONCLUSION

Taking a course involving a semester-long team programming project is a valuable experience for a student preparing for an applications programming or programmer/ analyst position. It can also be used to give students a foundation in data and file structures as an introduction to data base studies. Taken as an intermediate or advanced programming course, the team projects course not only improves a student's individual programming skills, but also provides experience in working with and encouraging other team members toward a common goal. It may be a student's first realization of the total process involved in implementing a new system, including creating initial test files and performing complete program and system tests designed solely by the student team.

Instructing such a course requires a lot of course preparation, and a different mode of interaction with the students during the course. The instructor must prepare and be familiar with an entire system of programs, instead of just one project at a time. Then the instructor must act as a master project manager, trying to get multiple teams to complete their projects on time.

Cross-Course Consulting Projects: A Technique to Relate Software Development to More Advanced Courses

Barbara Beccue, Janet Cook
Applied Computer Science Department
Illinois State University, Normal, IL 61761

ABSTRACT

Introductory System Development classes use projects in which students complete an analysis and general design of a software system. Senior courses such as Human Factors, Testing or Security, teach subjects which benefit from short-term practical projects. We decided to use students from specialized advanced courses as consultants to project teams in the System Development course. This paper describes the learning experience which we used and discusses the benefits and problems involved with using this particular activity.

INTRODUCTION

As teachers, we try to communicate how the material taught can be used in settings outside the classroom and to get students to appreciate the interrelationships among the subjects of different courses.

System development material, in particular, is too extensive to be taught in a single course. Usually an introductory overview of the system development process is followed by a variety of higher level courses which focus on aspects of the process in more detail. Unfortunately, many students consider more advanced courses such as DataBase to be stand-alone topics unrelated to system development.

At Illinois State University, we wanted to find a way to show the students in the introductory course the complexity of the system development process and their need for greater experience. For the advanced students, we needed to provide realistic settings in which to apply their learning.
It is difficult to find "live" projects which involve only database design or system tests. One learning experience which seemed to meet the needs of both types of courses was some type of joint assignment.

Our junior level Systems Development course [1], like many others [2] [3], uses projects in which students complete an analysis and general design of an information system for a local user. However, the naivete of the students and the need to fit the project into a single term frequently produce products weak in data organization, user interfaces, security, and user support materials. Senior level

courses in our program cover most of these areas. As an experiment, we decided to use students from a specialty course as consultants to the project teams in the introductory course.

At the time of the experiment, we found nothing in the literature on the topic of using a common project in two different courses. Since then, Gallupe, Shonoski, Nelson and Cameron have reported success in an experiment which coordinates projects among business and IS courses taken concurrently. [4]

ENVIRONMENT

As stated above, juniors in Computer Information Systems take a core course in Systems Development. Seniors must take Topics in Systems Design.

The Topic taught in Spring, 1987 was information system security. It is difficult to depict the complexity of the environment in which systems operate without an actual system to study, yet managers of working systems are reluctant to have students come in and seek to locate the risks inherent in their activities. The real projects in the Systems Development course provided the seniors an opportunity to work on full-scale systems.

The Topics course followed the same chronology as the System Development Life Cycle (SDLC) used in the Systems Course.

Analysts know what physical and logical hazards are likely to affect computing systems. Users know the human and physical hazards of the working environment, and can decide which items need to be protected. The analyst and user must collaborate in determining what significant risks the system is vulnerable to and how much potential damage each risk entails. Their synergism improves the quality of the product being produced.

By arrangement with the three faculty teaching sections of Systems Development, the Topics students were brought in as security consultants when the development teams had finished reviewing the existing systems. The Topics students were assigned to consulting teams of 1-3 persons. (The preferred team size was two, but adjustments were made to accomodate the students' and users' schedules.) The consultants first interviewed the project team to see if they had gathered enough data to complete a risk identification for the system. If not, the consultants primed them with a new set of questions and sent them to check with the users again. Finally, the consultants wrote a Risk Analysis Report which was submitted to the Topics instructor and to the System Development teams.

The twelve applications examined were very small systems of at most a dozen datasets, processes, and hands-on users. Nonetheless, all but one of the systems examined had at least one serious vulnerability that the original team hadn't dealt with.

BENEFITS AND PROBLEMS

For the juniors, work with the seniors made them realize that the older students really did have a better grasp of the development process. Most of the juniors come from programming classes where assignments are unambiguous, and the setting is simple. The junior project introduces an environment with complicating and contradictory

factors. The consultants' review of their analyses helped the juniors see additional subtleties in the systems they were constructing.

The use of consulting teams was beneficial in other ways. It added to the sophistication of the project and made the final product more complete. It provided an intermediate review process which improved the overall quality of the product as well as its overall security. This review was well received by the junior students. It was seen as helpful and non-threatening since it was done in the privacy of the group and since the reviewers were not grading the team product.

The teacher benefited in that the consulting teams reinforced the general methods taught in the junior course as well as supplementing the students' awareness of security concerns.

The Topics students also benefitted from the experiment. The consulting students were working in an established structure and applying their new expertise to an actual project. They had a short but realistic project. Taking on the consulting role late in the term made much of the total content of the advanced course available to them. They returned from their stints as consultants with two main conclusions:

(1) One needs more experience than can be gained in a first course on Systems Development in order to do an accurate and thorough system analysis. This Topics class provided some of that experience.

(2) Even small routine systems of no interest to criminals or hackers are subject to serious risks caused by physical calamities and human error. Most of those risks are easily controlled once they have been identified.

Some of the senior students drew secondary conclusions which dealt primarily with reactions of the users:

(3) "The users were impressed with our concern for their welfare."

(4) "The users were surprised by the questions we asked. They had assumed that we already knew about their problems and had handled them."

(5) "The users were surprised that we raised issues which were important to them, but that they had not considered before."

These conclusions strengthened the knowledge that the Topics students gained from the course. They showed its direct applicability to the goals of software development and reinforced the lessons learned from their own junior projects. Problems were minimal. Timing was the most critical problem. The consulting team arrived about 3/4 of the way into the term when the project team had completed the study of the existing system. Because it was late in the term, there was some resentment about the extra time the review required in the juniors' development schedule.

CONCLUSIONS

Interaction with consultants from the Topics course enabled System Development students to recognizethe introductory nature of their course. Association with Systems Development project teams allowed the Topics students to work with a real system while concentrating entirely on the particular subject of their course. Both the project designers and the consultants were able to expand their abilities to handle specific responsibilities within a team environment.

The fact that the learning experience was successful has led us to look for other situations in which to try this type of activity. The same benefits would accrue from consultation with students in advanced classes such as Human Factors, Database Design, Testing, or Interactive Graphics. Something similar could be done with students in courses that precede Systems Development, such as Microcomputing. For instance, when a System Development team reached the point of needing to select microcomputer hardware or software to support a system, they could describe their application to a microcomputing class, commissioning the class to produce a recommendation of suitable products. The presence of a customer adds interest to a useful microcomputing assignment. The Systems Development team would gain experience in making presentations and receive help on technical aspects of their systems design.

The experiment in crossing course boundaries seemed to provide benefits to the instructors of both courses as well as to the students. These benefits were accomplished with a minimum of problems and little special effort by the instructors.

1. Beccue, Barbara & Chrisman, Carol, "Integration of Methodology and Tools: An Approach to Teaching Systems Development", ACM SIGCSE Bulletin, Vol. 16, No. 1, February 1984, pp. 10-14.

2. Proceedings, ISECON'87, "Innovative Teaching Methods I - CIS Core": Lederer, Albert L., "A Project-Based Systems Design and Analysis Course", pp. 83-86; Barrett, Robert A. & Hockensmith, Dale K., "Project Course in Systems Analysis", pp. 87-91.

3. ACM SIGCSE Bulletin, Vol. 17, No. 1, March 1985, "Reviewed Papers: SOFTWARE ENGINEERING": Collofello, James S., "Monitoring and Evaluating Individual Team Members in a Software Engineering Course", pp. 6-8; Carver, Doris L., "Comparison of Techniques in Project-Based Courses", pp. 9-12; Bickerstaff, Douglas D., "The Evolution of a Project Oriented Course in Software Development", pp. 13-22.

4. Gallupe, R. Brent, Shonoski, Paulette L., Nelson, Ronald A., & Cameron, Robert A., "Merging I.S. Curricula with Business Core Subjects: An Integrative, Operational Level Case Project for Information Systems Majors", Proceedings, ISECON '87, pp. 9-14.

TACTICS FOR CREATING AN EXAMPLE BASED KNOWLEDGE BASE FOR THE 1ST-CLASS EXPERT SYSTEM SHELL

ANNE McCLANAHAN AND THOM LUCE
Management Information Systems
Ohio University
Athens, Ohio 45701

ABSTRACT

This paper discusses the use of 1st CLASS, an example-based expert system shell, in Expert Systems classes. Problems in structuring the knowledge base and in creating a comprehensive set of examples are discussed. A hierarchical design reduces the number of factors in each module, thus reducing the number of examples required to represent the knowledge. 1st CLASS uses induction to develop rules based upon the examples provided by the developer. Modules can be linked together for more efficient rule production. Even with hierarchical design, entering a comprehensive set of examples is tedious and subject to error. LOTUS 1-2-3 is recommended for entering examples quickly and efficiently, and for developing weights for the outcomes. These techniques are illustrated with examples from a Publishing Expert System developed to recommend signing a contract for a book or to pursue signing a coauthor or finding a new author for a potential book.

INTRODUCTION

Expert systems classes typically develop small expert systems using commercial shells on personal computers. One popular shell is 1ST-CLASS, an example-based program. This paper presents techniques for structuring the knowledge base and for streamlining the tedious job of entering examples for an expert system using the 1ST-CLASS program. Steps will be illustrated using a Publishing Decisions expert system the authors developed while students in an Expert Systems class.

AN EXAMPLE BASED EXPERT SYSTEM

1st CLASS uses an inductive process for creating rules. The developer enters examples which represent the expert's decision behavior; these are entered in spreadsheet format. Ruth (3) points out that this is a good procedure for learning about expert systems. "For students this is a very efficient learning experience, since writing examples is more easily done initially than writing rules" (1, 39). Then the system uses the examples to generate a rule structure, a hierarchical decision tree. 1st CLASS enables the user to select one of several models for the rule structure; it is also possible to customize the rules, either modifying the rule tree the system built, or constructing it on the screen (3, 45).

An expert system is created in ambiguous decision environments. If the problem is complex, many factors will be involved. An example-based expert system shell like 1ST-CLASS enables the developer to define a knowledge base as a set of factors, with a limited number of values possible for each factor. The knowledge is represented by a comprehensive set of examples; each example associates an outcome with a specific set of values of the

factors. The software develops rules inductively based on these examples.

For each factor, or characteristic, the developer enters a question to be asked of the user, who must then select one of the available values. When the user has responded to all the questions, choosing one value for each factor, the system produces a decision. If there are many values for each factor, the system can represent very fine distinctions. On the other hand, the number of examples increases dramatically with the number of alternative values per factor, as with the number of factors.

PROBLEMS

Organization of a knowledge base is similar to that of a database; it is possible to keep all the fields in one file, but it is awkward and inefficient to do so. When there are many factors, a particular response to one factor may render others moot. If all factors are in one file, the user must still answer the irrelevant questions. For example, the user of a publishing decision system may indicate that a book is intended for a new market and thus make questions about the current competition moot. The user will find answering irrelevant questions irritating and cumbersome.

For the developer, a simple structure with few possible values per factor limits the example space. There must be one example for each possible combination of values of the factors in order to produce a decision for every user session. Yet, even four factors, with 4 alternatives each, will require 4 X 4 X 4 X 4 or 256 examples. Such a limited set of alternatives maybe of little value practically; certainly the number of factors, if not the number of values per factor, will often be much larger. As the number of factors and examples grows, the number of examples quickly becomes unmanageable; but eliminating some of the possible examples yields a system that cannot function properly.

The 1ST-CLASS program requires the developer to enter a value for each factor in each example. When there are many factors, the values of all but one must be held constant while the values for the one are varied. Consider, for example, Figure 1 which shows information used to rate an established area of publication. These examples use a rating of the competition (#COMPETE), the uniqueness of a purposed approach (UNIQUE), enrollment treads in the field (ENROL), whether the field is over published (OVERPUB) and whether there have been significant changes in the field which might affect the success of a text (ENVIRON). The figure shows the complete set of examples for a unique approach when the competition is 90 but shows only the first example for a somewhat unique approach and none of the examples used for a non-unique approach (and no examples for competition levels other than 90).

It is tedious to enter many similar examples; it is also difficult to avoid missing some. It would be helpful if the system offered default values on each factor (the one selected for the last example); the developer could then choose the default for all the factors being held constant. However, 1ST-CLASS offers no means of doing this. While this example-based system is conceptually simple, implementing for a moderate size knowledge base can be tedious, time-consuming, and error prone.

SOLUTIONS

Two strategies can help eliminate the problems described above: modular (hierarchical) design and using LOTUS 1-2-3 to help generate the examples.

Hierarchical Design

Creating a hierarchical design is a major step in limiting the example space. Factors can be grouped into cohesive units, evaluated and the result passed to a higher level module. A module with four factors, each with three possible values, requires 81 (3 X 3 X 3 X 3) examples to cover all possible combinations. It may be, however, that the 81 combinations lead to

#COMPETE	UNIQUE	ENROL	OVERPUB	ENVIRON	RESULT
90	YES	GROWING	YES	YES	50
90	YES	GROWING	YES	NO	50
90	YES	GROWING	NO	YES	70
90	YES	GROWING	NO	NO	50
90	YES	STABLE	YES	YES	50
90	YES	STABLE	YES	NO	50
90	YES	STABLE	NO	YES	70
90	YES	STABLE	NO	NO	50
90	YES	DECLINING	YES	YES	50
90	YES	DECLINING	YES	NO	0
90	YES	DECLINING	NO	YES	50
90	YES	DECLINING	NO	NO	50
90	SOMEWHAT	GROWING	YES	YES	50

Figure 1.

a very small set of results, say four. Using a hierarchical design allows a module to consider only the results from its lower level modules instead of all the examples from which the results were derived.

Figure 2 Part (A) shows a top level module which calls three sub-modules, AUTHOR, MARKET, and BOOK (the # symbol indicates a sub-module). Part (B) shows the AUTHOR submodule. This routine evaluates authors by calling other modules to examine personal characteristics (#PERS) and professional qualifications (#PROF). There are 16 potential examples which combine personal and professional qualifications but they produce only four results (90, 70, 50, and 0). Examples used in the main module (part A) consider only the four results instead of the 16 different combinations used to arrive at the results.

#AUTHOR	#MARKET	#BOOK	RESULT
90	90	90	Sign
70	70	70	Sign_CA
50	50	50	Sign_New
0	0	0	Dontsign

(A)

#PERS	#PROF	RESULT
90	90	90
70	70	70
50	50	50
0	0	0

(B)

Figure 2.

The hierarchical design solves two problems: the user need not answer irrelevant questions, and the number of examples required is substantially reduced. Certain responses in one low-level module may cause the system to skip other modules at the same level. For the developer, this also has the effect of creating many of the possible examples automatically. The payoff is quite similar to that of normalizing data for a database; the information is still available, but redundancy is substantially reduced.

Lotus 1-2-3

Even with a reduced set of examples, it is still tedious to enter them. 1ST-CLASS enables the developer to export the file design and any existing examples to a LOTUS 1-2-3 compatible file. The complete set of examples can then readily be created, relying heavily on the COPY feature. This process is not only fast; it also makes it easier to avoid omitting a

few examples. Once the example set is complete, the worksheet is printed and imported into 1ST-CLASS.

An added benefit of using 1-2-3 is that it speeds the development of weights for the factors. Since an expert system is generally producing recommendations for an ambiguous decision, it is important to assign probabilities to the outcomes. Each outcome can be assigned a weight as the example is created. In 1-2-3, the developer can build formulas into the spreadsheet that will compute weights and store them in the appropriate location in the worksheet. The weights are then imported along with the examples. As testing reveals required changes in the formulas, the examples can be exported, updated with revised formulas, and then imported again (or the original Lotus 1-2-3 worksheet can be modified and the results sent back to 1ST-CLASS). This provides fast, accurate means of modifying the model to more closely approximate the expert's decision behavior.

SUMMARY

While an example-based expert-systems shell is easy to understand, it is awkward to implement. Modularization and example-development in LOTUS 1-2-3 improve design and speed development. The steps described above provide a framework for simplifying the development of an expert system using the 1ST- CLASS expert system shell.

AN ILLUSTRATED EXAMPLE

The example described here was developed by the authors while they were students in an Expert Systems class in August, 1987. The system was implemented in 1st CLASS as well as in a rule-based shell.

HIERARCHICAL STRUCTURE

The first major problem in expert system development is knowledge acquisition. If the class is intended simply to teach the key strokes required to enter a sample problem, perhaps the students can serve as their own experts. However, it is important to understand how to create a conceptual framework for the system, as well as to have the experts test the system. This problem is discussed in "Knowledge Acquisition and Expert System Development Classes: A Proposed Procedure" (2). The experts consulted for the expert system used in this illustration were the president and vice president for sales of Mitchell Publishing Company. Interviews were used to develop a list of the categories they used in making decisions about whether or not to offer a contract for a book. The categories were then grouped into sets of similar content, and the hierarchy of modules was developed.

The modules were developed using a top-down process. The first-cut system consisted of a single module containing only three factors: Author, Book and Market. Each of the factors had three possible values: Good, Fair and Poor. Twenty-seven examples (3 x 3 x 3) were created to associate each possible combination of values with an outcome. The number of alternatives was deliberately limited to restrict the number of examples required. In a more refined model, the number of values would probably be expanded to provide for more accurate decisions. Since these decisions are probabilistic, a trial weight was attached to each outcome. The weights are revised as the system is tested.

Figure 3 shows the factors, values and potential results from which the examples were drawn. The 'SIGN' result indicated that the book contract should be signed, while 'DONTSIGN' indicated the opposite. Two intermediate results, 'SIGN_CA' and 'SIGN_NA' indicated a good idea but suggested either a CoAuthor or a New Author.

AUTHOR	MARKET	BOOK	RESULT
GOOD	GOOD	GOOD	SIGN
FAIR	FAIR	FAIR	SIGN_CA
POOR	POOR	POOR	SIGN_NEW
0	0	0	DONTSIGN

Figure 3.

Next several sub-factors were identified for each of the factors in the top module. For example, the Market factor was divided into: New Market, Established Market, Risk, and Current Competition. Note that New and Established are mutually exclusive. Again, a set of examples was created with the outcomes identical to the values for Market at the higher module: Good, Fair, and Poor. Weights were also provided for each example. The outcomes for this module were then fed into the Market factor above. Using sub-modules, the user is asked questions at the lower level instead of being asked for a global evaluation of Market, as in the first prototype. New and Established each had two values (Yes and No), Risk had three (High, Medium and Low), and Current had two (Yes and No). This required 24 examples (2 x 2 x 3 x 2), each with a weight; however, since New and Established are mutually exclusive, only half of the examples were needed (see Figure 4). If the response was New, a value on Established was irrelevant, so a "don't care" (*) was associated with it.

#NEW	#ESTAB	RISK	CURRENT	RESULT
YES	*	HIGH	YES	50
YES	*	HIGH	NO	70
YES	*	MEDIUM	YES	70
YES	*	MEDIUM	NO	90
YES	*	LOW	YES	70
YES	*	LOW	NO	90
NO	*	HIGH	YES	50
NO	*	HIGH	NO	70
NO	*	MEDIUM	YES	70
NO	*	MEDIUM	NO	90
NO	*	LOW	YES	70
NO	*	LOW	NO	90

Figure 4.

The process is continues by breaking down each factor at the second level into another module. For instance, the Established factor was broken down as shown in Figure 1. Again, the responses on this module were fed into the factor above it; so the user was asked only the lower level questions. During the development process, each module was tested to see that the outcome was believable.

ENTERING EXAMPLES IN 1-2-3

A second step reduces the tedium of entering rules, though not the number of examples. 1ST-CLASS permits the user to export a set of examples to LOTUS 1-2-3 and/or to import a file from 1-2-3 into the example space. Since each rule must contain a value for each factor, creating a complete set of examples requires varying the values for one factor while the others are held constant; there are many repetitious entries. Entering a few rules in 1ST-CLASS and then exporting them to a Lotus 1-2-3 spreadsheet permits very quick entry. Once in Lotus, the COPY command can be used to duplicate rules which can then be modified as needed. The following samples show the steps required to create examples for Established Market characteristics:

#COMPETE	UNIQUE	ENROL	OVERPUB	ENVIRON
90	YES	GROWING	YES	YES

Duplicate this row, then change the second "YES" to "NO."

#COMPETE	UNIQUE	ENROL	OVERPUB	ENVIRON
90	YES	GROWING	YES	YES
90	YES	GROWING	YES	NO

Now duplicate these two rows, and change the OVERPUB value to NO for the bottom two rows.

#COMPETE	UNIQUE	ENROL	OVERPUB	ENVIRON
90	YES	GROWING	YES	YES
90	YES	GROWING	YES	NO
90	YES	GROWING	NO	YES
90	YES	GROWING	NO	NO

This process can be continued, so that many examples can be created very quickly. When the example set is completed, it is saved as a PRN file which can be imported into 1ST-CLASS.

After all the sets of values for the examples are completed, the outcomes are entered. In this project, the number of outcomes was restricted; as the system is refined through use, the number of

possible outcomes can be increased. Note that this module feeds the outcome into a factor at the next higher level; these outcomes must match the possible values in the module above. Figure 2 illustrates this requirement with the results of a sub-module matching the possible values found in the parent module.

WEIGHTING THE OUTCOMES

In addition to including all the required examples, it is easy to compute weights using a spreadsheet. Formulascan be stored on one part of the spreadsheet; the computed weights are stored in the appropriate part of the file for transfer to 1ST-CLASS. It is easy to modify the weights, either working with the experts or in response to testing the model. Because tests are performed on character (label) data, Lotus 1-2-3 version 2.0 or higher is required.

Figure 5 shows the formulas used to covert character answers into numeric answers for further processing. The first formula, found in cell I1, assigns a numeric value of zero to a competition score of "90". Competition of "70" is assigned a 5, "50" is converted to 7 and anything else is given a value of 9.

```
I1:@IF(A1 = "90",0,@IF(A1 = "70",5,@IF(A1 = "50",7,9)))

J1:  @IF(B1 = "yes",9,@IF(B1 = "somewhat",6,3))

K1:  @IF(C1 = "growing",9,@IF(C1 = "stable",6,3))

L1:  @IF(D1 = "yes",3,9)

M1: @IF(E1 = "yes",9,3)

N1:  +I1*0.2 + J1*0.2 + K1*0.2 + L1*0.2 + M1*0.2

P1:  @IF( + N1 > 8,90,@IF(N1 > = 6,70,@IF(N1 > 4,50,0)
```

Figure 5.

The formula in cell J1 converts an uniqueness value of "YES" to 9, "SOMEWHAT" becomes 6 and any other answer is considered a 3. The formula in K1 does the same thing for enrolment values of "GROWING", "STABLE" and "DECLINING". "YES" and "NO" are converted into 3 and 9 by the over-publishing formula in L1 and to 9 and 3 by the environmental formula in cell M1.

The individual values are weighted and summed by the formula in cell N1. This particular example assigns equal weight (20%) to each of the five factors. Should testing or additional work with the experts indicate that the weights are inappropriate, the formula can be changed and copied into all appropriate cells. The final result is calculated by converting weighted scores in the range of 8 - 10 into 90, scores between 6 and 8 become 70, those from 4 to 6 are converted to 50 and anything below 4 becomes zero. The final result then, is one of the four scores which can be returned from the sub-module.

The complete hierarchical arrangement of factors is shown in Figure 6. The number of values associated with each factor is indicated after each factor. Note that 1ST-CLASS permits subdivision in many combinations. In the market segment, too many characteristics were necessary for evaluating new and established markets, so they were put into separate modules. However, since current and risk seemed adequately covered by more global ratings, they were not partitioned. In the example above, reviews (under BOOK) seems a candidate for subdivision. This procedure works well for iterative development.

The techniques described above permit the user to concentrate on the concepts and the knowledge structure, rather than becoming lost in the complexity of using the expert system shell as a tool. This process can be used as a framework for developing Expert Systems classes.

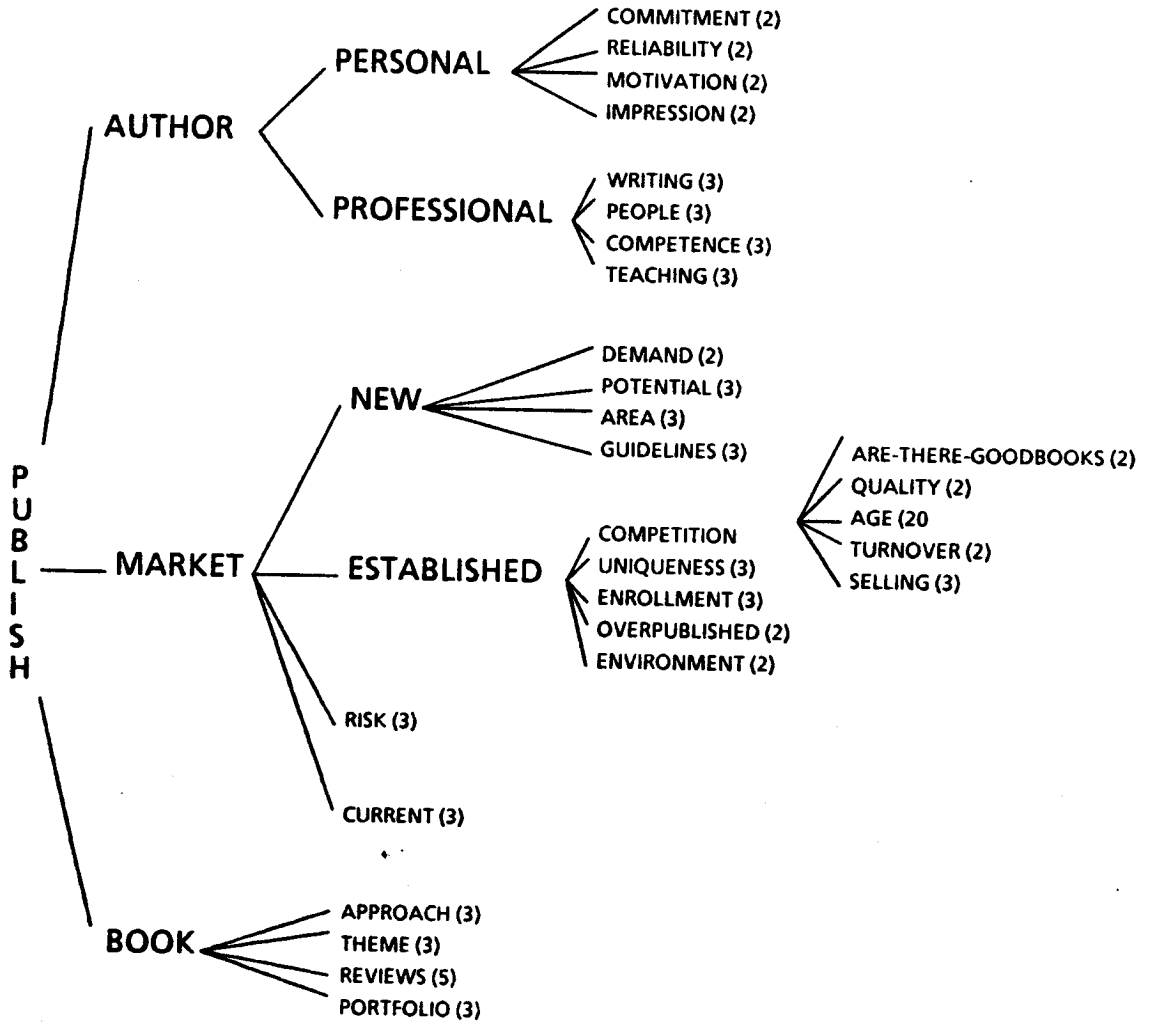


Figure 6.

REFERENCES

1. Gevarter, William B. "The Nature and Evaluation of Commercial Expert System Building Tools. COMPUTER, May 1984, pp 24-41.

2. Luce, Thom and McClanahan, Anne. "Knowledge Acquisition and Expert System Development Classes: A Proposed Procedure." Proceedings of the North American Conference of the International Business Schools Computer Users Group, July, 1988.

3. Ruth, Stephen R. "Introducing AI and Expert Systems In the Business School", Interface, 9(4) Winter 1987-88, pp 42-48.

AN INNOVATIVE TEACHING METHOD USING A THEORY-PRACTICE-LEARNING PROCESS TO INTRODUCE EXPERT SYSTEMS

Yvonne Lederer-Antonucci
Widener University

ABSTRACT

By using an Expert System Shell as a development tool, students are able to reinforce theory with practice. This supports an iterative teaching approach of theory - practice - learning. This approach begins with presenting the theory of expert systems followed by an assignment to allow practice, then analyzing the assignment which may lead to new theory and additional practice. The students complete an expert system development life cycle by implementing a course advisory expert system.

INTRODUCTION

Artificial Intelligence (AI) is on the verge of moving from the research lab into business applications in the form of Expert Systems. These Expert Systems are still far from the envisioned Artificial Intelligent systems, however the basic reasoning theory of AI has emerged in the form of Expert Systems designed to assist decision making(1). As educators we must keep our students abreast of these technologies and theories which may become a part of their corporate life.

TEACHING PHILOSOPHY

The most effective method of introducing this theory is to reinforce as much as possible with practice. This has been accomplished at Widener University through development of an advising Expert System for the School of Management using an Expert System Shell, and various innovative assignments. At the completion of each project phase and / or assignment, the "learning" is "pulled" out of the experience. This is an iterative learning process of theory - practice - learning for each topic in this 14 week course. In addition, the students have enjoyed and have taken pride in this project due to the actual need and use of the advisory system by students and faculty alike.

COURSE CONTENT

The order of the course content plays an active role in this iterative learning process. The topics begin with a general introduction to AI and Expert Systems.

(1) Introduction to AI
(2) Expert System Applications
(3) VPExpert Demonstration
(4) DSS, MIS, Databases, vs. Expert Systems
(5) Knowledge Representation
(6) Rule Based Systems
(7) Overview of VPExpert
(8) Fuzzy Variables
(9) Inference Engine
(10) Dealing with Uncertainty
(11) User Interfaces
(12) Knowledge Engineering and Planning for Expert Systems
(13) Developing a Knowledge Base
(14) Reasoning

(15) Confidence Factors
(16) Menus
(17) Knowledge Based Organizations
(18) Other Development Tools - PROLOG, LISP, LEVEL5, GURU

The first quarter of the material is designed to give the student an idea of what Expert Systems can accomplish. The second quarter introduces the student to development techniques and theory. At this point the student has a good grasp for the underlining theory of Expert System development. The next step is to introduce the planning cycle. The Expert System Development Life Cycle (ESCLC) adopted for this course is from the book by Waterman[2]. The remainder of the course was used to develop the advisory system which required the students to learn user interface and knowledge representation techniques. The students were exposed to all components of the Expert System (user interface, knowledge base, and Inference Engine) by playing the role of the Knowledge Engineer.

During the presentation of the first quarter material it is very important to present the major differences between an Expert System and AI. The student must realiz e that true AI applications are unlikely to happen, however AI like problems which employ routine diagnostic decisions can be simulated to assist the decision maker. It is also necessary to present example applications to give the student a "feel" for how Expert Systems are being used.

The first homework assignment occurs at this point, requiring the student to think of an application and write ten rules. These applications have ranged from advise for setting a camera shutter speed to training for a word processor. The student should attempt to create goals and sub-goals in their assignment. This is their first attempt of knowledge representation.

KNOWLEDGE REPRESENTATION

The student is probably most familiar with rule-based knowledge representation due to the student's exposure in previous programming courses. Therefore, it is very important to compare and explain other knowledge representation techniques such as frame-based and semantic networks. Related homework assignments should follow each discussion to give the student practice at the various knowledge representation methods. It is extremely useful if the student is asked to convert homework #1 into a frame-base rather than a rule-base and write about their difficulties. This makes the student aware of the differences between these methods.

At this time, I chose to introduce the Expert System Shell the class would be using as a development tool, VPExpert, and explained how to create a knowledge base using this shell. Of course, to reinforce learning with practice, the next assignment required the students to create a knowledge base in VPExpert using their ten rules from homework #1. As the class found from this assignment, the conclusions of their examples may have been wrong because they did not consider the reasoning method of VPExpert's Inference Engine.

INFERENCE ENGINE

The next lecture included the various reasoning methods of Inference Engines. The theory included explanations of (1) Backward Chaining, (2) Forward Chaining, (3) Breadth First, and (4) Depth First. Next, the

practical combinations were explained including (1) Backward Chaining, Breadth First, (2) Backward Chaining, Depth First, (3) Forward Chaining, Breadth First, and (4) Forward Chaining, Depth First. Once the students understood these concepts, they were able to infer that VPExpert has a Backward Chaining, Depth First Inference Engine. They then could write the knowledge base for this type of reasoning, and understood the importance of the Inference Engine.

Additional features of the Inference Engine were introduced next. The most important feature is the certainty factor (also known as the confidence factor) which is applied to the condition or conclusion in a knowledge base. Fuzzy variables were also introduced, which were combined with certainty factors. The students then added these features to the VPExpert knowledge base to reinforce the theory.

The students now had the practice of developing a simple knowledge base using Inference Engine features, however it was missing an important component of an Expert System known as the user interface.

USER INTERFACE

A lecture on various user interfaces proceeded to include explanation and comparison of various methods such as (1) command-oriented, (2) menu-guided, (3) natural language, (4) object/action, and (5) customized combinations.

The VPExpert shell is limited to a menu-guided and / or command-oriented user interface. For this reason, the students were asked to describe the interface which would be most appropriate for their system and why. Then, the student added the menu and command interfaces to their VPExpert subsystems and attached them using a main menu.

By this time the students had learned the Expert System theory and reinforced that theory by practice for each Expert System component: The Knowledge Base, The Inference Engine, and The User Interface. This section of the course took approximately five weeks to complete. The students were now ready to begin the Advising project, beginning with the identification and problem definition phase of the ESDLC.

KNOWLEDGE ENGINEERING AND PLANNING

Prior to knowledge engineering, the objectives of the Expert System had to be identified:

(1) Need Must Be Identified
  * The course advising procedure is a problem-solving activity where resident human expertise (Professors) are in short supply, overburdened, and unavailable when needed.

(2) Identify the Purpose
  * The course advising system will help a student choose courses to take in their major based on their interests and career goals.

(3) Identify Who Will Use It
  * Students in their sophomore, junior, and senior years and faculty members not familiar with specialty areas for advising tips.

(4) Identify When It Will Be Used
  * Primarily used during pre-registration each semester.

(5) What is the Expected Benefit
  * To consolidate various Expert knowledge, formalize rules which the catalog includes and make this available to all

students and faculty.

The domain of the problem was then narrowed to only include major and minor areas which included (1) Economics, (2) General Management, (3) Management Information Systems and Computer Science, (4) Human Resources, (5) Marketing, and (6) Finance. The class was divided into groups, each chose an area to develop. This was done due to the time constraint for development. Each group then began the ESDLC by acquiring knowledge from experts.

ACQUIRING KNOWLEDGE FROM EXPERTS

A lecture to explain knowledge acquisition methods followed the problem definition. The traditional approach requires the Knowledge Engineer to resent data, problems, and questions to the Domain Expert in order to acquire knowledge, concepts, and solutions. This knowledge is then formalized and structured by the Knowledge Engineer to establish a knowledge base. This traditional process involves methods such as (1) Interviews, (2) Observations, (3) Problem Discussion, (4) Problem Description, and (5) Problem Analysis.

These five methods assist in acquiring knowledge from a human expert, however expertise can also be acquired from written material such as books, forms, or policies.

Since the course advisory system was based on policies for each major, the class began by analyzing the current catalog and other pamphlets which explained the possible course choices for each major. Each group then used a combination of acquisition methods to acquire knowledge from various faculty members. This procedure followed the first three phases of Waterman's ESDLC. The phases are:

(1) Identification
(2) Conceptualization
(3) Formalization
(4) Implementation
(5) Testing

During implementation the students presented rules to more than one domain expert (Professor) to acquire alternatives and discover discrepancies. The students did find some errors in the catalog which were presented to the Dean for correction. The alternatives presented by the domain experts also revealed an additional purpose and use of the advisory system, which is to help students choose a major based on personal interests. These types of discoveries are common at this point in Expert System development, which displays another advantage of Expert Systems.

Once these discrepancies were solved and alternatives added, user interfaces were developed. An example is shown in Figure 1. Using the capabilities of VPExpert the students were able to incorporate color and windows.

The last phase involved testing and documentation development. The class was divided into two groups to accomplish these tasks.

WHY USE AN EXPERT SYSTEM SHELL?

During the last two weeks of class, time permitted additional topics to be presented. Other development tools were demonstrated such as PROLOG, LISP, LEVEL5, and GURU. The class realized if the project had been developed in an Artificial Intelligence language such as PROLOG or LISP, they would still be in phase one of the life cycle. This is one reason a shell was chosen as the development tool. Another reason was the high amount of training necessary to learn PROLOG or LISP which time

did not permit. The purpose of the course was not to learn a new language, but to be introduced to Expert Systems and AI, and acquire an understanding of the development process of an Expert System.

For these reasons, an Expert System Shell was a logical choice, however many have asked why I chose VPExpert. I had done a market search for a shell which was affordable, and contained enough versatility to develop minor systems. Most of the shells evaluated were rule-based systems which is an advantage because my students were most familiar with rules from other courses using procedure - oriented languages. VPExpert is microcomputer based, and Paperback Software offered an educational version for $19.95 which included the manual. Considering the price, VPExpert is a very powerful shell, and easy to use.

CONCLUSION

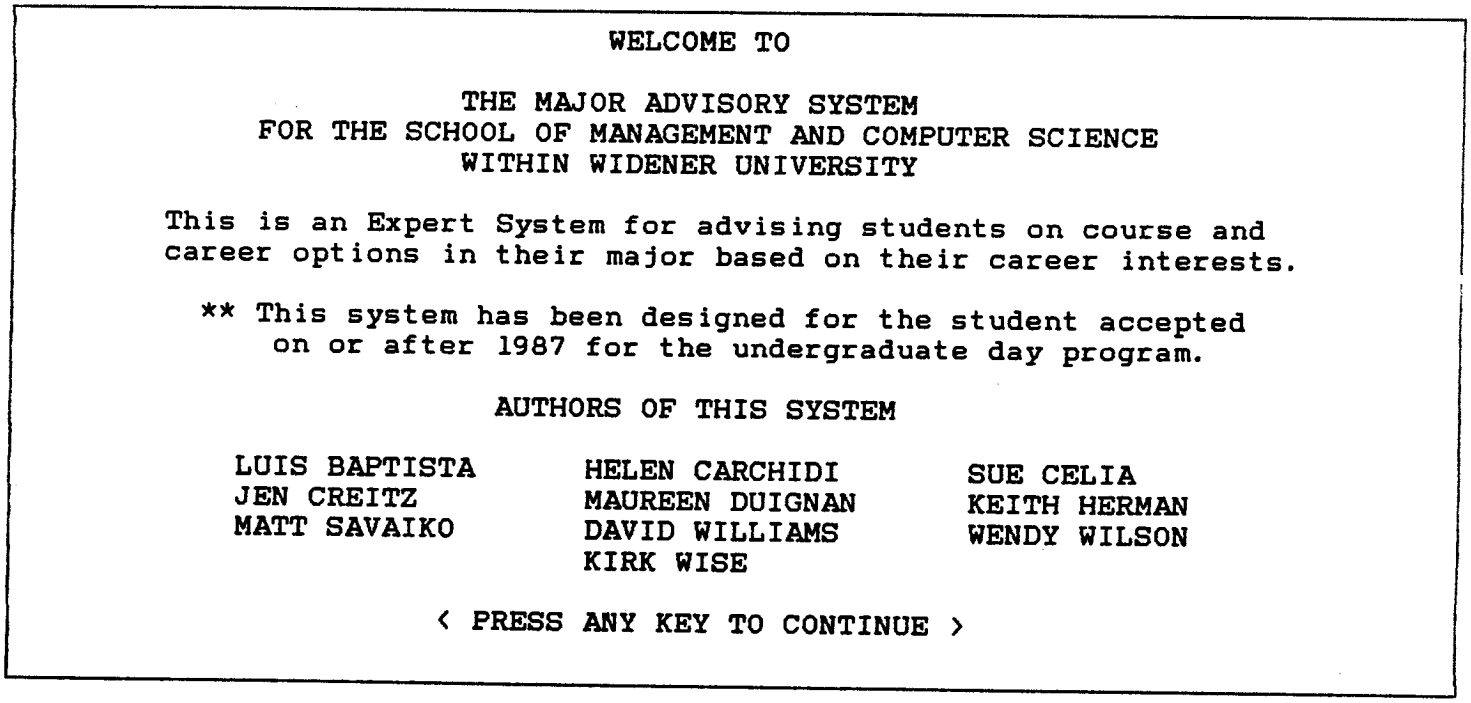
By using an Expert System Shell as a development tool, I have been able to reinforce theory with practice. The students have been able to experience an Expert System Development Life Cycle by planning, developing, and implementing an advisory system for the School of Management at Widener University. This process of theory-practice-learning was successful. The project was completed within the 14 weeks and implemented into the student micro-labs on campus. This project gained university recognition through university publications and is presently being considered as a prototype for a future LISP based university wide advising system.

REFERENCES

(1) Sheil, Beau. "Thinking About Artificial Intelligence". Harvard Business Review. July-Aug 1987. pp. 91-97.

(2) Waterman, D.A. A Guide to Expert Systems. Addison-Wesley Publishing Company. 1986. pp 135-141.

FIGURE 1


WELCOME TO
THE MAJOR ADVISORY SYSTEM
FOR THE SCHOOL OF MANAGEMENT AND COMPUTER SCIENCE
WITHIN WIDENER UNIVERSITY
This is an Expert System for advising students on course and career options in their major based on their career interests.
** This system has been designed for the student accepted on or after 1987 for the undergraduate day program.
AUTHORS OF THIS SYSTEM
LUIS BAPTISTA
HELEN CARCHIDI
SUE CELIA
JEN CREITZ
MAUREEN DUIGNAN
KEITH HERMAN
MATT SAVAIKO
DAVID WILLIAMS
WENDY WILSON
KIRK WISE
< PRESS ANY KEY TO CONTINUE >

APPLYING STRUCTURED WALKTHROUGH TO DSS DEVELOPMENT

George W. Morgan
Department of Computer Information Systems
Southwest Texas State University

R. Wayne Headrick
Department of Business Analysis and Research
Texas A&M University

ABSTRACT

With recent advances in hardware and software technologies, the development of automated systems specifically aimed at supporting the decision making processes within an organization (generically known as decision support systems) has often become a dynamic, user-driven process. This paper cautions against total abandonment of traditional design techniques when adopting new techniques for designing such systems, and suggests that the use of the structured walkthrough in conjunction with newer iterative design techniques typically used to develop a decision support system (DSS) is vital to successful, timely development of such a system.

INTRODUCTION

Problem solving and decision making are central to the function of management, with managers at all levels of an organization encountering both routine and non-routine problems on a daily basis. The success of the organization depends on those problems being resolved as accurately, quickly and consistently as possible. The quality of the decisions resolving such problems is directly related to the information available to the decision maker at the time of resolution.

An IBM survey has indicated that the information processing centers of most well-managed corporations have a three to four year backlog of new applications development requests, with such requests increasing by as much as 45 percent per year. To compound the problem, conventional structured systems design techniques tend to be quite time consuming, and were originally developed to deal with well structured procedures solving well structured problems. Because a decision maker's informational needs tend to revolve around unstructured or semi-structured problems, and often relate to crisis situations that demand fast resolution under stressful conditions [5], neither conventional information systems nor the structured techniques used in their development are capable of adequately meeting those needs.

The concept of DSS was developed to provide a framework for using the data contained in an information system to give computerized decision making support to management. The focus of DSS is on assisting managers in making decisions related to semi-structured problems, supporting managerial judgment with the goal of improving the effectiveness of the decision making process [7]. As the situations in which a DSS application might be created to aid in making a decision regarding a problem are relatively unique, user requirements for such applications have been difficult, if not impossible, to pre-specify. In addition, even after a DSS application has been created, it will often require dynamic user-driven modifications. Only recently have technological advancements in interactive tools such as prototyping became available to support interactive system development methodologies [3,6] so that the creation of DSS applications has generally become feasible.

DSS DEVELOPMENT

Developing a DSS for use by a decision maker in solving even just a single problem, or class of problems, typically requires many modifications during the development process. It is in this kind of environment that iterative systems development methodologies become particularly useful. Of course, such iterative methodologies have been able to become successful because of recently advanced techniques such as prototyping, supported by 4th generation languages and applications generators. Prototyping has become especially popular because it can get an initial simulation of a proposed DSS to the user quickly, making it possible for the user to see and review actual working functions rather than simply reviewing written specifications. This process also has the desirable effect of actively involving the DSS user in the development effort. [1,2]

As noted above, the benefits associated with using an iterative methodology such as prototyping in the development of a DSS are readily apparent. The costs, however, are not quite as easily recognized. Prototyping requires that the developer gain only an initial understanding of the problem, or needs of the user, before immediately going about the task of implementing that understanding. Such an approach to systems development leads inevitably to subsequent corrections, enhancements and additions to that initial system. In fact, it has been suggested [2] that prototyping creates the illusion that change is cheap and that the user has been issued an unrestricted license to change his/her mind. Prototyping is a very valuable tool for enabling DSS users to be actively involved in the system development process, but its cost in hardware, software and participant resources consumption must not be underestimated. It must be recognized that developing a reliable DSS at a reasonable cost still requires the application of structured design techniques to the various stages of the development life cycle.

DSS DESIGN

The iterative system design methodology known as prototyping is, as noted above, quite well suited to the task of developing a DSS. Because prototyping actually involves the user

of the DSS in the development process, it helps avoid one of the major problems that can result when a more traditional systems design methodology is used - that of building the wrong system [3,6]. At the same time, as the user becomes more and more involved in the design prototyping process, there may be a tendency for that user to make system changes without adequately reviewing the consequences and/or benefits of those changes.

Prototyping using 4th generation languages and application generators has been shown to be capable of significantly increasing the productivity of system developers. If such productivity increases are spent simply reducing the time it takes to develop an ill-defined DSS that doesn't serve the user's needs, they are obviously wasted. If, on the other hand, they are realized when developing a well designed DSS, the whole organization benefits.

Prototyping was developed as a means for overcoming the long lead times associated with more traditional structured design methodologies. Basically it relies on the system designer to somehow build a DSS prototype that is consistent with the needs of the user, with the user then reviewing the system to identify necessary corrections and enhancements. It is obvious that without the inclusion of at least some elements of structured design to the process, prototyping would lead to a system that is likely to have many errors introduced by both the designer and the user, with a significant number of them never being found.

WALKTHROUGH

A number of studies [3,4] have shown that even organizations that apply structured design methodologies to their systems development efforts experience errors. These studies have also shown that while as much as 60% to 80% of all developmental errors occur during system specification and design, the cost of uncovering and correcting those errors is lowest at that point. Using the cost of correcting errors detected at this stage of a system's development as a base line, the cost of detecting and correcting errors after the design has been completed, but before testing has commenced, will typically be 4 to 5 times higher; during testing, 10 times higher; and after release to the user, up to 40 times higher. . It is easy to see that errors should be detected and corrected as early as possible in the development life cycle.

To accommodate the need to detect and correct errors early, the walkthrough was made an integral part of structured systems design methodologies from their beginning. A structured walkthrough generally consists of a step-by-step group review of a system that can take place at any point in the system development life cycle. This means one can take place as early as specification development and as late as final acceptance testing. As the purpose of a walkthrough is to identify errors and omissions, so they can be corrected or resolved by the system designer, it fits quite well into the role of providing a structured review mechanism for use in conjunction with prototyping.

SUMMARY

It has been noted that the successful use of prototyping in the development of a DSS is particularly dependent on the correctness and completeness of the original design. A faulty design will almost certainly result in a significant increase in the number of

iterations required to move from an initial prototype to the final DSS. In fact, the addition of ill-defined user reviews to this scenario would doom the DSS to failure. Incorporation of structured walkthroughs into every phase of the prototyping effort will provide for the effective review and feedback necessary to ensure that the resultant DSS is as accurate and complete as possible.

REFERENCES

[1] Alexander, DS. "Planning and Building a DSS." Datamation. 15 March 1986, 115-21.

[2] Boar, BH. Application Prototyping: A Project Management Perspective. AMA Membership Publications Div., New York, 1985.

[3] Martin, J. Application Development Without Programmers. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[4] Pressman, RS. Software Engineering: A Practitioner's Approach. McGraw-Hill Book Company, New York, 1982.

[5] Schermerhorn, JR. Management for Productivity, 2ed. John Wiley & Sons, New York, 1986.

[6] Tozer, J. "Making The Systems Match The users". Management Today. October 1985, 41-8.

[7] Whyte, RG. "What Is A Decision Support System". Industrial Management and Data Systems. July/August 1986, 28-30.

DYNAMICS OF A SYSTEMS ANALYSIS AND DESIGN COURSE

S.K. SHAH
DIANE FISCHER
HOFSTRA UNIVERSITY

ABSTRACT

The authors examine a project-oriented course for systems analysis and design that has been successfully offered for several semesters. The approach integrates state-of-the-art software and ethical "realities" to enhance the project orientation and maintain the vitality and relevance of the course to current and future business concerns.

INTRODUCTION

A systems course is designed to familiarize the student with various activities associated with the tasks of analysis and design in an organizational setting. Students come to the course with knowledge of a programming language and a measure of computer literacy. The project orientation of the course enables students to leave with knowledge of the concepts and theory of analysis strengthened by first-hand experience in applying these to business problems.

This approach yields a course vital to the students' familiarity with and appreciation of information systems. However, as a mirror of a well-designed system, the course is carefully monitored and state-of-the-art elements periodically incorporated to reflect the currency of the course while reinforcing the utility of IS. The addition of a section on ethics serves to heighten student awareness of the importance and place of ethical reasoning in business and suggest techniques for the analysis and resolution of emerging moral concerns.

RATIONALE

A systems analysis course entails compartmentalizing the reality into meaningful components and integrating them in an environment in accordance with the specified needs of the organizational whole. Efforts to uncover causal relationships among operations require that a systems analyst understand the "reality" of the situation. Reality manifests a mix of rational and intuitive, fractional and holistic, amoral and ethical modes of behavior. The course provides the student-analyst an opportunity to be exposed to the reality and reflect on various types of situations encountered.

The course has been reinforced by the inclusion of two components:

(1) the introduction of an integrated workbench for system applications; and
(2) an overview of the concepts of ethical reasoning and its significance to the analyst.

SOFTWARE TOOLS FOR PROJECT MANAGEMENT

While the tasks involved in the term project focus on information systems as a goal, state-of-the-art technology is not fully utilized. In the past, students have used a few productivity packages to produce final reports. Other tasks such as duration and calendar time planning, work-flow and work-load analyses, data flow diagrams, questionnaire analysis, cost/benefit evaluations, design of input and output forms, etc., were done manually. Incorporation of new modules in the course has remedied this situation.

Both the student and today's managers have available a large array of software to help in the arduous task of tracking and controlling projects. Some of these packages provide numerous capabilities such as:

(1) project data summary: expenditure, timing and activity listing
(2) data manipulation, management and reporting capabilities
(3) critical path analysis
(4) what-if impact analysis
(5) early warning systems
(6) resource planning and analysis
(7) cost and variance analysis.

Index Technology Corporation's Excelerator is a comprehensive workbench package designed to assist a systems professional. Among the major features of the package are:

(1) Graphics: Offers both analytic and presentation capabilities in addition to flow charting, data flow diagrams, structure charts, etc.
(2) Data Dictionary: Helps create logical design of a system and reports on its contents. Also creates and updates entries in the dictionary.
(3) Screens and Reports: Provides a fast and easy method for developing prototypes of the screens to permit end-user input for immediate incorporation.
(4) Analysis Tools: Assures logical consistency, accuracy, and completeness of a design. This is very helpful on complex projects. Also, it permits cross-checking and correcting errors subsequently in the design process.
(5) Dictionary Interface: Allows team members to share portions of the same design safely and securely.
(6) Documentation: Reproduces the entire documentation periodically input throughout the project. All the work ranging from graphs to narrative text is brought together into one finished form for presentation to the class and the corporate management.
(7) Housekeeping: Provides project and system maintenance for backup and security.

Our experience, albeit somewhat limited, suggests that Excelerator has potential to minimize workload, reduce amount of time and volume of details needed to develop an in-depth and accurate set of specifications for a systems project.

THE ANALYST IN A MORAL WORLD

Any reader of current business publications quickly gains the sense that scandals are shaking the foundations of our business world. The latest news of insider trading is only the more sensational example in a field where stories of computer breakins, software piracy, use of communications networks to defraud, false or unauthorized advertising and liabilities for system faults have become common occurrences. Alongside this increasing awareness of the deficiencies in the practice of business ethics, comes a resurging recognition of the need for businessmen and women to have some liberal arts training. The liberal arts are viewed as a source of the culture and values lacking in today's business environment.

Once we allow for the need for an ethical component in business education, we must decide upon a method for achieving this goal. There are two possibilities:

(1) require all business students to take an ethics course
(2) require each business discipline to provide an ethical component within a suitable course.

This is not the appropriate forum for a lengthy debate concerning the better technique. However, it should be noted that the placement of ethical considerations within a business context will strengthen student awareness of the relevance of ethics to business in ways in which an isolated ethics course cannot.

In the discipline of business computing, it can be argued that analysis and design is particularly well-suited for the inclusion of an ethics component. It is a required course for majors; all students must take it. Further, within the context of business computing, the systems profession is uniquely dependent upon public confidence. When an analyst is hired, he is expected to bring an array of technical skills to the problem of computerization of a business system. He must submit a plan for computerization, including schedules, hardware recommendations, programs and/or software recommendations. Management must have confidence in the analyst's honesty and integrity, as well as in his competence and reliability.

A component of ethical reasoning in an analysis and design course is not proposed as a panacea; students are not guaranteed to become 'good guys' upon taking the course. But it will heighten their awareness of the importance of ethics vis-a-vis business, so that in the future, when students confront such potentially conflicting situations, they will be especially cautious.

Because most business majors have no prior course experience with the study of ethics, some background must be presented. The concepts of rights and obligations furnish a basis for a discussion of ethical theories that have remained current in our society. These include utilitarianism which justifies actions in terms of their usefulness, libertarianism which justifies actions in terms of respect for the rights of others, and pluralism which judges actions in light of duties. The influence of these prevalent moral theories is identified in our commonly accepted moral beliefs.

After this overview of ethical reasoning is given, students are presented with a series of cases for analysis. These cases reflect the types of problems which the analyst may face in the course of analysis and design. An analytic model is provided. For example, the debate over the moral acceptability of euthanasia involves considerations of social utility, the rights of the living and the dying, and duties we may have to alleviate suffering and not to kill. Guide posts are set up in the form of questions to answer:

(1) what, if any, are the ethical considerations?
(2) what assumptions are made?
(3) what values are involved?
(4) from a utilitarian view, what is best?
(5) from a view of fairness, what is best?
(6) from a view of duties, what is best?
(7) overall, what takes precedence?

An initial case concerns Consumer Reports and its "no commercialization" policy whereby the magazine accepts no advertising and purchases test products on the open market. To preserve their impartiality and reputation, neither ratings nor reports may be used in advertising. In October 1985, the magazine published a study of personal

computers. Leading Edge, an IBM compatible, was among those reviewed. Leading Edge used some statements from the report in their advertisements. Consumer Report is suing them.

The students analyze this case in light of the questions posed, and decide if this information should have any effect on future decisions to recommend Leading Edge. Initially, most students head straight for the bottom line: if Leading Edge is in fact one of the better compatibles, and is competitive in price, many would recommend it. The class considers arguments concerning the reputations of both Consumer Reports and Leading Edge. Often, students suggest that any recommendation for Leading Edge should be accompanied by information on the lawsuit. In this way, a decision is passed to management.

Other cases that have been analyzed include:

* accepting stolen hardware
* copying software
* using the office computer system after hours for private consulting
* displacing workers by computerization.

Students are invited to suggest other cases of personal interest.

This ethical component of a project oriented course has been successfully implemented on a trial basis for several semesters. Student participation is high and discussions have been lively. On the basis of these discussions, students occasionally incorporate ethical considerations into their project write-ups.

In light of the resurging recognition of the place of ethics in business education, the inclusion of such a component is well-taken. Further it can be argued that analysis and design is well-suited for such a component, more so than any other course in the major of business computing. Other courses are programming classes, database systems, simulation and hardware/software selection. These tend to be very technical while analysis and design is a blend of the technical and the evaluative.

CONCLUSION

We have proposed taking a strong project-oriented course and augmenting it with packaged software and discussions of business ethics. The software component would directly enhance the projects by offering students state-of-the-art tools for project management. The ethical component enhances the relevance of the course and gives students a place to reflect on the activities of an analyst from a moral point of view. These components should serve both to strengthen and round out this course on systems analysis and design.

REFERENCES

Eliason, A.L., Systems Development: Analysis, Design and Implementation, Boston: Little Brown and Co., 1987.

Goodpaster, K., Ethics in Management, Boston: Division of Research, Harvard Business School, 1984.

Matthews, J.B., Goodpaster, K., and Nash, L.L., Policies and Persons: A Casebook in Business Ethics, New York: McGraw-Hill, 1985.

Senn, J.A., Analysis and Design of Information Systems, New York: McGraw-Hill, 1984.

Shah, S.K., "Systems Analysis: A Project Approach with Visible Payoffs," Proceedings of the Sixth Annual Information Systems Education Conference, San Francisco, 1987, pp. 185-189.

TOWARD A LABORATORY SCIENCE APPROACH FOR TEACHING THE FIRST COURSE IN COMPUTING

Donald R. Chand
Department of Computer Information Systems
Bentley College
Beaver and Forest Streets
Waltham, MA 02254

ABSTRACT

This paper describes the need, the philosophy, and our initial attempt of supporting the first course in computing with a structured laboratory that is in principle similar to the labs used in the natural sciences. It addresses issues from the perspective of the Department Chair who conceptualized this approach and stimulated at least one faculty member who spearheaded the task of operationalizing it. Specifically, this paper sketches the background that led to the idea of teaching our first course in computing as a lab science, outlines the design philosophy of the lab manual, and describes how this course was implemented in the fall semester, 1987.

INTRODUCTION

Bentley College is an institution of higher learning that specializes in business education. It offers graduate and undergraduate programs in all the functional areas of business. With an enrollment of approximately 4000 full-time day undergraduate students, Bentley College is the eighth largest undergraduate college of business in the United States. The uniqueness of Bentley's education lies in its objective of producing graduates that are immediately productive at their first job and also possess the background to grow and progress in their careers. This objective influences the curriculum, the pedagogy, and the choice of support materials like textbooks and software. For example, at Bentley we use Lotus 1-2-3 or its clone to teach the principles of spreadsheets because Lotus 1-2-3 is the spreadsheet package that our students are likely to encounter upon graduation.

Three years ago Bentley College made the decision to require its full-time day freshmen class to acquire personal computers. This decision was based on the conviction that fluency in the use of microcomputers will provide its graduates an edge. It is important to observe that Bentley chose to use the term computer fluency rather than the commonly used term computer literacy. **The term computer literacy implies a general understanding of computing and its role in society, whereas computer fluency suggests that the student gets beyond the cerebral understanding of computers to a reflex response to problem solving using a computer.** At Bentley we associate computer literacy with instructor directed computer use, and view computer fluency as student directed computer use. In other words, the goal at Bentley is to provide its students the knowledge and expertise to use personal computers as a productivity and problem solving tool to support their school work as needed. This use of the computer is envisioned to be independent of any specific computer requirement in the course.

To achieve the above goal of computer fluency, it was decided that at least fifty percent of freshmen courses and almost eighty percent of sophomore level courses must make explicit use of personal computers. The assumption here is that by the time these students become juniors and seniors they would be computer fluent and they will use the computer on their own to support their learning and homework assignments. Explicit use of the computer as a tool in the freshmen and sophomore courses requires that the students be provided the fundamental operational expertise in computing early in their college careers. It was decided that this expertise will be provided in the first computing course that all freshmen must take in their first semester at Bentley College. This course is designated CS 101 and is called Principles of Computing.

This paper describes the motivation, the philosophy and our early attempts at implementing CS 101 as a laboratory science course. It begins with the specification of the objectives and contents of CS 101, follows with the description of its implementation as a laboratory course, and concludes with a summary that captures our experience thus far.

OBJECTIVES AND CONTENTS OF CS101

The primary objective of CS 101 is the traditional first course objective of providing an understanding of the principles of computing using a hands-on approach. The secondary objective of CS 101 is to provide the operational and problem solving skills needed to use the microcomputer as a learning and productivity tool in other courses in the curriculum.

Traditionally the hands-on portion of a first course on computing involved the act of programming. In CS 101, the traditional programming approach to explore the principles of computing is replaced by microcomputer productivity software. We have discovered that a working knowledge of a standard microcomputer operating system, of a comprehensive word processing package, of an integrated spreadsheet software and of a general purpose file management system on a microcomputer effectively addresses the goal of the hands-on portion of CS 101.

The computing topics covered in CS 101 are:

Functional model of the computer
Architecture of a Computer System
Input and Output devices
System Software
Operating System Functions
Programming Languages
Database Management Systems
Data Communications
Career Paths and Computer Industry

Microcomputer Software used are:
MS-DOS
Wordperfect
Joe Spreadsheet (an affordable clone of Lotus 1-2-3)
dBase III (Student version)

IMPLEMENTATION OF CS 101 AS A LABORATORY COURSE

Motivation:
The primary objective of CS 101 is to provide the foundations for the students to acquire computer fluency. As pointed out earlier this requires that students get beyond the cerebral understanding of computing to a reflex response to problem solving using a computer. Lab science teaches technique and mens-et-manus conditioning.

The operational goal of CS 101 is to expose the students to the principles of computing and provide the operational and problem solving skills to use the personal computer as a word processor, as an automated spreadsheet, and as a database management system. Although this operational goal is narrower than the stated objective of CS 101, it is still hard

to accomplish using the traditional laboratory instructional method. The difficulty lies in teaching several software packages within the constraints of a term. At Bentley we chose to address this difficulty by complementing the classroom instruction with a structured lab similar to a science laboratory. This section describes the goals and implementation of the laboratory portion of CS 101.

Objective:
The objective of the CS 101 laboratory is to support the learning of computing principles and problem solving strategies by providing an environment where the student acquires the operational knowledge of microcomputer tools by doing structured exercises under the supervision of a lab instructor. **Thus, the lab is not a tutorial session on what is taught in the classroom, instead it supports the classroom instruction by providing a working understanding and the operational skills of computing.**

Just as the organization and structure of a science laboratory is governed by its laboratory manual, the CS 101 laboratory manual drives the CS 101 lab. Thus a major contribution of our approach is the structure and organization of the CS 101 laboratory manual.

The CS101 Laboratory Manual:
The primary objective of the manual is to serve as a text for the CS 101 labs. There is an important secondary objective that stems from the nature of software tools. For example, general purpose software tools are very comprehensive and they cannot be learned and remembered fully. Therefore, a secondary objective of the manual is that it should also serve as a self learning tool and as a reference guide.

To achieve the secondary goal we postulated several design principles for the development of the CS 101 lab manual. They are:

1. Organize the material in small, consistent size, functional modules;
2. Emphasize operational knowledge with "why" explanations;
3. Motivate the need for each module;
4. Learn by doing.

The above design principles were operationalized with the following presentation guidelines:

- Module size is limited to two pages.
- First page describes the concepts and commands using a question-answer mode.
- Second page contains hands-on exercises, tips and traps.
- Exercises are stand alone in the sense they do not require the completion of previous exercises.
- Possible traps are listed to warn the user what can go wrong.
- Tips are presented whenever possible.
- A comprehensive functional index and a command index is provided to facilitate access.

The above principles and guidelines have been successfully implemented for MS-DOS and Joe Spreadsheet [1].

Implementation Details:
In the 1987 Fall semester approximately 1000 students registered in CS 101. Everyone acquired a Hewlett Packard Portable Vectra CS (a dual disk drive IBM XT compatible microcomputer) equipped with MS-DOS 3.2, Wordperfect 4.1, Joe Spreadsheet, Turbo Basic, and student version of Dbase III. Each student enrolled in one 60-student class section and one 20-student lab section. The class was taught by a regular faculty member and it met two days a week for 75 minutes each. The lab section was directed by a graduate assistant and it met once a week for 60 minutes. Students were required to bring their computer to every lab session.

The fourteen lab sessions were partitioned as follows. Two sessions were devoted to MS-DOS, two to Wordperfect, six to Joe Spreadsheet, and four to Dbase III.

Experience has shown that six modules of the lab manual can be performed comfortably in one 60-minute lab session. Students are required to read these modules and attempt the structured exercises before coming to the lab session. In the lab the instructor explains and guides the students to work through the exercises on their individual machines. The modules covered in the lab support the topic taught in the regular class.

The final lab exam tested the students ability to use the software packages learned in the course.

SUMMARY AND CONCLUSIONS

Informal input from the students, faculty and the lab instructors suggests that the structured lab approach we used was very effective in enhancing the learning of software tools despite the few implementation hitches. For example, although every attempt was made to synchronize the knowledge acquired in the lab with the material covered in the class, we found that scheduling logistics did not permit perfect harmony. That is, for a student enrolled in a monday-wednesday class and a friday lab, it is possible that a classroom assignment was due on a wednesday which required the expertise that was scheduled to be developed in a lab on friday. It is even more difficult to coordinate across other courses in the freshman curriculum that require the use of the computer.

In my judgement, at present we have merely adapted the structural component of the labs used in natural sciences and are evolving and learning how to specify the objective and content of each lab session. I must add, however, that the CS 101 coordinator did a credible job of specifying the functional knowledge for each lab session.

The lab manual, which is the tool that drives this concept, is incomplete. The sections on wordperfect and dBase III are under preparation.* The exercises have not been tested for their effectiveness in illustrating the fundamental principles underlying the specific features and commands. Also, the functional index has not been tested to match the search strategy of a novice user.

Despite these shortcomings, our intuition and experience suggests that this approach enhances learning and is effective in supporting the use of the computer across the curriculum.

Reference:

1. Chand, D.R. & D.R. Callaghan " CS 101 Computing Laboratory", Bentley College, Waltham, MA, Fall'87, p.160

---

* The WordPerfect portion of the manual has been completed.

CONSTRUCTING EXPERT SYSTEMS FOR INDUSTRY:

RESULTS IN THE ARTIFICIAL INTELLIGENCE COURSE

Scott C. McIntyre

Artificial Intelligence Applications Lab
College of Business and Administration
University of Colorado
Colorado Springs, Colorado 80933

ABSTRACT

This paper describes results obtained by twenty five students in the author's artificial intelligence course. Students are required to complete an entire expert systems lifecycle, through implementation, in an actual organizational setting. Recommendations are provided based upon those results.

INTRODUCTION

In a previous paper (2) an outline was provided for a first course in artificial intelligence for business majors.

The AI course is offered to senior and graduate level business students. There are three interwoven themes: the technology of AI, the application of AI technology to business, and the business of AI.

One of the requirements of that course is the construction of an expert system from conception through implementation. All students build a rule based expert system for and in cooperation with organizations in the Colorado Springs community. It is recommended that students construct an embedded expert system (ie. one that receives data from or sends data to existing information systems) but a complex standalone system is negotiable. Students' systems are expected to have significant capability but narrow applicability. Since they have only one semester to complete the entire development lifecycle, a minimum of 80 rules is required.

RESULTS AND RECOMMENDATIONS

PARTICIPATING STUDENTS

Table 1 presents a summary of the backgrounds of 25 students who have taken the course in the last two semesters. Of most interest here is the occupation statistic. All but 5 had a job in an organization for which it was natural to build an expert system. This was due to self selection: most were in the

course to learn AI technology for the purpose of applying it to their careers.

Of further interest is the incidence of hardware and software engineers in the class. The Colorado Springs area is a high tech area with many manufacturing, development and research facilities representative of national and local firms. Interest in AI is climbing rapidly among career oriented high tech professionals.

Characteristic		Values
Sex	Male	19
	Female	6
Age	20-3	5
	31-40	16
	41-50	4
Occupation		
	student	4
	technician	3
	hardware engineer	2
	software engineer	12
	manager	2
	lawyer	1
	marketing	1

Table 1. Characteristics of 25 AI Students

PROJECT CHARACTERISTICS

In his lectures on "Choosing the First Expert System Project" the instructor emphasized repeatedly that success in the project may be defined as follows: (1) A failed first system is as valuable a learning experience as a successfully completed system. (2) The first system must be narrow; students are enamored with this promising new technology and tend to promise the world. (3) Gaining the confidence of prospective domain experts is a matter of diplomacy and reassurance; they will not be interested if they feel their jobs are threatened.

Students were relieved to hear the first success parameter. At the end of the semester, those whose projects did not meet initial expectations (about one third) submitted analyses of their failures. In the author's opinion based on extensive experience in industry, these were as valuable as the successes.

Less experienced students gave lip service to the second and third parameters above but tended to violate them soundly. One domain expert absolutely refused a third interview when she realized that the student had every intention of automating her position. Those students who had experience in systems analysis and design (including the lawyer) were much more able to handle the necessary diplomacy and interviewing techniques.

Students specified three types of expert systems. The first was selection, chosen by nine students. A typical system determined eligibility for Veterans Administration benefits and selected broad categories for which the user was eligible. The second system type was diagnostic, chosen by twelve students. One system diagnosed errors in disk drives and suggested corrections. The third system type was analysis, chosen by four students. This system is typically the most difficult. One student is responsible for creating mobilization plans for a local

army base. She developed an expert system to analyze a data base to help create mobilization reports.

Twelve of the twenty five expert systems were considered by the students to be very important to the organization. The other thirteen were categorized as meeting organization's needs, but in a peripheral or less important way. By the nature of the project, no student chose a toy system.

Twenty students chose actual domain experts in the target organizations. The others obtained permission from the instructor to act as their own domain expert. This was primarily because the system each had chosen to build was important to the organization, but one for which the student was the only logical choice for obtaining expertise. Of the students who interfaced with domain experts, fifteen had successful experiences. In this case the students and instructor defined "successful" as an experience where the student was able to get needed information from the domain expert without alienating him/her or without being ignored. Of the five students with unsuccessful experiences, one alienated the domain expert inadvertently (this was the one mentioned above who became threatened that she was going to lose her job), one was unable to establish a working relationship with the domain expert, and three became enmeshed in political struggles which rendered void their efforts.

The knowledge engineering process averaged twenty hours of contact with domain experts. An additional twenty hours was typically spent by the student researching supporting material. The most successful projects were those for which the student gained political support within the organization, came prepared for each knowledge engineering session with goals and questions, and kept the domain expert's interest alive by demonstrating successive prototypes of the rule bases. Students who reported that they felt most comfortable with people at the beginning of the course, reported the most successful knowledge engineering sessions. The most difficult were reported by students with low social needs interviewing domain experts with low social needs.

A phenomenon reported informally by other observers of expert systems development was also observed in these projects. Students who were experienced in the design and implementation of conventional systems had the most difficulty implementing expert systems development. The reasons for this seemed to be two-fold. Some had a difficult time identifying problems which were rule and inference oriented as opposed to algorithmically oriented. Others wanted to design the entire system according to structured design methodologies, dividing rule bases into sequentially oriented, top-down structures. While there is growing evidence (1) that structured techniques can aid in problem definition, students who tried to apply them too thoroughly missed the point of iterative prototyping and had difficulty in the initial stages of development.

Of the systems built, fourteen

of them were forward chaining and eleven backward. For the most part this was due to the requirement that expert systems interface with other systems. A typical system would access a data base, a spreadsheet, or a file of diagnostic data and forward chain to reach a conclusion. Systems that used backward chaining tended to be standalone and diagnostic.

The author noted that students had much more difficulty initiating a first prototype for backward chaining systems. In discussion it became clear that forward chaining systems are enough like conventional IF-THEN-ELSE or CASE oriented systems that students had less conceptual difficulty. The process of working backward to prototype the logic of backward chaining systems became more clear once students studied the rule usage patterns of other completed backward chaining systems.

A final difficulty which students had, and one which expert systems developers have industry-wide, is that of testing. Since sequential paths are not taken through rules, testing techniques are a matter of current research. As a consequence, the author used students' difficulties to illustrate the current state of the industry's problems.

For the most part, students ended up building far less complicated expert systems than they had initially specified. They were encouraged to learn from the scaling-back process, but also to see their results as initial prototypes for far more sophisticated systems.

CONCLUSIONS

There is risk involved in requiring students to build expert systems for organizations in the community. Two factors seemed to contribute most to the successes experienced. The first was that most students were involved in the industries for which they built the systems, and were taking the course primarily to apply expert systems techniques to their careers.

The second factor was the industry experience of the instructor. Discussions of potential pitfalls were incorporated into lecture material along with sometimes striking true examples. Where students ignored advice or fell victim to circumstances, their experiences were used to instruct in a manner that made even failed projects successful learning experiences.

REFERENCES

(1) De Salvo, D.A. and A.E. Glamm and J. Liebowitz, "Structured Design of an Expert System Prototype at the National Archives," in Silverman, B.G. (ed), Expert Systems for Business, Addison-Wesley, Reading, Massachusetts, 1987.

(2) McIntyre, S.C. and R. Discenza and L.F. Higgins, "Artificial Intelligence for Business: A Course Proposal for the 1986 DPMA Model Curriculum," Proceedings of the 1987 Information Systems Educators Conference, San Francisco, CA.

MAXIMIZING DATABASE ACCESS INDEPENDENCE

John Boggess, Purdue University, West Lafayette, IN 47907

ABSTRACT

Logical data independence is the ability to change data access by programs without having to change the way data is defined or stored. *Physical data independence* is the permits storing data different ways without having to change the way data is defined or accessed. Database management systems provide varying degrees of data independence, depending on their architecture, data definition, and storage capabilities. This paper focuses on *physical* data independence, or the ability to change stored record formats, access methods, and data placement (for better disk utilization or access performance) without having to make major changes to the overall database definition or programs using the data. We especially want to look at ways to make programs more independent of particular database management systems (DBMSs). We would like to be able write programs now that could be used with different DBMSs later with very few changes. This is not only more flexible than the way programs typically are written, but permits standardization and optimization beyond that given by database management systems themselves.

INTRODUCTION

The degree of data independence directly affects how flexible systems development is in a database environment[1,2]. To meet development needs, it must be possible change how the database is stored and to update the overall database definition with little impact on existing programs.

Organizations which are considering database technology, but are not ready to install a database management system, do not want to have to re-do development when they are ready for database. Many IBM shops still use VSAM master files with CICS as the backbone of their online processing. They must be able continue development in such a way that they can install a database management system later with minimal re-work of these systems.

One strategy an organization can follow is to get training in database design techniques, then design VSAM or other conventional files so that they conform to these design guidelines as closely as possible. When the time comes to convert to a database management system, the conversion may then involve little more than mapping conventional records to records or tables supported by the DBMS.

Major database vendors also provide facilities to ease the transition from VSAM and other access methods to their database management environment.

DATA INDEPENDENCE

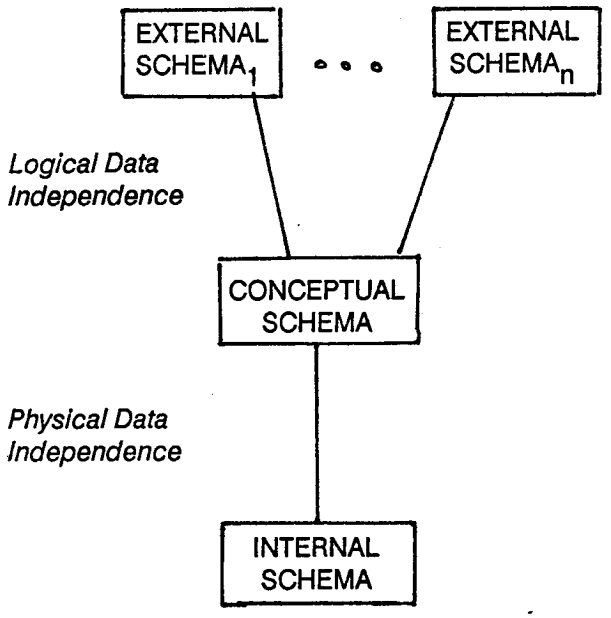


Figure 1. ANSI/SPARC Three-Schema Architecture

The above diagram represents a 3-schema architecture[3], where a *schema* is a logical model of a database. Data definition languages are used at each level to define schemas in a form which can be understood by a database management system.

The *conceptual* (schema) level represents an integrated view of the entire database. It maps to an *external* or *user view* level (that subset of the database a user needs and is authorized to access). The conceptual level also maps to an *internal* or *implementation view*, which defines data in the database as actually stored on disk or other secondary storage devices.

To maximize *logical data independence*, we want to be able to add to or change a conceptual database definition on an ongoing basis without affecting existing programs or how we store existing data. Only programs dependent on updated database structures should have to be changed or recompiled.

For *physical data independence*, we want to be able to move all (or part of) a stored database to another physical location or storage device without having to change the conceptual database definition or recompile programs. We would also like to be able to use different access methods (perhaps another database management system) or to change how records are stored in the database (different record clustering, compressed records, etc.).

How effectively access methods and database schema definition languages support the above objectives determines the degree of data independence you have with a given implementation strategy.

DATA INDEPENDENCE FACILITIES PROVIDED BY DBMSs

In addition to the general data independence features discussed above, other facilities are provided by some database vendors. Two key examples are discussed below.

ADR's VSAM and DL/1 Transparency Products[4,5]. To make conversion from existing VSAM or DL/1 files to their DATACOM database management system more practical, ADR provides "transparency" products which perform many of the translations required. With these products, application programs do not need to be modified to use DATACOM (a key factor for having a high degree of data independence). Instead, program calls for VSAM or DL/1 accesses are intercepted and translated into corresponding calls to DATACOM. Status codes and record images are returned to the program just as they would have been from VSAM, DL/1, or IMS systems.

Before existing VSAM programs can be used with DATACOM, it is necessary to convert existing files to DATACOM databases. First, file definitions are extracted from programs using the VSAM files and used to create DATADICTIONARY definitions required by DATACOM. In the new database that is defined, DATACOM records (tables) will have basically the same format as corresponding VSAM records. A utility program can then be used to read existing VSAM files and create a sequential

DATACOM load file. A standard DATACOM utility is used to process this load file and populate the DATACOM database that was defined.

Converting from a DL/1 database is slightly more complicated. First, an automatic database redesign facility reads DL/1 Data Base Definition (DBD) and Program Specification Block (PSB) source statements to create DATACOM control blocks. The DATADICTIONARY is then populated with database names, tables (corresponding to DL/1 segments) and keys (corresponding to keys and imbedded pointers in segments). After this has been done, steps similar those described for VSAM can be followed to transfer data from a DL/1 database to DATACOM and permit existing programs to be used against the new DATACOM database.

There are several major advantages to the above approach. First, an installation can quickly convert existing databases to a DATACOM environment and begin development of new applications using ADR's IDEAL fourth-generation language, DATADICTIONARY, DATAQUERY, and related tools. Second, there has been minimal impact on existing applications, since it has not been necessary to re-write programs to include DATACOM calls. Finally, the resulting environment is often much more flexible, since DATACOM provides both relational processing capabilities and tuning capabilities which keep performance acceptable (and in some cases better than with the original VSAM files).

Cullinet's Logical Record Facility (LRF)[4,5]. This product also provides distinct data independence benefits, but in a much different way than with the transparency products discussed above. LRF allows application programs to access IDMS/R data without having to know the physical structure (records and sets) of the database. This is done by selecting desired fields from database records (much as you derive views from SQL tables in a relational environment) to define a program view of the data (i.e. a logical record). Paths can then be defined to control how these logical records are accessed. Instead of issuing a series of IDMS database commands to get all the records associated with an order line (for example), the programmer can issue a single "OBTAIN ORDER-LINE-LR" command to get customer, order, and product values for a given order line. LRF also supports relational operations (select, project, join) not supported with standard IDMS database access commands.

Note that LRF insulates programs from the database, so that changes to logical and physical database structures have minimal impact on existing programs (i.e. LRF provides a high degree of both logical and physical data independence). This translates into a number of programming advantages, including:

(1) Simplified program access. Programmers do not need to know the database structure or understand navigation techniques and database currency concepts to use LRF.

(2) Reduced programming effort. Once again, programmers do not need to include instructions for individual database record accesses.

(3) Increased data integrity. The database administrator (DBA) can write database navigation instructions in such a way that logical relationships and other integrity constraints are preserved.

(4) Additional data security. Security constraints can be applied at a program view (logical record), in addition to other security which may be appropriate.

(5) Better run-time efficiency. A single program call can result in a number of database accesses being performed by the database management system before returning to the program (e.g. requesting an order line logical record, which results in customer, order, and product record accesses being done by the

DBMS before returning to the program). Operating system overhead can be reduced considerably with such an approach.

ADDITIONAL DATA INDEPENDENCE OPPORTUNITIES

Going beyond the facilities provided with different database management systems and related products, an installation can do several things to improve data independence. (It is important to note that most of these apply whether or not you are using a database management system.)

The concepts of *transparency* and *program views* (logical records) discussed above can be applied to data accesses in general. By introducing an intermediate level of processing, it is possible to insulate *all* data(base) accesses from an application program and handle them in standard processing routines. While this has the major disadvantage that someone must develop these routines up front, such an approach can provide a number of longer term advantages:

(1) Access modules, which do processing similar to that described for LRF, can be developed. These standard modules are passed commands and "views" of data to be added or updated, and return standard status indicators and "views" of data being retrieved. Different modules can be developed to perform the same functional processing (from a program point of view), but use different access methods (e.g. one might use VSAM, another IDMS, and a third use SQL as the access method).

(2) Different libraries can then be used with programs without having to change or recompile them. For example, a program could use a test data access module library initially, then refer to a production library later when turned over for production. The program might initially use VSAM for access, but later point to a library which uses corresponding SQL calls.

(3) Actual data definitions are stored in the access modules and can be changed transparently. Separate user view definitions can be maintained in a COPYBOOK or other source statement library for use by both application programs and the data access modules.

(4) Because programmers are not making direct access method or database calls, changes to access strategies will be transparent to the programmer.

(5) Naming and coding conventions, plus additional security and integrity checks, can be incorporated into access modules and used to enforce installation standards. This also reduces program redundancy, since it is not necessary to include these program checks and balances in each program. Similarly, if a standard changes, it will be reflected in all programs which use the access routines enforcing these standards. This avoids the problem of making changes to some programs which are affected, but not all of them.

The above approach requires a long-term commitment to providing as flexible a data environment as possible.[8,9] It is most workable a in predominantly batch processing environment, and in installations which recognize the *need* to use a database approach but do not have the *resources* (people, time, tools) to do so at this time. But the functions described above are all basic Database Administration needs which must be addressed, whether or not you have assigned an individual DBA responsibility or implemented a database management system.

SUMMARY

Data independence is an important aspect of database that does not get the attention it deserves. Most people recognize intuitively that you do not want to store data redundantly because it wastes disk space, is more difficult to update, etc. But problems associated with low levels of data independence tend to be more subtle. Only when you need to *change* a database definition, for example, and find that you have to recompile all your programs which access that database do you recognize the importance of data independence.

REFERENCES

1. Date, C.J. *An Introduction to Database Systems*, vol.1, 4th ed. Addison-Wesley, Reading, MA, 1986.

2. McFadden, F. and J. Hoffer, *Data Base Management*, 2 ed., Benjamin-Cummings, 1988.

3. ANSI/X3/SPARC Study Group, Database Management Systems, "Framework Report on Database Management Systems," *AFIPS Press*, Montvale, NJ, 1978.

4. Applied Data Research, *ADR/DL1 Transparency*, 1987.

5. Applied Data Research, *ADR/VSAM Transparency*, 1987.

6. Cullinet Corp., *IDMS Logical Record Facility Programmer's Guide*, 1983.

7. Cullinet Corp., *IDMS/R Logical Record Facility DBA's Guide*, 1986.

8. Martin, J., *Strategic Data-Planning Methodologies*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

9. Martin, J., *Managing the Data-Base Environment*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

THE USE OF A MODELING TOOL IN A COMPUTER NETWORKS COURSE

Jack E. Leitner, PhD
Division of Computer and Information Sciences
University of North Florida
Jacksonville, Florida

ABSTRACT

Realistic projects for the study of computer communication networks and their performance typically involve more mathematical rigor than is practical in a course for information systems students. This paper discusses a software modeling tool that has proven to be a valuable teaching aid in that respect because of its ease of use and powerful simulation properties.

INTRODUCTION

The need to transfer and exchange data between computers and terminals has caused the establishment of computer networks. These networks have expanded to include not only a host computer and the users' terminals but also multiple mainframe computers, personal computers and a variety of additional terminal equipment such as remote printers, plotters, point-of-sale terminals, and bank teller machines. To support and manage the flow of data between these units other devices such as modems, multiplexers, concentrators, switching nodes, and front end processors have been inserted into the network. To transport the data between these devices various transmission paths are utilized, often the public telephone system for wide area networks or specialized media for local networks. Many of the network devices are controlled by stored programs consisting of algorithms for access, routing, flow control, and error control. To assemble these components into a functioning system requires an organized method of analysis and design. The study of networks, therefore, can include all these topics and more.

A NETWORKING COURSE

The University of North Florida's Division of Computer and Information Sciences has added to its curriculum a one semester course for senior undergraduates and beginning graduate students to address the study of networks. It resides in the Information Sciences/Systems tracks and does not assume an extensive background in mathematics. Prerequisites for the course include Data and File Structures, Systems Software, System Analysis and Design, and Systems Implementation.

Because the course is only one semester it is impossible to cover the entire spectrum of networking in detail. Some of

the material is covered in a sister course, Computer Communications. That course encompasses the details of transporting data between two adjacent devices and how most of the communication devices function. The material covered in the Computer Communications course corresponds to the lower two levels of the International Standards Organization's (ISO) reference model. The thrust of the Networks course is to cover the other five layers of the ISO model. Particular emphasis is given to the problems involved in designing data networks and evaluating their performance.

Designing Data Networks by Robert Ellis [2] is currently being used as the text. It has a good discussion of the process of network design as the process of allocating resources within a network to accommodate the demand of the network's users, while still meeting the cost and performance goals set by the organization. Thus, it covers not only the components of networks but also the process of assembling the diverse parts into a functioning whole. Ellis stresses the design process for wide area networks, both centralized and distributed. The OSI model does not receive much attention, so additional readings are assigned to cover this. Local area networks (LANs) are not covered in detail in this course because they are included in another course on office automation.

DIFFICULTIES WITH PROJECTS

A perennial problem in a networks course for information systems students is finding appropriate assignments. The assignment should illustrate various concepts covered in the lectures and they should have at least some degree of practicality. Unfortunately, the exploration of the operation of a network and the analysis of its behavior tends to be a mathematically intensive process, if done on a theoretical basis, or an expensive process, if done on an empirical basis. The mathematics background required of the students in this course is limited to one semester of calculus and an elementary statistics course, so any mathematical treatment must necessarily include a great many simplifying assumptions. The resulting paradigm may or may not accurately reflect the system under study.

An alternative to manual or analytical approaches to network analysis is to build a model of the communications system using a software simulation tool. Unfortunately, these tools are often prohibitively expensive. Even if they are affordable they frequently require learning a new programming language and some familiarity with creating models. Teaching this in a networks class takes too much time away from the presentation of material about networks.

A MODELING TOOL SOLUTION

One solution to these problems is a professional modeling tool from CACI, Inc. offered at a very attractive price through their educational discount program. This tool, NETWORK II.5, uses a generalized modeling approach to provide a simulation technique for communication system analysis. A model of the system

under study is built by describing it in terms of NETWORK II.5 components. A simulation of the model is then run and yields measures of hardware utilization, software execution, response times and the effect of conflicts between devices on total system performance. Systems can be modeled that range from a single processor with just a few terminals to a complex system of processors and communication devices connected by data links into a large network. It is possible to model portions of a network at a detailed level while the rest of the network is modeled at a coarser level.

The network to be simulated is described to the tool in terms of the system's hardware and software building blocks. Each of these building blocks has a series of attributes whose values are supplied by the user. The data structure describing the system is assembled in a text file which is then read by the NETWORK II.5 program, the simulation executed, and the results printed.

Hardware devices are specified from a group of three basic types: processing elements, data transfer devices, or data storage devices. The users characterize the processing elements (which are the logical elements in the model) by building an instruction set and specifying a basic cycle time. The instruction set allows the processing elements to communicate by reading or writing from the storage devices, sending messages over the data transfer devices, and setting globally read flags. Data transfer devices are characterized by their data transfer rate, data transfer protocol (selected from five preprogrammed protocols), and connections to specific hardware devices. Data storage devices are characterized by capacity, access time, and access method.

The software of the simulated network is specified as software modules. Each module contains a specification of the processing elements that are allowed to execute it, when it may run, what it is to do when running, and which other modules (if any) are to start upon completion. The user can specify the conditions that must be met before a module can start executing. These conditions include the availability of hardware, the receipt of a message, the state of a flag, and whether copies of the same module may execute concurrently on different processing elements.

An interactive input program is provided with the system. It is command driven but has an extensive help function. The program prompts for the information needed to describe the different building blocks. It permits default values for some of the attributes and performs a range check on the numerical values. The output of this program is a text file containing the data structure describing the system.

The standard output after the simulation is executed consists of several tabular reports. They show the statistics on the operation of each device or module in the simulation. Average, maximum, minimum and standard deviation values are reported for execution, delay, and queue sizes. A "snapshot" report gives the

status of each entity as of the end of the simulation. While the simulation is in progress it is possible to have a trace report generated that gives the step-by-step execution. Also available are time based plots showing the percentage of time a device was busy during the simulation. NETWORK II.5 has proven useful as a vehicle for assignments in the networks course. It is possible to assign projects that illustrate a particular facet of network behavior. The student can build the model and then easily and dynamically alter the model and see the effect on the model's performance.

As an example, the first assignment in the term was to model a system consisting of three data entry terminals connected to a network through a simple time division multiplexer. The object of the assignment was to have the students become more familiar with the characteristics of the multiplexer and to show how even fast data entry clerks can't achieve high utilization of a transmission line. The simulation showed that the utilization was less than five percent of capacity on even a low speed line, so clearly something needed to be done to more effectively utilize transmission resources.

The second assignment built on the first one and was a model with three concentrators connected to a host through a switching node. The students were asked to determine the response time at four different utilization rates. This project was designed to show that while higher utilization could be achieved, once utilization went past a certain point response times became unacceptably long. Both the first and second assignments have been included in Appendix A.

CONCLUSIONS

The use of NETWORK II.5 is not without it's drawbacks. At the beginning of the term some class time must be spent instructing the students in the use of the tool. There is some difficulty learning the syntax used to described a model, although the extensive on-line help facility with the program makes this somewhat easier. Another part of the problem is that the students are not very familiar with the concept of using a model. This generates comments like "I told it that I wanted the network to run for 5 minutes and it came back after 30 seconds and said it was done." and "How does it send a message called 'Character from terminal A' if I'm only sending eight bits?" However, by starting off with relatively simple models at the beginning of the term the students can develop increasing expertise in the use of the tool until quite sophisticated networks can be modeled.

Overall the use of Network II.5 in an introductory course in networks has been successful. It has given the students a better comprehension of how a network functions and an intuitive grasp of how performance parameters are affected by the various design considerations.

BIBLIOGRAPHY

1. Black, Uyless. Computer Networks: Protocols, Standards, and Interfaces. Prentice-Hall, Inc. Engle-

wood Cliffs, New Jersey. (1987).

2. Ellis, Robert L. Designing Data Networks. Prentice-Hall, Inc. Englewood Cliffs, New Jersey. (1986).

3. Hammond, Joseph L. and O'Reilly, Peter J. P. Performance Analysis of Local Computer Networks. Addison-Wesley Publishing Company, Inc. Reading, Massachusetts. (1986).

4. Leigh, William E. and Burgess, Clifford. Distributed Intelligence: Trade-offs and Decisions for Computer Information Systems. South-Western Publishing Company. Cincinnati, Ohio. (1987).

5. Sharma, Roshan Lal., de Sousa, Paulo T., & Ingle, Ashok D. Network Systems: Modeling, Analysis and Design. Van Nostrand Reinhold Company. New York, New York. (1982).

6. Tanenbaum, Andrew S. Computer Networks. Prentice-Hall, Inc. Englewood Cliffs, New Jersey. (1981).

APPENDIX A

HOMEWORK ASSIGNMENT 1

Using NETWORK II.5, model a system consisting of three asynchronous EBCDIC terminals and an asynchronous time division multiplexer. The three terminals output a character at a time at 1200 BPS and the output from the TDM is on a character by character basis at 3600 BPS. The terminals are each directly connected to the multiplexer by their own point-to-point line. The multiplexer sends its output to some unknown device.

The input to the terminals is from the keyboards which are operated by data entry clerks. The clerks average typing at a speed of 50 words per minute (a word has five characters), never going more than 90 words per minute or less than 10 words per minute. The variation in keystroke speed is a normal distribution with a standard deviation of .1 seconds per keystroke.

Build the model so that the multiplexer assigns the output time slots on a fixed interval rotating basis to each terminal in turn. (The terminals won't always have something to send so some of the output slots from the multiplexer will be empty.) Turn the queue flag on in the message instructions on the multiplexer so that the input queue at their destination will show what has been sent to it.

Run the simulation for 5 seconds and turn in the output from the NETWORK program consisting of the:

Input playback report
Trace report of the multiplexer for one second
Summary reports
Ending snapshot reports

HOMEWORK ASSIGNMENT 2

Using NETWORK II.5, model a system with three concentrators that each connect several terminals to a message switch which is then connected to a host. This is a synchronous system that uses SDLC for its link protocol. All of the

links in the system are FDX and operate at 9600 BPS. Every device is connected in a point-to-point configuration. The messages in this system consist of 50 byte queries from the terminals which are answered by 1000 byte responses from the host. (Assume for this system that SDLC will put 1000 bytes in one frame.)

Instead of including the terminals in your model simulate their transmissions to the concentrators. The system should reflect queries arriving with equal probability at any interval above one per second. Instead of transmitting responses to them just have a process in the concentrators that disposes of any incoming responses. Since the queries and responses may occur faster than some of the devices can accommodate them be sure to allow more than one copy of the messages to exist in the queues and the modules to execute concurrently.

Use this model to investigate the effect of resource utilization on response time. Determine the overall response time (the interval between when the user presses the enter key on their terminal and when they first start seeing a response) when the utilization of the most congested link is 70% + 2%, 80% + 2%, 90% +2%, and 95% +2%. Assuming each terminal generates a query at the rate of one per minute, calculate how many terminals each concentrator is serving.

Run the simulation for 10 minutes at each of the utilization levels. Reset the statistics after 1 minute and turn in the output from the NETWORK program for each:

Input playback report
Summary reports
Ending snapshots

Computerized Testing: a Wave of the Future?

Thomas C. Richards, J. B. Spalding, and Tom Marshall

University of North Texas
College of Business Administration
University of North Texas
Denton, Texas 76203

ABSTRACT

Based on our experiences, we believe that computer administered examinations will play a growing role in our introduction courses at the University of North Texas. This application of the use of the computer demonstrates to the often unmotivated and uninterested student a live illustration of the use of the computer and may peak his or her interest. The administration of examinations in large sections and multi-sections of introductory level courses has always been a problem. We believe that the use of the computer as a testing tool can overcome some of these problems. This paper outlines our experiments with this method of testing.

THE PROBLEM

The administration of examinations in large sections and multi-sections of introductory level courses has always been a problem. It is particularly difficult in many schools of business where enrollments in the business core courses continue to increase. At The University of North Texas we offer two information systems courses in the business core both of which have an enrollment of approximately 700 students per term. Both large (225 students) and small sections (45 students) are scheduled. We ask the graduate students and faculty members who teach these courses to follow a standardized course outline. We have also been attempting to increase the structure and the uniformity of the instruction in these courses.

One method of achieving this goal is to use standardized final and midterm examinations. This strategy introduces a number of problems including examination security, feedback of examination content between students in different sections of the courses and the cost of duplicating hardcopies of the examinations.

In order to overcome some of these problems, we decided to experiment with computer administered examinations. The automatization of the routine work required for test delivery, assessment and analysis of the results is not a new concept (8,3). We believe that the administration of common examinations via the microcomputer will allow us to address many of the above problems plus provide a number of advantages as outlined below.

THE EXPERIMENTAL DESIGN

During the fall term of 1987, students currently enrolled in our first introduction to information systems course were asked to volunteer to take the second midterm examination using the computer. The midterm examinations in this course consist of fifty multiple choice questions. Two versions of the examination are prepared with the same questions presented in a slightly different order so that students in adjacent seats can not copy from one another. Out of a class of 193 students, 123 volunteered to take the examination on the computer. Twenty-nine

of these students (the experimental group) were matched with classmates with identical scores on the first examination (the control group) and selected to participate in the experimental group.

A commercially prepared microcomputer package was used to administer the same fifty questions used in examination taken by the remaining 164 students in the class. This package presents one multiple choice question at a time to the student. The questions as well as the responses are presented in a random order so students at adjacent workstations are not viewing the same question at the same time. The student can scroll through the exam both forwards and backwards by pressing the appropriate function keys. He or she can mark each question with a tentative correct answer by pressing another function key. By pressing the return key the question is answered and no longer can be viewed by the student. The answer to a marked question can be changed where an answered question can not. Students were told to mark each question and then answer them after they were sure that all questions had been correctly answered.

Each student was assigned a microcomputer in one of our micro laboratories which had been reserved for the administration of the examination. Both the experimental and the control group took the examination during the same time interval so that we had no cross fertilization between the experimental and the control groups.

The Wilcoxon Matched-Pairs Signed-Ranks Test was used to determine if there was a significant difference between the mean scores received by the control group and the experimental group. This research hypothesis was rejected at the .05 level and we concluded that there was no difference in the mean scores between the experimental and control groups. At least one other researcher (4) has found this to be true in the use of computer administered examinations.

All students completed the examination within the 60 minute period. Students taking the computer administered examination were all familiar with microcomputers since we teach the use of several packages in these courses. There was no significant learning time required to use the examination package since the use of only four keys was needed to complete the examination. Students were given their scores on the examination as soon as they completed it. Later they wanted to know how they had responded to each question. Unfortunately the package we used does not record this information. We also had some complaints about the noise level generated by the microcomputers in the laboratory.

FUTURE RESEARCH EFFORTS

Based on this experiment we believe the administration of examinations on the microcomputer is pedagogically sound and cost effective. We are now undertaking additional efforts to fully develop a comprehensive system for the administration of examinations using this method.

Our first efforts will require the building of a testing package to fit our particular needs. We found the package we used, in additional to the problems mentioned above, had a number of other deficiencies and believe it would be to our advantage to develop our own package which fits the particular needs of our institution and courses. We have completed an initial version of this package and we are now experimenting with its use.

Some of the primary features of our package for the administration of examinations on the microcomputer include: the option of the student changing his or her response to any question, the recording of the students response to each question on the exam for future reference by the student, the collection of response information for all students on each question for further statistical analysis, the networking of the microcom-

puters to a central file server to increase security of soft copies of the exams, the option of printing hard copies of an examination and the inputting questions from other data banks of questions into the test question databases.

The current version of our program will run on an IBM PC or compatible machine with 640K memory and a hard drive. A color monitor adds to the visibility of the screens. Future enhancements to the package will include the option of the student self-testing over any given selection of topics. The system will then provide the student with a profile of topics which require further study. Since the system can generate unique examinations from a question databank, it would be possible to let a student retake an examination several times and perhaps improve his or her grade.

COMMENTS ON COMPUTER ADMINISTERED EXAMINATIONS

Based on our experiences, we believe that computer administered examinations will play a growing role in our introduction courses at the University of North Texas. In additional to the advantages mentioned above, the following should be considered. This application of the use of the computer demonstrates to the often unmotivated and uninterested student (7) a live illustration of the use of the computer and may peak his or her interest. It is possible to use the computer for implementing the delivery of diagnostic tests (2) as suggested above. Taking this concept a step further, the implementation of self-organizing tests is possible where the administration of the test items to a particular student is controlled by the students previous answers. This concept moves into the realm of intelligent tutoring systems (9).

The question of the degree and amount of feedback (1,6) to the student is also important. Fletcher (4) concluded that immediacy of scoring and immediate feedback on incorrect answers was a characteristic which students felt was a major advantage of computer administered examinations. Students also found these exams more convenient, straight forward, easy to use and faster than written exams. The particular package we used in our experiment had an option which displayed the number of correct responses each time the student answered a question. We turned this feature off during our experiment since it would introduce one additional effect. In the future we plan on providing the student with the option of setting a clock which will display his or her remaining time to complete the exam.

Each course using this system can have its own database of questions which are appropriately keyed to topics and subjects. These questions can then be used to build an examination. Such examinations can be generated with a predefined number of questions randomly selected from designated subject matter topics. As statistics are automatically collected on each question in the database for a course, difficulty levels can be statistically determined for each question. These difficulty levels can then be used to further refine the characteristics of a particular examination.

We felt it was important to allow the student to change his or her answer to a question as the examination was taken as well as being able to scroll through the exam. The lack of this option was found to be a major disadvantage of computer administered examinations (4).

The use of the computer to administer examinations opens up the realm of graphics, animation and simulation. For example, when questioning students about the components of a computer, the student could be shown a picture of a computer and asked to identify the desired parts. When asking questions about a microcomputer package, they could be shown various screens and output generated by the package and

asked appropriate questions. Via simulation they could be asked to perform and be tested on the required skills on a word processor or spreadsheet including inputting, formatting and editing.

CONCLUSION

From the students' point of view, a computer administered examination provides a number of advantages over a written examination including immediacy of scoring, convenience, and ease of taking. From the instructors' point of view it will provide test security, flexibility, control and low cost. There are however, substantial front-end costs for both software and hardware. If these costs are amortized over several years the cost savings are substantial. A well design package could be of significant value to other institutions. For example, we estimate a cost savings of $8,000 per year in our courses if we implement this package.

REFERENCES

1. Birenbaum, Menucha and K.K. Tatsuoka, "Effects of On-Line Test Feedback on the Seriousness of Subsequent Errors," **Journal of Educational Measurement**, Vol. 24., No. 2, Summer 1987.

2. Ferraris, M., V. Midoro and G. Olimpo, "Diagnostic Testing and the Development of CAL Remedial Sequences," **Computers in Education**, Vol. 8, No. 4, October 1984.

3. Feuer, Dale, "Computerized Testing a Revolution in the Making," **Training**, Vol. 23, No. 5, May 1985.

4. Fletcher, P. and M.A.J. Collins, "Computer-Administered Versus Written test - Advantage and Disadvantages," **Journal of Computers in Mathematics and Science Teaching**, Vol. 6, No. 2, Winter 1987.

5. Janaro, R.E. and W. J. Mein,"A CAI Approach to Teaching the Introductory Business School Computer Course,"**Interface**, Vol. 9, No. 4, Winter 1987-88.

6. Merrill, John,"Levels of Questioning and Forms of Feedback: Instructional Factors in Courseware Design," **Journal of Computer-Based Instruction**, Vol. 14., No. 1, Winter 1987.

7. O'Brien, J.A., "Motivating the Non-Information Systems Student," **Interface**, Vol. 5, No. 2, Summer 1983.

8. Smith, L.S., "Computer Assisted Testing a Bibliography," **Computers in Math and Science Teaching**, Vol. 1, No. 3, Spring 1987.

9. Wenger, E., **Artificial Intelligence and Tutoring Systems**, Morgan Kaufman Publishers, Inc., Los Altos, 1987.

Personnel / Human Resources Management and Information Systems: A Combination Whose Time Has Come

Jon W. Beard
Texas A&M University

The study of information systems (IS) has involved the evolution and integration of diverse fields. But, if the evolutionary process stops with a field absorbing ideas and not applying them to other disciplines a great deal of the potential is lost. This paper discusses the possible integration of IS with personnel / human resources management.

The study of information systems (IS) has involved the evolution and integration of fields such as organization science, management science, and computer science(1). This integration and subsequent application should continue. Information systems, to be a useful field of research, must be applied to the specific and sometimes peculiar needs of other disciplines. One discipline recently strengthening its association with the study and use of IS is the area of personnel and human resource management (PHRM). The integration of these two different disciplines is called human resource information systems (HRIS).

An HRIS is an inventory of an organization's positions and the characteristics, skills, and experiences of its employees(18). This information has always been available, though it has rarely been centrally collected or maintained. Recent changes in the competitive environment and legislative imperatives have led organizations to search for new ways to optimize their productivity. With the proliferation of microcomputers, the rapid reduction in computing costs, and the development of easier to use software systems, it has become possible for almost any organization to create a computer-based HRIS. Now computers assist in every area of PHRM (8). But PHRM has not always been at the forefront of technology use.

HISTORICAL OVERVIEW

The 1940s and 1950s saw the advent of automated payroll systems. Limited information, such as an employee's name, salary, birth date, sex, department code, and so on were kept and processed. But few other PHRM tasks beyond payroll processing were automated in most organizations.

The advancement of computer technology accelerated in the mid-1960s. Computer systems became increasingly elaborate and technology quickly outrun the needs of all but the most sophisticated personnel departments. Only the aerospace and defense-related industries moved into nonpayroll personnel systems (22). These systems often contained employee skill inventories. This information could be rapidly matched with organizational needs to facilitate rapid employee placement on company projects. Most organizations, however, did not follow this path. For instance, during the 1970s, the federal government used data processing more extensively for PHRM activities than did most private sector organizations (22).

In recent years organizations in general, and personnel departments in particular, have come to realize that integrated information systems can and should be developed using the capabilities of computer technology. This change in thinking has been partially spurred by the growing body of legislation requiring reports on equal employment opportunity (EEO), the Occupational Safety and Health Administration (OSHA), and affirmative action programs. Beyond these mandated requirements, however, organizations have begun to see the potential for making use of IS technology to assist in improved management for making use of IS technology to assist in improved management personnel development.

But, there seems to be a major piece missing in completing this process. There are many human resource specialists available as well as a substantial number of information system specialists. Unfortunately, there are few who can claim to be specialists in both disciplines. The remainder of this manuscript will discuss the potential for combining the study of IS with the tasks of PHRM.

PHRM

PHRM consists of many different, yet, interrelated activities. PHRM has been very broadly defined as the most effective use of employees to achieve organizational and individual goals (9). Three of the many PHRM activities will be used here as examples of what can be done with HRIS.

Planning. Organizations are concerned about having adequate numbers of trained people on hand when critical projects are undertaken (3). As with most organized endeavors, ensuring that qualified people will be available to fulfill organizational needs requires planning. This becomes increasingly important as the economy becomes sluggish and competition increases. Census information in an organization's employees can be compared to the job descriptions and characteristics of available positions. Manually this comparison process would be tedious, if not impossible. With an HRIS, employees can be easily matched with positions. Information in job openings can be compared to application information in order to make better selections of employees.

Planning can also involve career management for an organization's employees. The system could track an employee's past positions and training. Combining this information with individual career goals, as well as with organization needs and goals, can provide a much more efficient development process for employees. Future organizational growth can be forecast and compared to present employee capabilities for planning future recruiting.

Benefits. Employee benefits, such as health and dental plans, as well as salary options, could be flexibly handled with an HRIS. With a manual system, allowing a variety of options for each employee can quickly become unmanageable. A computerized system could allow the flexibility for each individual to tailor the benefit options to his or her specific needs.

Performance Appraisal. One last area which will be mentioned is the use of an HRIS to assist in performance appraisal. One problem in performance appraisal has been fairness -- how does the organization get fair evaluations. Software systems have been developed which can help monitor the employee appraisal and evaluation process. This can consist of comparing one person's evaluations with past evaluations to look for any discrepancies. This type of comparison may detect bias, either positive or negative, for or against an individual.

Another problem has been in how to provide appraisal feedback to employees. Peterson(17) recently completed a study where a computerized expert system was used to assist managers in preparing to provide the performance feedback to their subordinates. The results suggest that this type of assistance might provide an organization with an opportunity to provide better appraisals by directing the supervisor to focus on the appropriate means of providing the feedback to the subordinates.

Other areas for HRIS use include recruiting, employee selection and placement, computer-based training, and counseling, to name just a few.

A CRY FOR ATTENTION

This cry for attention is not without some justification. Among the popular texts in PHRM only a few mention HRIS (e.g. 15, 18). Schuster (19) even provides a chapter on HRIS. But this material is five pages long, followed by twenty pages of short readings and cases.

Zientra (24) has reported that nearly 80% of all U.S. corporations with over 5000

employees have a formally established HRIS. HRIS budgets are estimated to be growing at over 40% annually(13). Most of the HRIS managers come from personnel backgrounds (24). Desanctis (2) suggests that the concerns of HRIS are generally quite similar to those of management information systems. Yet, one unpublished manuscript cited by Desanctis found that HRIS managers did not view data processing as a "valuable resource" (p.22).

The literature to alleviate the gaps in a PHRM professional's knowledge about HRIS does exist. Some of the topics include discussions on whether to make or buy an HRIS (20), what software is available for use in various PHRM functions (4,5,6), how to pick software (14,16,23), how to build your own system (7,11,12), and common mistakes in building an HRIS (21). Walker (22) has even produced a book devoted entirely to developing an HRIS. Other literature is devoted to more specific application as described earlier. But, reading the material and being able to use the information, if it is even known to exist, are not the same thing.

DISCUSSION

How do we solve the problem of separate disciplines? One answer with great potential is to provide instruction on HRIS. Typically, computer instruction in a PHRM course might consist of word processing and simple spreadsheet and/or database applications. In an information systems course, there would be a project of some sort. Why not combine the two courses into one? This is not intended to suggest that either the general PHRM or MIS course (or both) be replaced. What is being suggested is that a followup course be provided which allows for and encourages the specific and detailed integration of IS with an applied discipline, such as PHRM. This course would not only provide an integration of the two disciplines, but would also create the necessity for students to review what they had learned in a previous course and apply that knowledge to very realistic problems in an area of high demand. What better way to reinforce learning and show how our sometimes vague and "fuzzy" information systems concepts can be applied to real problems and situations.

One of the statements in the CIS curriculum that has been developed by the DPMA suggests that coursework should provide students with an experience that resembles business practices as closely as possible. The CIS 86-8 course, which is a systems development course, is a very good location for learning of systems such as HRIS.

An HRIS course as envisioned here would provide opportunities for students to apply all of the IS skills learned in earlier courses (e.g. systems analysis, prototyping, information resource planning, data bases, and perhaps expert systems and decision support systems). This course would allow students to use the skills they are learning for realistic problems. Finally, and perhaps, most importantly, a course of this type would provide the opportunity for different disciplines to work together toward a common goal -- the education of students. This cross-fertilization of ideas can only be beneficial to both disciplines and to our students.

REFERENCES

(1) Culnan, M. J. & Swanson, E. B. (1986) "Research In Management Information Systems, 1980-1984: Points of Work and Reference," MIS Quarterly, 10(3), 289-301.

(2) DeSanctis, G. (1986) "Human Resource Information Systems: A Current Assessment," MIS Quarterly, 10(1), 15-27.

(3) Devanna, M. A., Fombrun, C. & Tichy, N. (1981) "Human Resources Management: A Strategic Perspective," Organizational Dynamics, Winter, 51-67.

(4) Frantzreb, R. B. (1986) "Microcomputer Software for Human Resources: A Directory," Personnel Administrator, July, 65-79.

(5) Franztreb, R. B. (1987) "Microcomputer-Based HRIS: A Directory," Personnel Administrator, 32(12), 84-94.

(6) Frazee, J. M. & Nichols, R. O. (1986) "Mainframe Human Resource Software: Current Issues and Market Trends," Personnel Administrator, July, 83-94.

(7) Gridley, J. D. (1987) "Six Steps to a More Efficient HRIS: A Pre-Vendor Evaluation," Personnel, April, 64(4), 8-12.

(8) Harris, D. (1986) "Beyond the Basics: New HRIS Developments," Personnel, January, 49-56.

(9) Ivancevich, J. M. & Glueck, W. F. (1986) Foundations of Personnel / Human Resource Management (Plano, Texas: Business Pubilcations, Inc.).

(10) Kustoff, M. (1985) "Assembling a Micro-Based HRIS: A Beginner's Course," Personnel Administrator, December, 29-36.

(11) Lederer, A. L. (1984) "Information Technology: Planning and Developing a Human Resource Information System," Personnel, May-June, 14-27.

(12) Lederer, A. L. (1984) "Planning and Developing a Human Resource Information System," Personnel Administrator, August, 27-39.

(13) Magnus, M. & Grossman, M. (1985) "Computers and the Personnel Department," Personnel Journal, April, 42-48.

(14) Mahal, D. & Magnus, M. (1987) "Microcomputer Software Buyers' Guide," Personnel Journal, April, 73-88.

(15) Milkovich, G. T. & Glueck, W. F. (1985) Personnel / Human Resource Management: A Diagnostic Approach, Fourth Edition (Plano, Texas: Business Publications, Inc.).

(16) Munn, R. W. (1985) "HRIS: Towards An Effective Requirements Analysis," Personnel Administrator, May, 14-15.

(17) Peterson, T. O. (1988) "The Acquisition of Managerial performance feedback Skills through the use of a Knowledge-based expert system: An Empirical Evaluation," unpublished dissertation, Texas A&M University, College Station, Texas.

(18) Schuler, R. S. & Youngblood, S. A. (1986) Effective Personnel Management (West Publishing Co., New York).

(19) Schuster, F. E. (1985) "Human Resource Information Systems," in Human Resource Management: Concepts, Cases, and Readings, 2nd Edition (Reston, Virginia: Restion Publishing Co.) 543-567.

(20) Tollberg, D. (1986) "HRIS Build-Or-Buy Analysis," Personnel Administrator, July, 28-31.

(21) Walker, A. J. (1980) "The 10 Most Common Mistakes in Developing Computer-Based Personnel Systems," Personnel Administrator, July, 39-42.

(22) Walker, A. J. (1982) "The Newest Job In Personnel: Human Resources Data Administrator," Personnel Journal, December, 924-928.

(23) Wolke, J. W. (1987) "Software Success: A Proactive Plan of Attack," Personnel Administrator, 26-30.

(24) Zientra, M. (1983) "New Job Links DP, Personnel Management," Computerworld, 17(9), 8.

HUMAN FACTORS IN SOFTWARE DESIGN: A COURSE TO MEET EMERGING NEEDS

Barbara Beccue
Carol Chrisman

Applied Computer Science Department
Illinois State University
Normal, Illinois 61761

Abstract

Ease of use is rivaling ease of maintenance as the most desirable characteristic of a system. Systems developers need to be educated to consider human factors when designing a system. This paper discusses a course on human factors in software design and suggests appropriate learning activities.

Introduction

The importance of human factors in software design is increasing as a result of the large numbers of people using computers to accomplish their jobs. These people have a greater variety of backgrounds and less computer experience than earlier computer users had. Consequently, applications need to be easy to use. Frequently, application systems have not been fully utilized because they are difficult to use. Typical complaints about a system's shortcomings include: the system is hard to understand, instructions are missing or don't make sense, it takes longer to do the job using the computer, the system doesn't give the needed information.

Human factors is one of the terms used to represent the study of factors that make a system work well in human terms. The application of human factors engineering or ergonomics to the creation of software involves all aspects of the user interface [2]. The interface is everything that is involved as the user interacts with the computer system. Software, hardware, task procedures, documentation, and training materials are all included. System designers need to find a satisfactory match between users and their tasks, equipment and environment. In order to do this, systems designers need to know the underlying concepts of human factors engineering, to be able to apply the concepts during the systems development process, and to be able to evaluate the success of the resulting design.

The systems development process has often ignored the need to consider human factors. Education and training of the systems developer has stressed techniques for developing flexible systems which meet functional requirements and are easy to maintain. Systems developers must adjust their orientation to include

consideration of human factors. They need to emphasize design characteristics which will enable the end user to better work with the machine.

Most Information Systems curriculum do not currently include the study of human factors. This paper describes a senior level elective course on human factors that can be added to the Information Systems curriculum. The material is also appropriate for a graduate course where some topics can be covered in more depth.

Course Description

This course is intended to provide students with an overview of human factors issues. The application of human factors to the design of information systems draws from an interdisciplinary field of study. Psychology, computer science, education, physiology, and engineering are some of the fields which contribute to the study and knowledge about human factors. Using a conceptual framework for why some systems work better than others, practical design guidelines can be introduced. How to incorporate human factor considerations into each phase of the systems development life cycle is discussed. Students learn to apply human factors in the systems development process to create systems that people can and will want to use.

Specific objectives for this course include the following. A student should

1. understand the human factors which impact the use of a software system and be able to apply human factors design criteria during systems development to improve user/system interaction
2. be able to evaluate the quality of a human- computer interface and the user aids for a specific system and be able to re-commend human factors improvements.

Course Content

The following content outline is used:

I. Introduction

- Concepts and terminology
- History of human factors
- Why human factors?

II. Understanding human/system interaction

- Human limits and differences: concept of limits, Physical limits and differences
- Psychological concepts and human performance: Cognition, Perception Motivation, Memory

III. Incorporating human factors in the systems development life cycle

- Analysis phase
- Design phase
- Implementation phase
- Installation phase
- Review & evaluation phase
- Maintenance phase: improving existing systems through inclusion of human factor considerations

IV. Designing Systems

- Task analysis and work modules

- Human/computer interface: Styles, Guidelines
- Screen design: Prompts, Dialogue
- Error messages
- Input/output form design
- Support material: Tutorials, Manuals, Training, Help function

V. Assessment and evaluation of existing or proposed systems

- Iterative design
- Use of prototypes
- User involvement
- Quantitative evaluations

The selected text for this course is The Human Factor by Richard Rubinstein and Harry Hersh [3]. Two texts being used for additional readings are: Human Performance Engineering by Robert W. Bailey [1] and Designing the User Interface by Ben Shneiderman [4]. For a graduate level course, the book by Bailey is a more appropriate text.

Student Activities

There are a number of possible activities which can be used to provide students with appropriate learning experiences. The activities should provide the students with opportunities to apply the concepts, guidelines, and methodology taught in the course. The type of activities will depend on the variety of software available and the experience students have.

An early activity is needed to create an awareness of human factor considerations. We have demonstrated 1 or 2 packages in class, and had the students discuss the ease of use of each package. Students can point out features that make the user's work easier and features that hinder, frustrate or confuse the user.

This activity can be followed by an assignment that has students practice evaluating a specific software system. Students can submit a written report on the quality of the system's interface. If the class is small, we have each student select a package to demonstrate and lead an in-class discussion and evaluation.

Another practical activity is to have students make recommendations for improving an existing system. For example, we had students examine systems developed as projects by a database class. Since the developers in the database class were inexperienced and had not received formal training in human factors, they had made many typical mistakes. It made a good activity since the students were able to apply guidelines discussed in the class and were successful in finding improvements to suggest. An even better activity is to have the students prototype their improvements and get the users' reaction to the revised system. This prototyping activity has been used with a few students and was quite successful.

Activities are needed that help students gain insight into the problems experienced by an end user. It is difficult to structure appropriate activities since the typical student has so much more computer experience than the typical end user. This

semester, students are evaluating various CASE tools. Actually, they represent fairly typical end users for this type of package. Most of the students are seeing the package for the first time but they are familiar with data flow diagrams and data dictionaries, the part of the tools they are using. Help features, tutorials, and other supporting materials designed for the new user are being examined in addition to evaluating the system interface.

Another possible activity is writing a user's guide or tutorial. Students can have someone else try following their guide or tutorial and provide feedback. The students in the class can exchange and evaluate the support items.

Possible activities that do not involve hands-on use of software systems can also be used. We have had students locate recent journal/trade articles on human factors and write a summary of an article. article. We prefer the work with software packages since students are more actively involved in the learning and class participation is thereby improved.

Summary

The study of human factors provides a framework for systems designers to use in building systems that are more satisfying for the end users. The success of an information system is increasingly determined by the ease with which users can interact with the system. With the recognition of the importance of human factors, there is an increasing awareness of the need for systems developers to have a basic knowledge about human factors. The course discussed in this paper offers one way to meet this need.

Students taking the course will learn the conceptual framework necessary to understand the human factors issues of information systems. They will also learn to evaluate proposed or existing systems in terms of human factors. They will be able to design and develop higher quality systems by including human factor considerations in the development process.

References

[1] Bailey, Robert W., Human Performance Engineering: A Guide for System Designers, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[2] Gasen, Jean, "Human Factors Design in Information Systems: Curriculum and Teaching Issues", ISECON Proceedings, San Francisco, 1987, pp. 341-344.

[3] Potosnak, Kathleen, "Creating Software That People Can and Will Use", IEEE SOFTWARE 4:5 (Sept. 1987), pp. 86-87.

[4] Rubinstein, Richard and Harry Hersh , The Human Factor: Designing Computer Systems for People, Digital Press, Burlington, MA, 1984.

[5] Shneiderman, Ben, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley, Reading, MA, 1986.

COMPUTER CRIME: SECURITY AND CONTROL

By Carol S. Chapman, MBA
Assistant Professor
Business Computer Programming
Haywood Community College
Clyde, NC 28721

ABSTRACT

Information is power. Constructive use of information opens great potential for improvements and benefits. The fact that millions of computers exist establishes that they are wanted and that they are serving the needs of people and organizations. Used properly computers are powerful, valuable tools. However, computers can be and have been misused. In the hands of a dishonest person with special knowledge about sensitive systems, the computer can become a powerful, hard-to-detect burglar tool. The news media has detailed many crimes which involved computers. Experts fear that reported cases make up only a small portion of overall computer crime. Security and control measures are essential to the protection of vital information--vital to power.

INTRODUCTION

In excess of ten million personal computers are being used in American businesses, with approximately 75% of them in small and medium-sized businesses. The transformation from manual systems to personal computer systems opens new vistas for owners and managers, but it also causes problems. Managers often find information more readily available but with a lessening of direct control over company procedures. Expensive knowledge and data is at risk from within the company and from external forces.

Within the company, data is at risk from employees who have growing access to important data. Owners and managers, being confronted by a new and powerful technology, are less able than they once were to understand and audit the actions of subordinate end-users who are masters of their electronic environment. A 1985 Data Processing Management Association survey showed that only 65% of the surveyed companies had a budget for data security. Small businesses could be betting the entire business future on the security of their computer system, but awareness of the risks in this new environment is in itself a big advantage. Small business often has an edge over big business in dealing with white collar crime because the small business manager knows the employees better and can be more able to spot potential computer abuses.

The scope of this paper begins with a brief discussion of the types of computer crime. Considerations in security, the prevention of computer crime, are followed by a detailed analysis of the major forms of computer controls. These controls include physical and data security controls, electronic data processing controls, management and organizational controls, data center resource controls, input, output, and processing controls, and employee controls.

TYPES OF COMPUTER CRIME

Computer crime is a criminal act that is accomplished using a computer or that poses a threat to those who use computers.(3) Most computer crimes fall into one of five categories. Manipulation of computer input involves redirecting input data to direct events and people°s actions. Changing computer programs is usually committed by computer professionals to cover up the crimes committed using the computer. A

third type of computer crime is to steal data. Financial data, customer information, or special designs or plans can be stolen from computer files. Fourth, computer time can be stolen. Computer time is money. The criminal either uses the time or sells it to others. Using computer communications to transmit unauthorized data is an example. Finally, computer programs can be stolen. In this crime, people make illegal copies of computer software instead of purchasing their own version. Since theft of computer data and software often involves making copies, the original remains intact. As a result, this type of computer crime often is difficult to detect.(2)

Several warning signals may indicate that the potential for computer crime exists. These indicate poor control of the computer system or a lack of education on the part of the users. Some common signals include:

1. The computer seems to run the company; management just reacts.
2. Management expects computers to solve major existing problems.
3. Management and users do not (cannot) communicate with the computer professionals.
4. Users are told how their systems will be designed.
5. Documentation of system design and use is incomplete.
6. Decision makers are active with programming and trouble shooting.
7. Computer professionals are given no guidelines within which to work.
8. Access to data and software facilities is easy and uncontrolled.
9. Errors in processing occur frequently but without adequate investigation.
10. People are not held specifically accountable for the system operations.
11. Management fails to implement audit recommendations.
12. No EDP audits are performed.(5)

PREVENTING COMPUTER CRIME-SECURITY

There is no such thing as a completely secure data processing installation. First, completely secure computers are not provided by computer manufacturers. Once an ingenious programmer has found a way to modify the operating system, computer security features such as passwords and account numbers are ineffective.

Second, keeping up with the business and with changes in computer technology involve so much energy that time is not taken to adequately consider computer security. Computer inputs are not well controlled. Accuracy and completeness of computer outputs are not checked. Security issues are often only superficially considered when systems are designed or when programs are written due to the notion or hope that computer crime will not happen to their business.

Finally, effective security can be costly. Time and resources are spent to build a secure system. Security features often make the system more expensive to operate by requiring verification of input and output. Good computer security also means that programs operate more slowly to accommodate instructions of security functions.

Companies must strike a balance between no security and near-complete security. The cost of security should be weighed against the price of potential losses. Security needs should be defined according to the files and the equipment they protect.(2)

COMPUTER CONTROLS

To protect data and equipment, rights of access must be defined, then enforced. Several areas of control should be addressed.

PHYSICAL CONTROLS

The physical protection of equipment can be difficult. Computer equipment can be located in windowless rooms to deter unauthorized entrance. The equipment can be bolted down. Each piece might have a unique serial number to allow for easy identification of

misplaced or stolen items. Special locks and entrance restrictions can be installed to limit physical access to a computer system. Physical security requires more than control over unauthorized access to a facility. Disasters such as fires, floods, and earthquakes are major threats that occur unpredictably but require preparation and readiness. Protective and recovery measures must be in place including arrangement of equipment, danger detectors, and protection which will not destroy the electronic equipment. Floppy disks and hard disks are relatively safe from unauthorized access when locked in a vault.(5)

DATA SECURITY CONTROLS

Logical measure must be taken to ensure that sensitive data are kept confidential. Data security takes place after physical security measures have been established. Many software houses add copy-protect schemes to their programs to prevent an operating system°s utility programs from making a copy of the disk. Data encryption is the act of scrambling data and programs stored on disk so that a copy of the data is unintelligible without a decoding program.

Another type of logical control is a system password, indicating varying degrees of security clearance. Passwords and special accounts can separate and secure the work of individual users or a large system, but passwords are effective only if they are kept secret. Electronic security such as closed-circuit television and sound sensitive listening devices are effective measures.

Having a duplicate set of all vital tapes, software, and company data stored in a secure, isolated storage space accessible to only a few top data processing executives is a valuable security measure. Also, periodic security audits performed both internally and independently can help determine whether security is sufficient.(5)

ELECTRONIC DATA PROCESSING CONTROLS

The American Institute of Certified Public Accountants has recognized the possibility of computer crime and issued an official statement (called SAS-3) directing certified public accountants to pay special attention to business computer systems. Since computer crime is usually discovered by accident, computer professionals are faced with the challenge to design systems that will detect unauthorized computer use. Auditors and computer professionals have worked together to develop recommended procedures or controls over computer systems.(2)

MANAGEMENT CONTROLS

Management must be closely involved with data processing decisions. Senior managers should take an active part in the management of data processing functions. They should recognize the importance of data processing to the company. They should set the direction for, and be actively involved in, data processing plans. Awareness by top managers of the processing activities taking place can increase the security of a computer system.(2) Robert Campbell, president of the computer security consulting firm Advanced Information Management Inc., believes that between one and five percent of each Management Information System department°s budget should be devoted to security. The increase in networking alone has created many logic pathways which may be unknown, and this necessitates increases in spending for computer security. Management must not become complacent, even when there appears to be no security break. Systems must be implemented properly if they are to succeed.(1)

Senior managers can handle data processing in several ways. Managers should demonstrate appreciation for and interest in the data processing function. Showing interest includes occasional visits to the computer staff and recognition in verbal and written forms.

Senior managers can place the data processing function high in the organizational structure instead of burying it within accounting or finance. Management is more apt to pay attention when data processing is on a par with other business departments.

Senior managers can understand the company's vulnerability to computer crime. They can then communicate the importance of controls to the entire organization by being very positive about the need for controls.

Another responsibility for management is to form a steering committee made of senior management personnel. The committee is to receive reports about project status, provide general direction to the data processing staff, and provide go/no-go decisions for the stages of systems development.

Finally, management can take a role in data processing by requesting and paying attention to periodic operations reports. These allow management to know how well the computing resources are being used, how satisfied the users are with the data processing function, and where exists the major data processing problems. Increasing the amount of communication between data processing and senior management is the objective of these reports.(2)

ORGANIZATIONAL CONTROLS

Organizational controls concern the organizational structure of the company. Structure should separate authorization and duties. When no separation exists, employees can have unlimited access to the computer. Operations personnel and development personnel can provide checks and balances on each other. The series of checks and balances involves the authorization of one or more levels of management regarding performance duties.(2)

The operations group should control the equipment and the production program library. The development group should develop new programs in accordance with requirements. They should not have access to the tape library or to the production programs.

Only the programmers could develop program changes, and only the operations department could change the production library, which would require a supervisor's authorization.

DATA CENTER RESOURCE CONTROLS

Data center resource controls should be implemented. The use of computer equipment should be restricted to authorized personnel. Processing should be controlled by schedules, and records of use should be reviewed.

Computer operations should be controlled with the aid of scrupulous record keeping. Processing needs should be accommodated within a precise schedule. Procedures and job schedules need to be documented and verified. Supervisors must examine operations to verify that procedures are followed. Records of computer activity need to be reviewed regularly. Schedules and procedures must be stable so that operators are not able to change them easily and frequently.

Access to the computer must be controlled. Only authorized personnel should be allowed in the computer room. Equipment use also needs to be recorded to ensure that only authorized personnel gain access to computer tools. These measures not only protect the equipment from damage but also help to ensure that outputs are delivered only to the right people. Limiting access to the computer room reduces traffic, allowing for a quieter working environment and helping to eliminate operator errors.

In addition to protecting computing resources during normal operations, plans and procedures must be adopted to recover from potential problems. A recovery plan specifies measures to be applied if data resources are damaged or destroyed. The use of backup files and rented computers in off-premise locations are types of recovery systems. Recovery procedures should be well documented. Staff should be well trained in recovery system use and the creation of backup file copies.(5)

INPUT, PROCESSING, AND OUTPUT CONTROLS

The responsibility of placing controls on input, processing, and output belongs to management. Input controls affect data before it is entered into the computer. A standard form for input data needs to be established. Inputs presented in improper form must be rejected. Control totals should be used where appropriate. Control totals are calculations made independently of the computer system. These totals are compared against the outputs to double-check the accuracy of figures produced by the computer system. Users should be trained to record control totals effectively and use them properly as an important precaution.

Teleprocessing applications input is harder to control. Passwords and account numbers can be used to cause the system to accept only certain input from designated users or locations. The system can be fooled by a user who has gained unauthorized access to a password or account number. Restricting the hours of a computer terminal's use is an added control to prevent fooling the system. Users should change their passwords periodically as an added control.

Documentation is important to controls over the processing of data. All operations procedures and schedules should be clearly documented and followed in order to achieve effective security of a computer system. The performance of the operators should be monitored periodically by the supervisor to ensure that the procedures and schedules are being followed.

Online systems pose special security challenges because transactions often can be difficult to trace. An activity log is used to record any change to the system. This summary of online activity is used to correct errors or to assist auditors in reviewing records.

The operations department should keep records of all processing errors and system failures. Each correction requires documentation. These records should be reviewed periodically by data processing supervisory personnel to trace unauthorized activity, to determine the nature of system failures, and to assess other problems. The records can be a tool in assessing the need for additional training and to evaluate employee performance.

Finally, the output of all data processing activities should be controlled. Procedures for distributing output must be documented and followed. Only authorized users should have access to output, which they need to examine for completeness and accuracy. Output control totals should be reconciled against input control totals.(5)

EMPLOYEE CONTROLS

Computer crime is difficult to detect in a large company because the criminal usually appears to be a near perfect employee. A profile of the computer criminal indicates a young and ambitious person with impressive educational credentials. The person may be employed as a technician, programmer, manager, or high-ranking executive. The expertise of this person causes detection of crime to be difficult and often accidental.

To aid in preventing computer crime, in addition to a well-trained security force, the screening and selection procedures of potential employees should be examined closely and performed carefully. Any violation of company rules by an employee should lead to action by management, and possibly to discharge of the employee.

Although companies can implement many security measures to protect computers and funds, often the security depends upon the ethics of individuals. Computer ethics refers to the standard of moral conduct in computer use. Ethics can govern an individual's attitude toward use of data, behavior on the job, and the copying of software.

Matters regarding ethics often involve hackers. A hacker is a computer enthusiast who feels a challenge to

break security measures and to gain unauthorized access to a computer system. Hackers can erase valuable data or cause a system malfunction or breakdown, even if the hacker intends no harm.(3)

Employees often use a company's computer and software for personal use. Companies should adopt a standard of behavior or written plan to set limits on employee use of company resources. The plan can only be as effective, however, as the ethics of each employee. Only through ethical behavior will the ultimate security and privacy of computers and computer data be assured.(4)

Another problem requiring control is the software piracy, the illegal copying of software. Legally defining the boundaries regarding copyright violation is difficult. As laws are enacted defining illegal copying and establishing penalties for violations, companies must develop policies in accordance with the laws to control employee activity. Once again, ethics plays an important role in control of software piracy.(3)

Finally, control measures should be instituted regarding terminated employees. These employees have had access to company data which may be sold to competitors or destroyed prior to the exit of the employee. Some controls which can be implemented include changing the locks on doors and equipment and asking the employee to return keys and equipment. Physically checking the desk and file cabinets of employees prior to their exit can prevent crime. System passwords can be changed to disengage the exiting employee's password and assign new passwords to remaining employees. An experienced programmer and an auditor can investigate the system to uncover any destructive devices planted within the programs and data by the terminated employee.(1)

SUMMARY

Essentially, there are five categories of computer crime:(1) manipulation of computer input;(2) changing computer programs;(3) theft of data;(4) theft of computer time; and (5) theft of computer programs. Several warning signals can be observed which may indicate the potential for computer crimes.

Preventing computer crime is difficult due to the lack of existence of completely secure data processing installations. A manager often does not have time to adequately consider computer security while just keeping up with business and technological changes. Also, computer security can be costly. Companies must determine their level of security based upon the data and equipment which must be protected.

Management should consider several areas or types of computer control. Physical controls involve the protection of equipment from unauthorized access and from physical disasters. Data security controls are logical measures which must be taken to ensure confidentiality of sensitive data. Copy-protection, data-encryption, duplicate data stored off-site, and passwords are data protection devices.

Electronic Data Processing controls cover the procedures of computer control in the various areas of business operations. Between one and five percent of the MIS budget should be dedicated to security. Management controls are dependent upon awareness by top managers of the processing activities. Management's appreciation for and interest in data processing can be expressed by placing the data processing function high in the organizational structure. Senior managers must also understand the company's vulnerability to computer crime and communicate the importance of computer controls to the organization.

Management should form a steering committee to provide general direction to the data processing staff and to guide systems development. The

committee should request and pay attention to periodic operations reports to evaluate the functioning of the data processing department and increase communication between management and the department.

Organizational controls include the separation of authorization and duties in order to limit access to the computer and provide checks and balances within the system. The checks and balances involve authorization by one or more levels of management regarding performance of duties.

Data center resource controls involve processing controlled by schedules and review of use records. Supervisors must verify that procedures are followed by regularly reviewing records of computer activity. Recording equipment use protects the equipment, ensures that output is distributed properly, and that equipment access is limited to authorized users.

Problem recovery procedures should be established for the possibility of data resource damage or destruction. A well-trained staff following well-documented procedures will help ensure data protection.

Implementation of input, processing, and output controls is the responsibility of management. Established standards for input data must cause rejection of data in improper form. Control of input to teleprocessing applications attempts to keep the system from being fooled into accepting improper users. Changing passwords and restricting terminal use hours are effective control measures.

Documentation is important to any controls system. Operations procedures which are clearly documented and followed achieve the greatest security. Periodic review of operators will determine if schedules and procedures are followed. Activity logs are used to record on line activity and to assist with error correction and audits.

All processing errors and system failures must be documented and reviewed in order to determine the source and nature of the errors or failures. Output controls address the issues of proper output generation reconciled against input control totals.

Employee controls involve screening potential employees and being aware of the profile of the computer criminal. Hackers wish to gain access to a system simply to prove it can be done. Company computers and software are often used for an employee's personal use. The illegal copying, or piracy, of software is an employee problem which must be controlled. A company policy regarding employee computer operation activity should be developed and enforced. Computer security ultimately depends upon the ethics of the users and a well-trained security force.

Finally, control measures are necessary regarding terminated employees. Employees who have been terminated may wish to damage or destroy data and equipment or to steal valuable data. Control measures include searching the employee's desk and file cabinets, deleting the employee's access code to the computer, changing access codes or programs, changing locks on doors, and having an experienced programmer check programs for "logic bombs."

CONCLUSION

Computers have the capacity to be a boom to humankind and to help solve humanity's greatest problems. However, they also have the capacity to be destructive, to limit personal freedom, and to eliminate personal privacy.

The greatest strength humans have is knowledge. As more people learn how to deal with computers, they will exercise more power as consumers. They will learn to observe the quality of computer service and will choose to do business with companies that have secure systems. As time passes, competition in the marketplace will eliminate companies that operate sub-standard computer systems.

Computer abuse and its prevention center around individual privacy,

computer crime, and computer system security. The impact of computers can best be controlled through legislation and the education of consumers. Any organization can be victimized by computer crime. Most organizations are reluctant to prosecute due to the negative publicity. Many crimes go undetected since most are discovered only by accident.

All computer systems can benefit from physical controls and data security. Organizations with large computer systems need EDP controls. All components of a computer system must be protected from computer crime. Access to hardware and software must be limited to qualified people. Data must be complete, accurate, and monitored for illegal changes. Procedures must be developed to help people operate and control the system properly.

Only time will tell whether the benefits the computer offers will outweigh the risks it poses for life in the Information Age. However much humans may fear the risks of computing, no alternative exists to widespread public use of computer technology today. Massive increases in population, societal complexity, and information require that people adopt computer-based knowledge processing systems. To give up computers would mean resignation to many unsatisfactory outcomes. For example, most banks and corporations could function only three or four days without computerized information systems.

No technology is intrinsically good or bad, beneficial or risky, favorable or unfavorable. A good technology can be put to bad, risky, or unfavorable use by people. The outcome of history will not be the victory of a "good" technology over a "bad" one. The outcome will have much to do with the ethics, integrity, lawfulness, knowledge, and vision of the people who use technology.

REFERENCES

(1)Altman, J. "MIS Computer Security Slacking." MIS Week, 21 September 1987, p. 1, p. 13.

(2)Kroenke, D. M., Dolan, K. A. Business Computer Systems (3rd ed) Santa Cruz: Mitchell, 1987, pp. 494-509.

(3)Mandell, S. L., Hopper, G. M. Understanding Computers (2nd ed). New York: West, 1987, pp. 355-359.

(4)Pfaffenberger, B. Personal Computer Applications. Boston: Little, Brown, 1987, pp. 452-454.

(5)Trainor, T. N., Krasnewich, D. COMPUTERS! Santa Cruz: Mitchell, 1987, pp. 357-368.

TOWARD A DESCRIPTIVE THEORY OF INFORMATION SYSTEMS

John T. Overbey, PhD, CPA
Department of Accounting and Information Systems
Western Carolina University
Cullowhee, NC 28723

Raymond G. Crepeau, MBA
Department of Accounting and Information Systems
Western Carolina University
Cullowhee, NC 28723

Human beings have had information systems from the beginning of our existence. It is only lately however that information systems per se have become a field of academic and organizational study. Today we hear the phrases "information age", "information as a resource", "information-based economy" etc., everyday. In a social science such as accounting or information systems there are no "natural laws" to discover, hence the people in the field must develop a theoretical structure and explanation of the field to use as a guide in developing, evaluating, understanding and using information systems. This paper represents an attempt to develop some beginning theoretical concepts for discussion purposes.

TOWARD A DESCRIPTIVE THEORY OF INFORMATION SYSTEMS

Human beings have had information systems from the beginning of our existence. It is only lately however that information systems per se have become a field of academic and organizational study. Today we hear the phrases "information age", "information as a resource", "information-based economy" etc., everyday. Obviously the more important information becomes, the more important information systems become. In any field of endeavor such as physics, psychology or information systems it is helpful to have a defined theoretical structure as a point of reference. In a natural science such as physics experiments are conduced to discover the underlying natural laws. In a social science such as accounting or information systems there are no "natural laws" to discover, hence the people in the field must develop a theoretical structure and explanation of the field to use as a guide in developing, evaluating, understanding and using information systems. This paper represents an attempt to develop some beginning theoretical concepts for discussion purposes.

Websters New International Dictionary, Second Edition, Unabridged, defines theory as follows: "the body of generalizations and principles developed in association with practice in a field of activity (as medicine, music) and forming its content as an

intellectual discipline." A principle is defined as: "a comprehensive and fundamental law, doctrine or assumption on which others are based or from which they are derived". Thus in a field such as information systems, a theoretical structure can be developed from an analysis of practice in the field and principles thus distilled can be used as an explanation and foundation of the discipline.

In the past 30 years there have been several expositions of general systems theory, but what is attempted here is an effort at a descriptive theory of current practices. The principles herein suggested are in a form to be of practical use to students, practitioners, and others who wish to obtain a framework from which to understand current information systems.

PRINCIPLES

Eight principles are suggested:

(1) Component Principle: An information system is composed of the following components: input, processing, output, data base, controls, boundaries, and interfaces.

(2) Cost-Benefit Principle: An information system must pass the cost-benefit test or produce mandatory information in order to justify its existence.

(3) Information Flows Principle: The information flows of an information system must be able to be defined separately from the technology which transmits them.

(4) Structure Principle: Information systems are structured in three basic ways: (1) algorithmic or deterministic, (2) heuristic or rule of thumb, (3) a combination of both.

(5) Interface Principle: Information systems must be able to interface with supra-systems, sub-systems, and equivalent level systems.

(6) Ownership Principle: Information systems are owned by their users not their designers.

(7) Environment Principle: An information system is affected by its environment.

(8) Life Cycle Principle: Information systems go through a life cycle of analysis, design, and implementation use, maintenance, and modification; and termination.

COMPONENT PRINCIPLE

An information system is composed of the following components: input, processing, output, data base, controls, boundaries, environment, and interfaces. The irreducible elements of a system are input, processing, and output. As organizational and personal life become increasingly more complex so must the structure of information systems that serve people and organizations. For a particular processing run or query an information systems gets its data either from input or from its internal data base (Obviously at some prior point the data in the data base must have been input). Therefore it is necessary to include the environment in which the information system exists and its interfaces with other information systems as components as these are sources of input to the information system. To be manageable in scope and size an information system must have boundaries which separate it from other information systems, sub-systems, and supra-systems. For security of data and reliability of information an information system must have controls.

COST-BENEFIT PRINCIPLE

An information system must pass the cost-benefit test or produce mandatory

information in order to justify its existence. Information has a cost and a value. It is possible to divide information into two major categories: discretionary or optional and mandatory. If the government says all organizations of a certain type must file a certain report, then the information system must produce that report for its organization to stay in existence. Management must weigh the cost of other, i.e., discretionary or optional, information against its value to determine if it passes the cost-benefit test.

INFORMATION FLOWS PRINCIPLE

The information flows of an information system must be able to be defined separately from the technology which transmits them. A military example may help to clarify this principle. The basic information that an infantry general located behind the lines might wish to pass to his commanders on the line about attack plans (where, when, how many, etc.) has not changed much in the last several thousand years, but the technology used to communicate that information has obviously changed dramatically. From the user's standpoint it is the information that is paramount, not the supporting technology.

STRUCTURE PRINCIPLE

Information systems are structured in three basic ways: (1) algorithmic or deterministic, (2) heuristic or rule of thumb, (3) a combination of both. The older transaction processing oriented information systems fall into the first category. In these types of systems output was completely predictable because the processing was precisely and rigidly defined. These information systems might be referred to as solution oriented systems. Many of the newer Decision Support Systems and Expert Systems operate in a less rigidly structured and less predictable manner. These systems often employ "fuzzy logic", and can deal with incomplete or probablistic input. They may be looked on as goal oriented as opposed to solutions oriented.

INTERFACE PRINCIPLE

An information system must be able to interface with supra-systems, sub-systems and equivalent level information systems. The different individual information systems which go together to form the organization's total information system must be able to communicate with each other. This necessitated compatible hardware, software, data bases, etc.

OWNERSHIP PRINCIPLE

Information systems are owned by their users not their designers. In the older, traditional, larger systems, i.e., those designed by information system professional, two common problems were lack of participation by the user in design process and an attitude on the part of the systems analysts that the system was theirs. With the advent of end-user computing in many cases the user and designer are one and the same.

ENVIRONMENTAL PRINCIPLE

An information system is affected by its environment. The environment of an information system includes such things as the underlying technology, managerial and decision style of the users, the type of output desired (e.g. a transaction summary or information for a high level decision) the type of input available, etc.

LIFE CYCLE PRINCIPLE

Information systems go through a life cycle of analysis, design, and implementation; use, maintenance, and modification; and termination. So often the systems life cycle is thought of in terms of only the first part: analysis, design, and implementation,

but in reality often as much as 50% of the cost of the system is incurred in maintenance and modification. The life span of an information system is dependent upon such things as the rate of change in information technology, change in the organization, etc. Eventually it becomes too expensive to modify the old system for needed change and a new system must be begun.

OBJECTIVES

Webster defines an objective as: "Something toward which effort is directed" or "an aim or end of action". Thus information systems objectives can be used as a means of evaluating the quality of information systems. The following objectives are proposed for information systems.

An information system has the following objectives:

(1) To decrease the uncertainty of personal, organizational and societal life

(2) To meet internal and external information requirements

(3) To provide an additional strategic/competitive tool/weapon to management

(4) To produce information that is timely, reliable, relevant, and free from bias

SUMMARY

The information system field is one that is daily increasing in importance and is also achieving some degree of maturity. As such it needs a well developed theoretical framework. This paper has proposed eight principles and four objectives as a basis for discussion of a descriptive theory.

THE DANGERS OF NETWORKED SYSTEMS

James Perotti
Professor of MIS
Ohio University
Copeland Hall
Athens, Ohio 45701

ABSTRACT

Approximately 17 million PCs have been sold to businesses, an estimated 6 million more PCs will be sold in 1988; sales continue to increase each year. By the end of 1991, 90 percent of the managers and professionals will be using personal computers on a regular basis as a part of their daily work. The fastest growing segment of the computer market is networking hardware, hardware to link PCs together. The number of corporate sites with LANs will nearly double this year. Estimates are that LAN implentations will grow approximately 45% per year for the next three years. By 1991, many of the 33 million PCs will be part of a networked system. Although linking PCs together seems an innocent evolutionary step in the growth of end-user computing, many knowledgeable users and data processing managers understand that a networked system of PCs has enormous power and capacity. This new style of computing requires a concerted and immediate effort to manage it, because it represents a very real threat to organizational communications, hence organizational effectiveness.

WHY NETWORKS ARE RISKY

The advent of PC computing seemed more an irritant than a threat to those managing computer resources in large organizations. PCs were small, inexpensive and, when judged by mainframe standards, of limited usefulness. Since PCs were 'stand-alone' devices, users typically created and worked with their own data, hence PCs functioned like calculators and typewriters. Although linking PCs together seems an innocent evolutionary step in the growth of end-user computing, many knowledgeable users and data processing managers understand that a networked system of PCs has enormous power and benefit. This new style of computing requires a concerted and immediate effort to manage it, because it represents a very real threat to organizational communications.

Networked systems have become a major concern of MIS directors. According to a recent Forrester Research, Inc. study of Forture 1000 firms, network management is the top concern among MIS directors surveyed (1). A recent Datamation / Price Waterhouse Survey of 500 information executives identified decentralized communication systems as their second greatest concern (2). An earlier Datamation survey of Fortune 1000 MIS directors, "MIS Rates the Issues", confirms that PC style end-user computing is much less bothersome than departmental system growth:

> Three of the four issues that round out the top 10 suggest another

> grouping: integration of technologies, telecommunications technologies, and office automation ....A major surprise is the absence of end-user computing (ranked 12th) from the top group (3).

THE GROWTH OF LOCAL AREA NETWORKS

Approximately 15 million PCs have been sold to businesses and sales continue each year. Sprague and McNurlin suggest this possibility: "The end of the 1980s will find about 90 percent of the managers and professionals...using personal computers on a regular basis as a part of their daily work (4)."

The fastest growing segment of the computer market is networking hardware, hardware to link PCs together. "The number of corporate sites with LANs wll nearly double this year ... 45% of the sites surveyed plan to have installed at least one network by the ned of 1988--up from 27% a year ago (5)."The International Data Corporation (IDC), a leading market research firm, predicts that almost 50% of those PCs will be networked by 1990 (6). In 1986 110,580 networks were installed, 220,900 in 1987; IDC predicts that 385,000 networks will be installed in 1988, 526,000 in 1989, and 647,000 in 1990. Networking is a fast growing multi-billion dollar business which is obviously meeting the needs of businesses.

Business Week again featured networking in its Feb.1 issue using this title in its Information Processing section: "DEC's new plan, if you can't beat PCs, join 'em--to each other. President Ken Olsen of DEC strategy is there described:

> The next phase of his (Olsen's) battle with IBM will involve selling networks that can begin to link the nearly 15 million personal computers now used by business--both to each other and to larger computers (7).

New hardware purchases suggest that organizations are enthusiastic about networking technology. Networking hardware is a way of unifying stand-alone PCs, a way of making a single system out of an aggregation of computers, printers, and disk drives. At a deeper level, the networked system is a way of unifying the PC users, the managers and professionals, belonging to the work group. A new computing style occurs; it comes with the system. The largely independent PC user become part of a 'departmental system' computing style. The users, often purchasers, of the system expect four major benefits from their departmental resource processor (DRP):

> First, it provides access to a wide variety of corporate information. Second, DRP implementation parallels corporate structures that are based around departments. The departmental computer also satisfies a user's psychic need for independence. Finally, DRPs give middle management a more active role in technology management .... Today, micro users are beginning to demand integration--integration that enables their desktop machines to tap into corporate resources including databases, networks, and server-based software (8).

LOCAL AREA NETWORKS CREATE THE TOWER OF BABEL

MIS directors are concerned because each LAN can have the power of a large mainframe; each can evolve into a new free-standing data processing center. As each department optimizes its own internal information managment, the organization's multiple computing centers become less and less coordinated, less and less able to share information. The ultimate danger might be termed the "Tower of Babel" syndrome: incompatible departmental systems preventing communication between departments,

making it impossible to coordinate the tasks necessary to achieve the organization's mission.

The battle for control over computing will be fought over data, not hardware. It is a war in which everyone can lose, a war in which the organization could become a casualty. While an evolution into LANs could occur without conflict or attention, the inability (or unwillingness) to share information between departments will create conflicts and become recognized as a major barrier to coordinating intra-departmental tasks. The evolution into departmental computing could easily occur without integrative mechanisms to fuse the systems into one coherent shared information resource. Without top management intervention, planning, and coordination, the departmental systems could result in a "Tower of Babel".The irony is that improving departmental communications will worsen organization wide communications. Just as each department cannot be allowed to administer its own accounting data, so also each department cannot control data which must be shared across the organization.

Each department will optimize its design of the data to suit its own requirements, its own perception of how the organization operates. This uncoordinated data design will create a communications barrier; each department will automate using different vocabulary for the same objects. The result will be that data differences (different languages) will prevent departmental systems from communicating with one another. The common goals of the organization, in so far as they rely upon cooperative efforts from the departments, will come to naught.

But who will tell a department that it must follow data definitions used by another department? With the "dp general" having lost the war, who will now control the information resource? This is a data and people problem: the data is "locally owned" and controlled by independent departments; organizations are unprepared to coordinate the emergence of departmental systems. The fragmentation of information will be especially tragic because many organizations now realize the strategic necessity of unifying organizational members into a cohesive group.

There is a clear contradiction between PC style computing as user controlled, and the need to exercise better centralized control over organizational computing. Many organizations intially vested control over all computing with DP management; this originally appeared to be necessary because of concerns about incompatibility. The standardization of the hardware and operating system and the low price of the PC "commodity" make it difficult to justify the exercise of control over PC computing.

> For the most part, 1986's dp generals and their commanding officers are approaching the MIS challenge in one of two ways: they are either holding on to control, at least of the big systems, or they are turning the war over to the end users, letting them fight their own dp battles (9).

STRATEGIC USE OF NETWORKED SYSTEMS

PCs can exist and proliferate without much danger to the organization; LANs should be planned, policies should precede implementation. As with the Tower of Babel, planning the management information system and coordinating its implementation are essential steps to ensure that the construct all fits together. At the point when communication breaks down in the organization or the tower, it is too late for solutions; the end is close at hand.

Top management will have to become convinced of the need for an organization-wide integrated information system. Top management must then appoint a high level coordinator of information -- a Chief Information Officer (CIO). The CIO's

major role will be that of strategist or planner.

> The cio ... has corporate responsibility for information technology, policies, standards, procedures, and technical architectures. In many cases the cio is also in charge of telecommunications and sometimes even applications. More importantly, the cio acts as technology strategist for the organization (9).

The problem of incompatible networking hardware is much more easily solved than the formible task of managing the data. The organization needs a single hardware architecture, needs a way to ensure compatibilties between LANs and the mainframe. tworking hardware policies can be based upon the plan for a Wide Area Networking (WAN) strategy to link the Local Area Networks (LANs). An Ethernet WAN, such as those at Ohio University and Ohio State University, permit a wide variety of LANs to be tied into an organization-wide system, linking LANs to one another and to the IBM mainframes. The WAN will dictate which LANs are compatible, hence permissible, the LANs will then dictate which PCs are compatible and permissible.

Managing and integrating information, managing the data practices of PC users, is extremely difficult since end-users are autonomous and regard the data in their PC as owned by them. By extension, the data stored on a network server and shared with others is also regarded as departmental property. Centralized controls and data definitions need to be imposed on autonomous managers and professionals--no easy task. It is most easily achievable when the departmental managers see that the goal is to support the strategic direction of the organization. The departments cannot be forced to comply with a company directive. Organizational integration of people built upon a set of shared goals must precede the attempt to achieve an integrated organizational information system.

REFERENCES

(1) Dalrymple, R. (1988) LAN diagnostics. COMPUTERWORLD, XXII (February 22, 1988) 51-56.

(2) Statland, N. (1988). Datamation / Price Waterhouse Survey. DATAMATION, 34 (February 15, 1988)106-107.

(3) Herbert, M. and Hartog, C. (1986). MIS rates the issues. DATAMATION, 32 (November 15, 1986) 79-86.

(4) Sprague, R. and McNurlin, B. (1986) Information Systems in Practice. Englewood Cliffs, N.J: Prentice-Hall, 1986, 289-291.

(5)Musgrave, B. Stacking up LANs. DATAMATION, 34 (February 15, 1988) 56-64.

(6) Keefe, P. (1987) PC local-area networks. COMPUTERWORLD, XXI (November 30, 1987) 114.

(7) Helm, L. and Verity, J. DEC's new plan: if you can't beat PCs, join 'em- to each other. Business Week, 3036 (February 1,1988), 83-84.

(8) Colony, G. (1987) New ports of call. DATAMATION, 32 (November 15, 1986) 69-74.

(9) Winkler, C. (1986) Battling for new roles. DATAMATION, 32 (October 15, 1986), 82-88COMPUTERWORLD, XXI (November 30, 1987) 114.

SOFTWARE ACQUISITION OPTIONS AND EFFECTIVE IMPLEMENTATION: THE ROLE OF ORGANIZATIONAL CHANGE

Joanne E. Hurd and George M. Kasper
Texas Tech University

In the rush to install software, the critical importance of organizational change is often ignored. It is proposed that this results in unanticipated organizational change that may be responsible for ineffective software implementation. A model of the relationship between software and organizational change inputs is developed and the tradeoffs discussed. The effect of management's objective of improved efficiency and/or effectiveness is also explored. Finally, some suggestions for recognizing and dealing with the organizational change that results from software installation are given.

INTRODUCTION

The decision to implement an off-the-shelf package is often made based upon the software cost savings. However, a variable seldom included in the implementation analysis is the cost of organizational change.

The costs of implementing a computer-based application come from two related sources. The first, technology, includes hardware and software costs. The second is any change required of the organization in order to effectively utilize the software.

Tradeoffs exist between these two sources of cost. Organizations may choose to implement an off-the-shelf package due to economic constraints. However, when the procedural match is poor, the organization must change to effectively utilize the software [4]. The cost of this organizational change is seldom included in the software decision. This paper discusses the tradeoffs between software and organizational change costs. In selecting an off-the-shelf ready-made application, an organization must include in its cost/benefit analysis any organizational change cost that will be required to effectively implement the software application.

The purpose of this paper is to clarify the relationship between software acquisition costs and organizational change costs. Software acquisition costs increase as one moves from off-the-shelf systems to custom-built systems [14]. Inversely, the cost of organizational change increases as one moves from custom-built systems to off-the-shelf packages [6]. Therefore, a point exists at which the combined software and organizational change costs is minimized for a particular organization's application.

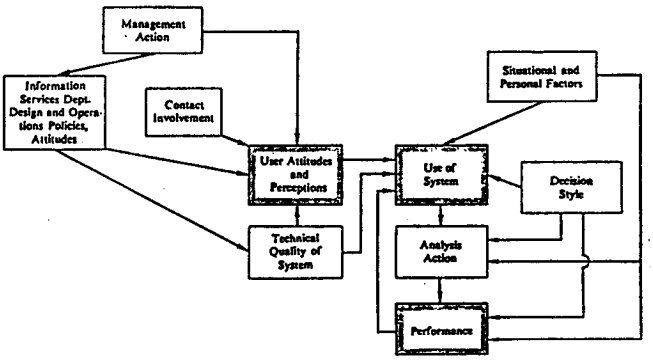
ORGANIZATIONAL CHANGE

Under the open system theory, an organization is viewed as a set of interrelated components that interact with each other and the organizational environment [7]. The three major components within an organization are: (1) people, (2) technology, and (3) structure. These three components are seen as highly interdependent. A change in one will either directly or indirectly change the other components. As a result, computer technology affects all the subsystems of an organization.

Lawler et al. present a model describing the interrelationships among the subsystems of an organization [8]. They propose six broad classes of related organizational variables. Technology, such as a software application, affects organizational effectiveness indirectly through attitudes, beliefs, and expectations, and through work behavior. This model is consistent with Lucas' work, which is presented here as Figure 1 [9].

Figure 1.

A Descriptive Model of Information Systems in the Context of the Organization



Lucas, H.C., Jr., Why Information Systems Fail, Columbia Press, N.Y. 1975, p. 20.

Lucas' model relates several categories of variables to the use and performance (effectiveness) of an information system. The technical quality of a system affects system performance through user attitudes and perceptions, and through use of the system. Both of these models suggest that any information system change will result in some organizational change, regardless of whether the system specifically meets the organization's needs or not. The intended change is usually an increase in organizational effectiveness or efficiency. However, the concern of this paper is organizational change that occurs solely to meet the requirements of the software, rather than to increase effectiveness.

MODELS OF ORGANIZATIONAL CHANGE

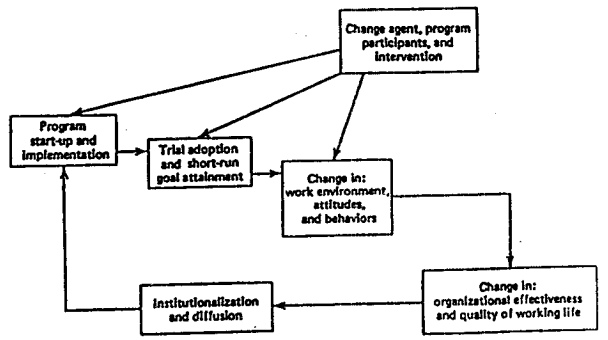
To better understand the effect of an

information system change on the organization, we must first understand how organizational change takes place. Two models of organizational change are the Lewin/Schein [2] and the Lawler et al. [8] models.

According to the dynamic Lewin/Schein model of change, change is a three-stage process. The first stage, unfreezing, requires developing a felt need for change and an atmosphere in which organizational members are open to change. The second stage, changing, occurs when a system change is actually introduced. In the case of an information system,changing involves the actual installation of the application. The final stage is refreezing. This is the process of institutionalizing the change. During this stage, the new application and procedures are integrated into existing organizational operations and relationships.

Lawler et al. expanded this model to include trial adoption and the influences of the change agent, program participants, and the nature of the intervention itself. Shown as Figure 2, the first stage of this model, program start-up, corresponds to Lewin/Schein's unfreezing stage. This is followed by the trial adoption of the changes by pilot units and, over time, the achievement of the projects' immediate objectives. The third and fourth stages of this model correspond to the changing phase of the Lewin/Schein model. Lawler's model reflects the fact that changes in work behavior must precede changes in organizational effectiveness. The fifth and final stage is institutionalization of the change, corresponding to the refreezing stage discussed above.

Figure 2.



Lawler III, E. E., Nadler, D. A., and Mirvis, P.H. "Organizational Change and the Conduct of Assessment Research," in Assessing Organizational Change edited by Stanley Seashore et al., John Wiley and Sons, Inc., New York, 1983, p. 26.

For the purposes of this discussion, it is assumed that the organization has progressed through the unfreezing or start-up phase. Our concern is with the actual change process (stage 2 of Lewis/-Schein and stages 2 - 4 of Figure 2), and in particular the costs associated with organizational changes required to effectively implement the application software.

COSTS OF ORGANIZATIONAL CHANGE

After the implementation of an organizational change program such as an information system, the primary question is whether or not the ends justify the means [12]. Has the introduction of the system had the desired results at a reasonable cost? However, several problems exist with answering this question. The first is that intangible costs and benefits, such as changes in employee well-being or cooperation, are difficult to measure. The second is that the existing evidence of the economics of change programs is sparse [12]. A review of the organizational change and work design literature shows little effort devoted to assessing the effects of organizational change in terms of economic indicators such as productivity [3]. An exception to this is Mirvis and Lawler's methodology for assessing the cost of organizational change [11].

Mirvis and Lawler developed a methodology for assessing the relationship between an organizational change program and attitudes, and tracing the effect of attitudinal changes on behavior and its financial consequences. Their methodology is based on the assumption that attitudinal measures are indicators of subsequent behavior, which in turn have economic implications for the organization. This relationship between organizational change and attitudes is consistent with that of Lawler et al. and Lucas (Figure 1) and empirical evidence [16]. In general, their methodology entails (1) correlating relevant job attitudes (such as satisfaction) with an employee's behavior, and (2) assigning behavioral costs through cost accounting techniques (detailed in 10). These are then combined to determine the cost per employee associated with measured levels of work attitudes.

SOFTWARE ACQUISITION

In general, there are three approaches to the acquisition of software. The first, off-the-shelf software, is an existing program or series of programs that a company licenses from a software vendor. The second, custom built software, is specially developed for a particular company's applications and operations, either by an in-house programming staff or a contractor. The third approach combines the previous two, incorporating custom modifications in an existing off-the-shelf package. This last approach retains many of the cost and quality advantages of off-the-shelf software and yet allows adjustments and tailoring to meet a specific company's individual needs [13]. Collectively, these options may be described as buy, make, and help, respectively. The applicability of these approaches is generally limited by the design of any already developed software and the uniqueness of the problem to be

automated.
When implementing custom built software, systems development is seen as a process of designing a tool to meet an organizations' needs. The primary tasks are to understand and define a particular need, and to design a system to meet that need.

When implementing an off-the-shelf application, the emphasis of systems development seems to shift dramatically. Tailoring of the software to meet the organization's needs is limited to the system's flexibility. Instead of centering on needs analysis, "analysis and design" seems primarily concerned with adapting the organization to fit the requirements of the new software. Since the programming phase of purchased software is almost totally complete, the danger is that in the rush to automate, managers may not spend the time needed to do a proper job on the early phases of the system life cycle; that is, feasibility study, system analysis, and system design. This may result in a poor fit between the organization's needs and the software capabilities. In turn, this results in unnecessary pressure being placed upon the organization to adapt to the software's requirements, as opposed to the software supporting the organization.

TRADEOFFS

It is almost a dictum that the cost of purchasing an off-the-shelf software package is much less than that of constructing the same code from scratch. Because of the vendor's economies-to-scale, purchased software may result in savings of as much as ninety percent when compared to the cost of producing the same code in-house [14].

From the perspective of securing software, the cost of acquiring and testing the actual code is most certainly reduced when it can be purchased from an outside source. However, securing computer code is not the objective, but rather a means in the development of a solution. While the actual computer code may cost less, ready-made software may not result in the most effective and/or efficient approach from the overall organization's perspective. Both suppliers and users of manufacturing-oriented systems agree that off-the-shelf software will at best meet eighty percent of a given user's requirements [12]. Presumably, spanning the remaining twenty or more percent requires organizational change, software modification, or both.

When implementing off-the-shelf systems, an organization enters the systems development lifecycle at the final stages. Therefore, early detection of required changes is not possible. Examining in-house made-to-order software, Boehm has shown that the cost of correcting software deficiencies increases geometrically as the project moves through the systems development lifecycle [1]. He showed that the cost of correcting an error during operations may be as much as one hundred times that of correcting it at the requirements definition stage.

PROBLEM STATEMENT

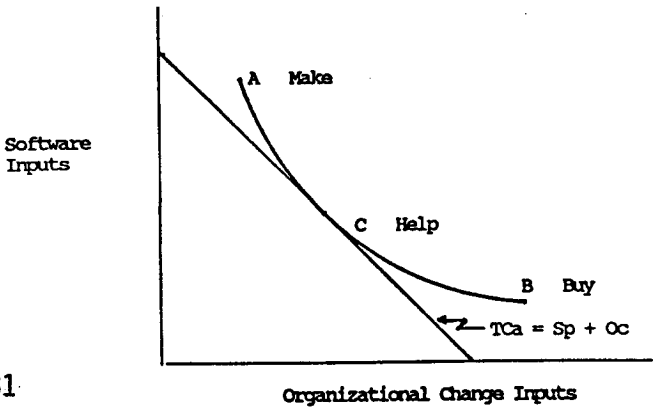
Because of the interrelationships of organizational components, all information system implementations result in changes to the organization [17]. While the desired change is usually an increase in organizational effectiveness, some organizational change may result from the lack of fit between the organization's needs and the software's capabilities. It is this change resulting from the software's requirements that is typically counter- productive to organizational effectiveness.

Off-the-shelf programs have their limitations. They are necessarily written to satisfy the needs of a broad range of similar, but not identical, organizations. This means that there are compromises and assumptions made so that there will be very few occasions when an off-the-shelf package will meet all the requirements of any single firm. In addition, it is the cost of modifying the software or, as is more often the case, changing the organization to fit the software, which may far exceed any savings resulting from the purchase of a generic package. Once the decision to automate a process has been made, alternatives must be evaluated from both a software life cycle and an organizational change perspective.

A PROPOSED RELATIONSHIP

From the organization's perspective, the cost of a computer-based application is the cost of the related software inputs and organizational change inputs needed to produce the desired output. Figure 3 shows the software and organizational change input substitution needed to achieve a desired outcome.

Figure 3.
For a Given Output,
Tradeoff Between
Software Inputs
and
Organizational Change Inputs



The ordinate of Figure 3 is the amount of software (whether to make, help, or buy; assuming appropriate hardware), the abscissa is the amount of organizational change, and the relationship shows the rate of substitution between software and organizational change to implement an application. The labels "make","help", and "buy" serve to identify points on a continuum ranging from software which exactly meets the organization's effectiveness specifications to that which potentially offers a poor fit.

At one extreme (A), the software is custom built, requiring the maximum amount of software input, yet the organizational change needed to produce the desired level of effectiveness should be minimized by the software's custom-made specifications. At the other extreme (B), the software is an off-the-shelf product, requiring little in the way of software input, however, an extensive change in organization practices and procedures may be necessary to achieve the desired level of effectiveness. This latter cost may more than offset any savings resulting from the purchase of a package.

The relative cost of software and organizational change inputs is needed to determine the appropriate combination needed to produce an effective result. Denoting the quantity of software and organizational change by S and O, respectively, and their unit prices by p and c, the total cost of an application is TCa = Sp + Oc. The point labeled C on Figure 3 shows the software and organizational change combination that minimizes the cost of producing a desired level of organizational effectiveness.

MANAGEMENT OBJECTIVES

One might expect management objectives to affect the substitution relationship between software and organizational change inputs. In general, the actions of management may be categorized as: those intended to result in more efficient operations, those intended to result in more effective operations, and those intended to produce both more efficient and more effective operations. However, despite management's objectives, the organization and its people will be required to fill the gap between existing practices and procedures and those required to implement the acquired computer code and produce the desired outcome. The greater the mismatch between the acquired software and that needed to produce the desired outcome, the more the organization must adapt.

In terms of Figure 3, different management objectives would likely result in a change in the relative substitution of software and organizational change inputs. For example, we would expect software inputs to be primarily associated with objectives concerning improved efficiency. Conversely, changes in effectiveness would seem to require proportionately more organizational change. This would result in a change in the rate of substitution between software and organizational change inputs. This would also result in a different cost minimizing combination.

DISCUSSION AND RECOMMENDATIONS

Based upon the preceding discussion and proposed relationship, recommendations for researchers and practitioners are discussed below.

The low cost of off-the-shelf applications is an important consideration in software selection. However, this must be balanced by considering undesired organizational changes that may accompany the use of an information system that does not fully meet the needs of an organization. To encourage the inclusion of organizational change costs in the application selection process requires a detailed analysis of the organizational implementation of the software. The tendency to purchase off-the-shelf applications must be resisted until a proper assessment is conducted. Without this assessment, the bias toward ready-made software solutions will result in overestimating the benefits and cost savings of off-the-shelf software and underestimating or totally ignoring the cost of organizational change.

To account for organizational change, project management must be made responsible for the total cost of an application including organizational change expenses. To encourage a proper accounting of these costs it is suggested that: (1) organizational change personnel be included in the project management team [2] and (2) that the steps in the software system life cycle preceding code generation be completed prior to, and independent of, the software acquisition (make, help, or buy) decision. The procedure for selecting software should be based on defining goals from the original suggestion through the feasibility study, system analysis, and logical design phases of the project. These goals can then be used as a standard for evaluating software acquisition alternatives.

The evaluation should also proceed from an application implementation perspective. In general, there are two approaches to evaluation. One approach involves comparing the capabilities of the alternatives being considered on an attribute-by-attribute basis with those of the goals. The second approach encourages comparing alternatives as total systems. This latter methodology recognizes that a system may be more (or less) than the sum of its component parts; that systems, and not components,

are implemented; and that while a particular property of the goal system may not be critical, per se, when combined with other marginal properties it may make the difference between success and failure. Examples of this approach can be found in Stabell [15] and Grudnitski [5].

Both the intent and perspective of management must be clearly defined before organizational change can be evaluated properly. This requires identifying the intended outcome(s) of the project as well as the major and critical stakeholders. In addition to actual users of the system, the latter includes management, data processing personnel, people external to the organization, and others. Obviously, if management's objectives are unclear, no informed third-party evaluation can be conducted.

Once improved efficiency, effectiveness, or both have been agreed upon as the intended outcome of the project, factors which motivate employees to achieve this objective must be established. The establishment of employee motivators is critical to the success of the project.

SUMMARY

This paper has highlighted the critical importance of issues often ignored in the rush to install software. It is suggested that the evaluation of any application software must recognize the inevitable organizational change. It is also proposed that the often unrecognized cost of organizational change is directly related to the mismatch between what the software code can do and the intended outcome of the implementation. It is the organizational, through its people, that must bridge this gap.

To increase the likelihood that these organizational costs are included in the decision-making process, the project's management must be made responsible for the outcome, not just the software installation. In this way they are encouraged to "cost out" projects from an organizational perspective as opposed to only a software application view.

Throughout this discussion we have assumed the extremes, recognizing that reality falls somewhere in between. For example, we have assumed that the organizations' needs are not met by off-the-shelf software. Obviously, this is not always the case. However, when there is a poor fit between the software capabilities and the organization's needs, which is known only after a proper analysis, our conclusions hold. Therefore and despite these limitations, we feel that in general our contention is correct: software acquisition is a decision that should be made only after, and based upon, a sound systems analysis and design. There is no substitute for a feasibility study, systems analysis, and systems design in effective software implementation, regardless of how the actual software code is acquired.

REFERENCES

1.Boehm, B.W. "Software Engineering," IEEE Trans. on Computers, Vol. C-25, No. 12, December 1976, pp. 1226-1241.

2.DeSanctis G. and Courtney, J. F. "Toward Friendly User MIS Implementation," Communications of the ACM, Vol. 26, No. 10, 1983, pp. 732-38.

3.Epple, D., Goodman, P.S., and Fidler, E. "Assessing the Economic Consequences of Organizational Change," in Assessing Organizational Change edited by Standley Seashore, John Wiley and Sons, Inc., New York, 1983, pp. 478-499.

4.Gremillion, L. L. "Managing the Implementation of Standardized Computer Based Systems," MIS Quarterly, Vol. 4, No. 4, 1980, pp. 51-59.

5.Grudnitski, G. "Eliciting Decision-Makers' Information Requirements," Journal of Management Information Systems, Vol. 1, No. 1, 1984, pp. 11-32.

6.Hoeffer, E. "Factors to Consider in Developing Software," Purchasing, November 5, 1981, pp. 25-29.

7.Katz, D., and Kahn, R. L. The Social Psychology of Organizations. John Wiley and Sons, New York, 1966.

8.Lawler III, E. E., Nadler, D. A., and Mirvis, P.H. "Organizational Change and the Conduct of Assessment Research,"in Assessing Organizational Change edited by Standley Seashore, John Wiley and Sons, Inc., New York, 1983, pp. 478-499.

9.Lucas, Jr. H.C. Why Information Systems Fail. Columbia University Press, New York and London, 1975.

10.Macy, B. and Mirvis, P. "Organizational Change Efforts: Methodologies for Assessing Organizational Effectiveness and Program Costs Versus Benefits," Evaluation Review, Vol. 6, No. 3, 1982, pp. 301-372.

11.Mirvis, P., and Lawler III, E. "Measuring the Financial Impact of Employee Attitudes," Journal of Applied Psychology, Vol. 62, No. 1, 1977, pp.1-8.

12.Mirvis, P.H., and Macy, B. A. "Evaluating Program Costs and Benefits," in Assessing Organizational Change edited by Standley Seashore, John Wiley and Sons, Inc., New York, 1983, pp. 501-527.

13.Raysman, R. "Manager Involvement Needed In Computer Selection," Harvard Business Review, Sept.-Oct. 1981, pp.54-58.

14.Segal, H. "Software solutions - choosing your flavor," Modern Office Procedures, May 1981, pp. 22-28.

15.Stabell, C.B. "On the Development of Decision Support Systems as a Marketing Problem," IFIP74, 1974, pp. 962-966.

16.Zmud, R. W. "Individual Differences and MIS Success: A Review of the Empirical Literature," Management Science, Vol. 25, No., 10, 1979, pp. 966-979.

17.Zmud, R. W., and Cox, J. F. "The Implementation Process: A Change Approach," MIS Quarterly, Vol. 3, No. 2, 1979, pp. 35-43.

Computer-Aided Software Engineering: Concepts and Classroom Experiences

Geoffry S. Howard, College of Business, Kent State University
Kent, Ohio 44242

ABSTRACT

The advantages of Computer Aided Software Engineering (CASE) techniques over conventional system design approaches are explained. Excelerator, a representative popular CASE product is discussed, and examples of analysis products produced using Excelerator are presented. The paper relates how CASE techniques have been taught in the undergraduate classroom setting with favorable outcomes for both the school and the students.

Computer-aided software engineering (CASE) refers to an emerging family of tools and techniques that are significantly changing the systems development process. "CASE" is defined as the broad portfolio of tools that automate any portion of the system development life cycle, from graphics tools that speed the creation of data flow diagrams through artificial intelligence-based products that actually generate procedural code. Sometimes also referred to as "analysts' workbenches", most of these products run on PCs and have the ability to eliminate much of the drudgery of analysis and design, while at the same time forcing structured discipline on the development process. In addition to their impact on the costs of development, the use of CASE products is expected to drastically system maintenance costs, because the tools promote the design of highly modular systems and assure the creation of good documentation.

All indications are that CASE is here to stay--it is a near-breakthrough approach to system development. Hence, it is incumbent on educators to learn about the use of CASE tools and to provide our students with the best possible skills and experiences in using them. Injection of large numbers of CASE-trained entry level graduates into the workforce will accelerate the diffusion of this important technology, and enable their organizations to more quickly realize the potential productivity gains.

THE ADVANTAGES OF CASE

CASE tools offer three generic advantages over conventional techniques:

(1) Use of CASE tools forces structured discipline on the systems development process.

(2) CASE tools dynamically produce high quality system documentation, almost automatically and effortlessly, as a by-product of the development process.

(3) Much of the drudgery of the pencil and paper approach to creating data flow diagrams, screen and report layouts, entity-relationship diagrams, and other products of analysis is eliminated when using CASE tools. Additionally, modifications to these analysis products can be made quickly and correctly.

STRUCTURED DISCIPLINE

There is general agreement that the structured approach to analysis and design produces better systems--systems that are easy to modify, modular, and reliable [e.g. 5, 12]. CASE products are designed so that their use automatically forces analysts to comply with the overall concepts of structured analysis [4]. For example, the top level data flow

diagram for a system should contain only one process. The next "lower" level data flow diagram for that system will contain several processes (typically 3 to 7) which represent a functional decomposition of the single process on the top diagram. Each process on a data flow diagram must, because of the conditions imposed by the CASE product, be associated with a particular lower level "explosion" diagram. This means that the analyst is forced to design systems in an organized fashion, from the top down, modularly, following the edicts of functional decomposition that have been shown over the years to lead to more understandable and maintainable systems [6].

Analysts catalog every aspect of a system design into an analysis data dictionary that is maintained by the CASE software. The DD will assure that there are no missing names or conflicting identifiers, and will subject the entire data dictionary to rigorous integrity checks. For commercial systems, with hundreds of processes and thousands of data element specifications, minor specification flaws that would have gone unnoticed until the system testing phase can be detected by the CASE product and fixed before they become expensive and difficult to remedy.

AUTOMATIC DOCUMENTATION

CASE products produce documentation automatically, as the system is developed. Once a final set of data flow diagrams, with all explosion levels, has been produced and finalized, the diagrams can be printed or plotted by the CASE product for future maintainers. Data dictionary entries, particularly for record and data element specifications, can be printed and added to the documentation package, together with the entity-relationship diagrams provided by most CASE products. Screen and report layouts, once finalized, can be easily printed by the CASE package. Even processing specifications, usually in the form of pseudocode or structured English, can be printed by the package for inclusion in the documentation package. This means that essentially all of the system documentation is produced by the CASE tool with no additional work by the analyst. Additionally, individual analyst idiosyncrasies in documentation style are wiped out by the use of the CASE tool.

LESS DRUDGERY

Analysts using CASE tools can throw away templates and erasers. CASE tools typically use high resolution bit- mapped graphic displays with mouse pointing devices, thus making it very easy for analysts to produce and modify data flow and other types of diagrams. If is is necessary to move a symbol on a diagram, the analyst need only point to the symbol and to its new location, and the CASE tool will move the icon, all connecting arrows, and labels and tags automatically.

Various CASE products contain various capabilities and options, but the fundamental advantages: structured discipline, automatic documentation, and less drudgery, are common to essentially all CASE tools [2].

CASE FEATURES - A REPRESENTATIVE PRODUCT

Excelerator is a CASE product from Index Technology, Cambridge, Massachusetts. The product runs on an IBM PC-AT-like microcomputer, and requires at least 512K of RAM, monochrome graphics capability, at least 10MB of hard disk space, and a mouse.

The graphics capability is the most stunning aspect of Excelerator. To create a new data flow diagram, the analyst selects the graphics drawing screen with the mouse, names the

diagram, and then uses an extremely friendly interface to select symbols for processes, entities, and data stores, places them on the screen with the mouse, then specifies how they are interconnected by data flows. Labels, notes, and other text are typed from the keyboard. Data dictionary entries may also be made directly from the graphics screen by pointing to an object to be "described" to the data dictionary, then keying the DD information.

The package supports several symbol standards for DFDs, including the Yourdon and Gane-Sarson standards. Analysts can select a variety of graphics options that control the font size of the lettering, symbol size, horizontal or vertical orientation on the page, method of drawing connector lines, and many others. Users can view a small region of the DFD in detail using a "zoom" feature.

Once an entire set of specifications for a system has been entered into Excelerator, the analyst can point to any symbol on the system DFD and request that an explosion be performed. External entities explode to a description screen containing free text that explains the role of that entity in the system. Data flows explode to record description screens, which in turn explode to data element description screens. Data stores explode to record layout screens. Processes explode to the next lower level data flow diagram.

Additional graphics capabilities support the creation of entity-relationship diagrams for data modeling using a variety of symbol standards, hierarchical structure charts, and "presentation graphs" which contain symbols (such as a computer terminal) that are easily understood by users, and can be important in the construction of user documentation.

Another important capability of Excelerator is its ability to support screen and report layouts. Screens can easily be constructed and modified, and field attributes such as "flashing", "reverse video" and others can be specified with a high degree of flexibility. Screen designs can then be prototyped, and editing rules and accompanying error messages tested. Also, prototype screens can be made to transfer control to other screens, based on user inputs. In this way, users can see an accurate representation of the input and output screens in a system under development, and can actually operate those screens, making suggestions for modifications at a stage in the life cycle where these modifications are easy and inexpensive.

CLASSROOM EXPERIENCES WITH CASE

Excelerator was implemented at Kent State in January of 1987 in support of a 3 credit hour undergraduate course in systems analysis and design. The software was acquired through the Index Technology Inc. educational grant program. Two IBM PC-ATs were acquired for exclusive use by Excelerator students through the proceeds of a State of Ohio grant for instructional computer hardware. Color monitors, dot matrix printers, mice, and HP 7475A plotters were also obtained along with the ATs. Access to these ATs is controlled by a issuing the key for the hard disk drive only to students currently registered for the systems sections (about 40/semester) and selected prior systems students who wish to continue to develop proficiency with the package. Software security is provided by a hardware "key", only two of which are supplied by the vendor. This eliminates the software piracy problem, and protects the school from liability if the software were to be stolen.

After some minimal initial coaching

from the instructor, students have been successful at essentially teaching themselves to use Excelerator with the assistance of a carefully designed programmed learning workbook [10]. This book assumes zero knowledge at startup, and uses a large, integrative case which walks students through all aspects of a system specification project. Students are required to turn in printed Excelerator products at intervals throughout the semester, and have had no difficulty in completing the entire book in a 15 week term. At the conclusion of the course, the students have exercised even the most sophisticated aspects of Excelerator, and are accomplished users of this CASE product.

In parallel with this workbook-based, out-of-class activity, the students receive a standard course on analysis and design, supported by lectures and a conventional textbook. Additionally, they are placed with regional business organizations and required, on a team basis, to execute the analysis and logical design phases of a real-world project. Use of Excelerator is optional for this outside project, but nearly all students have opted to use the package to produce most of the project specification documents.

The educational outcomes of using a real-world CASE product in the course have been positive. Student interest in the course is very high, because the package spices up what is otherwise a deadly dry topic for many students. Additionally, the students are required to read several articles [3, 7, 8, 12] about CASE early in the course. This develops a sense of commitment, because most students appreciate that they are acquiring skills on the leading edge, and that these skills will be salable in the job market. As a result of the workbook exercises, the students emerge from the course with a solid working knowledge of the concepts and mechanics of structured analysis and design. Employers have been uniformly impressed that Kent is using a package that they have heard so much about. There have been several instances of graduates of the program having been hired for above-entry-level positions on the strength of the practical experience acquired in the course using the CASE tool in connection with the real world project. The reputation of the school has benefitted generally from the introduction of state of art CASE technology at the undergraduate level.

SUMMARY

At present, the logical conclusion of the unfolding process of CASE technology development is not clear. While the systems profession is unlikely to become obsoleted by such tools, it appears nearly certain that profound changes are ahead in methods for developing application systems. The profession is already beginning to witness dramatic improvements in system development productivity through the use of CASE tools, such as in the DuPont example. Whatever the eventual shape and form of CASE tools, the immediate urgency for data educators is to embrace these tools and transmit skills in their use to our students. Many key software vendors are making sophisticated CASE products available on attractive terms, and in most cases the hardware required to run them is modestly pricedy. Given the relatively minimal resources needed to include CASE in systems curricula, there is little excuse for inaction.

REFERENCES

[References and a full-text version of the paper are available from the author.]

PROTOTYPING AND ITS PLACE IN INFORMATION SYSTEMS DEVELOPMENT

W. Gregory Wojtkowski, Computer Systems & Decision Sciences, Boise State University, Boise, ID 83725, (208)385-1227
Wita Wojtkowski, Computer Systems & Decision Sciences, Boise State University, Boise, ID 83725, (208) 385-1372

ABSTRACT

In this paper we discuss the nature of the prototyping and the need to systematically introduce it in the System Analysis and Design classes.

We emphasize that this methodology should be considered as a natural part of any course concerned with the fourth generation languages, stressing both, advantages as well as pitfalls, of the prototyping method.

We give an example of a prototyping project and discuss its relevance.

INTRODUCTION: ON PROTOTYPING

The concept of prototyping represents an alternative to the traditional information systems applications development life cycle (SDLC) [2].

In the engineering fields the method of prototyping has been used for years. In the construction of computer application systems it represents rather recent development [1] [14].

The prototyping methodology as applied to computer information systems is based on one essential observation: users of the computer information systems can point to features they don't like in an **existing** system or indicate when a feature is missing more easily than they can describe what they think they would like in an **imaginary** model of a system [11] [13] [16] [17].
This is especially true for the case of the SDLC, where the model of the system is in a form of charts, tables and descriptions which are very often too complicated for the user to understand [18] [19].

Essentially, prototyping represents a dynamic system development model, whereas other classical methodologies such as the SDLC are static.

Information System Prototyping is a methodology used for the system requirements determination, where the user needs are extracted, presented and developed by building a dynamic working model of the intended application system.

The first model of the system is used to enhance communication between the developers of the system and its future users. The model is then gradually expanded and refined as the project participants (developers and users) increase their comprehension of the application [11] [17].

It is often stressed by the practitioners that prototyping should not be used as a tool to compensate for a lack of analysis and design in the system development process. Such a misapplication of the prototyping methodology turns the computer information system development into a trial-and-error methodology [3] [5] [10].

The possibility of a failure of the prototyping process is usually caused by one or more of the following problems:

(1) Not having sufficiently educated users who have an essential understanding of the prototyping process and realistic expectations

of prototyping outcomes;

(2) Not selecting an appropriate business problem (prototyping methodology will not work);

(3) Not selecting the correct prototyping mode. This usually stems from a misunderstanding of the basic motivation for the particular prototyping development effort itself (see point 1);

(4) Not having the proper technical environment to support the prototyping process; and

(5) Lack of effective management methods and controls for prototyping.

Proper solutions to these problems comprise a planning process involved in prototyping, that includes:

(1) Assessment of the application to see if gains will be made from the prototyping;

(2) Development and documentation of a prototype life cycle model that makes sense for a particular situation;

(3) Acquisition and training with the appropriate software tools;

(4) Establishment of the management and control procedures for the prototyping development process; and

(5) Training of both, the users and the technical staff in the application prototype development procedures.

Prototyping, when used well, can lead to a better understanding of the actual application area, especially when it is a complex one, and it can help system designers in getting the initial versions right, before the final construction of a complicated system. This is achieved by experimenting with proposed design [7] [15].

Actually, there are three distinct types of prototyping:

(1) **Exploratory prototyping or requirements prototyping**: This approach can be helpful during rapidly changing market or business conditions or during the establishment of a new business function or department. A prototype of this type will be used to approximate an initial solution to the information system problem quickly, say, in one to three months.

The user operates it for a period of time, analyzes feedback and makes refinements, and the process continues until a stable set of business requirements can be established. Then the prototype is discarded, and a more conventional development process is initiated using the results of the requirements prototyping process as the actual specification.

(2) **Experimental prototyping** is appropriate in any development phase after the initial specifications have been established. This type of prototyping is regarded as an enhancement of the final system specification. Depending on the phase in the development process in which this mode of prototyping takes place, the prototype could serve as a complementary form of specification, a form of refinement of the specification, or an intermediate step between specification and implementation.

(3) **Evolutionary prototyping** is characterized by an initial approximation of the problem, where the prototype becomes the nucleus of the evolving system. Each evolutionary cycle adds more features. In this case, prototyping becomes a replacement for the entire development process.

The most important feature of these

three prototyping approaches is that a tangible model or a prototype is used as a feedback mechanism to help define the final system solution.

PLACE OF PROTOTYPING IN THE C.I.S. CURRICULUM

The prototyping methodology that allows students to experiment with tangible models of real systems offers perfect teaching opportunities which focus on the process not on the technical detail.

The initial version of the prototype, by definition, contains its designer's understanding of the database, screens, reports and inputs that the system will possibly use for communicating with users or other interfacing systems. That is why the prototyping methodology presents an ideal vehicle for teaching about databases, systems designs, output design etc.

In the IS curricula of schools of business, there are two natural places for teaching the prototyping methodologies.

First place is in the course on System Analysis and Design. Here we can stress the fact that the prototyping is a new methodology which can help to communicate the design solutions to the users. We also can show, through examples, that this is one of the methods able to produce system designs that are better and more acceptable to users.

If coupled with now evolving fourth generation tools for rapid prototyping, this methodology can be used successfully in student projects, to specifically create the designs of the real world systems. Later in this paper we will describe an example of such a project conducted in the System Analysis and Design class in the Computer Systems and Decision Sciences Department at Boise State University.

A second natural place to teach prototyping methodology is in the course on Fourth Generation Languages.

Prototyping is one of the most crucial new tools for designing systems with the use of Fourth Generation Languages.
As a matter of fact, fast prototyping is possible because of the fourth generation tools now available on the market. And those are improved every day.

Quite often, in the IS practitioners world, the misapplication of the fourth generation technology is due to the adaptation of inappropriate design technic resulting in unsuccessful system implementation [8] [12] [17]. Students need to be made aware of these problems.

The topics selection for a prototyping teaching module will be described in the next section of this paper.

CONTENT OF THE PROTOTYPING MODULE

Unit 1, Principles of Prototyping, introduces students to the prototyping methodology. It starts with the definition of information system prototyping and describes possible prototyping variants and their applicability to the process of the information system creation. Essentially, it answers the following question "Why should we prototype?".

This unit ends with the description of a prototyping cycle, basic prototyping principles and activities. It also addresses the issue of responsibilities of those who are prototyping participants.

Unit 2, How to Prototype, helps students with the mastery of prototyping methodology.

Subjects discussed are: prototyping objectives and planning, selection of projects for prototyping, life cycle

models for the variants of prototyping (exploratory prototyping, experimental prototyping and evolutionary prototyping), necessary tools for prototyping, managing and controls of the prototyping process and training of users and data processing staff.

This unit ends with some examples of the prototyping success stories from the real word of MIS applications. This helps our students to appreciate the methodology.

At the end of this unit students will be able to recognize projects appropriate for the prototyping process.

Unit 3, Some Management Issues in Prototyping, deepens the discussion begun in the end of Unit 2 on managing prototyping projects.

Students are introduced to some of the more advanced topics on management of prototyping projects. These topics include: project management, specifications management, change management, staffing prototyping projects and establishing the systems delivery intervals.

This unit ends with the discussion of possible prototyping pitfalls.

PROTOTYPING PROJECT FOR THE SYSTEM ANALYSIS AND DESIGN COURSE: AN EXAMPLE

The project in the following example was executed by the students in IS420-System Analysis and Design course taught at Boise State University, School of Business.

The project concerns an online system which handles legal data on land properties, land leases and water rights for a large corporation located in Idaho.

The system requires the design of 10 files and table lookups, 20 data entry screens and menus, and 15 reports.

The relationships between records for the properties, land leases and water rights files were many-to-many.

For example, students have to handle relationships such as: a property with many leases, a property with many water rights, a water right which spreads over many properties and a lease which includes many properties and so on.

The corporate end-user for whom this system was designed initially had no idea what kind of information was necessary and sufficient for the designers of the intended systems. The Legal department of this corporation can be classified as an inexperienced end-user of computer data processing.

To start with, the initial legal information system in this corporation was completely manual. Moreover, the designers of the computerized system (students in IS420) just learned how to prototype. They also had no knowledge of the end-user expertise domain. For example, they did not know how to handle water rights or property rights nor did they know what the essential information needs of this end-user were. Under those circumstances, the project was perfect for exploratory prototyping.

The prototyping models of the final system served as practical demonstrations of possible system functions. It was a critical catalyst for eliciting good ideas and promoting a creative co-operation between the end-user and the students-designers.

The fast creation of the functioning models of the final system was possible because we used the fourth generation tool from COGNOS Corporation - an Architect.

The Architect is a powerful screen-driven system used for creation of

complete system prototypes using the fourth generation language POWERHOUSE. The complete system and the documentation associated with it is created in a fraction of the time normally required. The Architect integrates every stage of the application development, from definition of the first data element to the maintenance and modification of working prototypes.

Created prototypes are online, menu driven systems.
From within the Architect development environment students- designers can:

(1) Build a data dictionary and data files;
(2) Build a working prototype system automatically:
(3) Modify any component of an application;
(4) Cross-reference files, data elements, programs, defined items, temporary items and procedure; and
(5) Completely document the prototyped system.

Eight teams of the students-designers (5 persons in each team) worked independently on the same design project. They were able to deliver different prototypes of the final system on a weekly basis.

In each iteration, their models became better approximations of the system the end-user needed, since they used these prototypes to communicate to the unexperienced end-user what the computer could do in the application. In addition, students began to truly understand the information needs and the expertise domain of the end-users.

The final prototype was built in five weeks and formed the base for the requirements specification. These were used in the next semester to build the final system in IS430 - Project Implementation course.

CONCLUSIONS

Given the result from our courses, we feel that introducing students to the prototyping methodologies, when combined with the Fourth Generation technology, allows one to introduce technical concepts more rapidly. Students undertake design and programming tasks at greater depth and difficulty, and this is done more quickly.

This approach is of particular importance given the wide scope of material to be covered within the short time span, in a single semester. An interaction with the end-user and experience gained from it is most valuable to the students.

REFERENCES

1. Adamski, L., "Prototyping," Computerworld, Vol. 19, No. 18, pp. 23-32, May 6, 1985.
2. Boar, B. H., "Application Prototyping," John Wiley and Sons, 1984.
3. Brooks, F. P., "No Silver Bullets", Unix Review, Vol. 5, No. 11, pp. 38-48, 1987.
4. Brancheau, J. and Wetherbe, J. C., "Key Issues in Information Systems," MIS Quarterly, Vol. 11, No. 1, pp.23-45, March 1987.
5. Cesena, M. L., and Jones, W. O., "Accelerated Information System Development in the Army," SIM Spectrum, Vol. 4, No. 3, September, 1987, pp. 1-8.
6. Cobb, R. H., "In Praise of 4GL'S," Datamation, pp. 90-96, July 1985.
7. Goyette, R., "Fourth Generation Systems Soothe End User Unrest," Data Management, Vol. 24, No. 1, pp. 30-32, January 1986.
8. Grant, F. J., "The Downside of 4GLs," Datamation, pp. 99-104, July 1985.
9. Harel, E. C., McLean, E. R., "The Effects of Using a Nonprocedural Computer Language on Programmer Productivity," MIS Quarterly, pp.

109-120, June 1985.
10. Kolodziej, S., "The Fate of 4GLs," Computerworld-Focus, Vol. 22, No. 5A, pp. 25-28, February 3, 1988.
11. Kozar, K. A., Mahlum, J. M., "A User Generated Information System: an Innovative Development Approach," MIS Quarterly, Vol. 11, No. 2, pp. 163-175, March 1987.
12. Kull, D., "Anatomy of a 4GL Disaster," Computer Decisions, January 11, pp. 58-65, 1986.
13. McCracken, D. D., "Software Systems in the 80's: An Overview," Computerworld Extra 14, No. 38, September, 1980, pp. 5-10.
14. McNurlin, B. C., ed., "Developing Systems by Prototyping," EDP Analyzer, Vol. 19, No. 9, September, 1981, pp. 1-14.
15. Nelson, R. R., Cheney, P. H., "Training End Users: An Exploratory Study," MIS Quarterly, Vol. 3, No. 4, pp. 547-559, December 1987.
16. Neumann, J. D. and Jenkins, M. A., "Prototyping: The New Paradigm for System Development," MIS Quarterly, pp. 29-41, September, 1982.
17. Nolan, D., "The First Two Years, Four Months, Two Weeks and Seven Minutes of a New System's Life," Computerworld, Vol. 20, No. 41, pp. 93-100, October 13, 1986.
18. Paul, L., "Prototyping Hailed as User-Tailored Approach," Computerworld, Vol. 18, No. 22, Special Report p. 62, May 28, 1984.
19. Scharer, L. "General Problem-Solving Methods Will Ease Your Move to Prototyping, if You Use Them," Computerworld, Vol. 18, No. 22, Special Report, pp. 30-31, May 28, 1984.

STRUCTURED FILES FOR INTERACTIVE GRAPHICS PROGRAMS

Dan R. Olsen Jr.
and
Robert P. Burton

Brigham Young University
Provo, Utah 84602

Abstract

A data management system has been designed for interactive graphics programs. Files can be built from records, unions and variable-length sequences. These datatypes can be manipulated in main storage, much as Pascal or C objects are manipulated. The system preserves the characteristics of files and datatypes loaded from main storage or saved to secondary storage. The system supports pointer and associative accesses to datatypes in a file. Each file is self-describing, permitting viewing, manipulation and general purpose programs independent of the generated file format. The system is in continuous use in production programming.

INTRODUCTION

A large portion of any interactive program is designed to browse and edit the data representation of the situation the program is intended to model. Browsing and editing frequently require data to be saved and loaded to and from secondary storage. Files in secondary storage become the communication media that link together several interactive programs in an application environment.

Unfortunately, the data structures used in many interactive programs are not well suited to secondary storage. Graphics applications, requiring efficient interaction between saveral programs modules, are supported inadequately by both a UNIX environment and a Data Base Management System. Linked lists and trees cannot be handled efficiently. Pointers containing main storage addresses lose meaning when structures containing them are moved to secondary storage. Pointer based structures typically cannot be accessed associatively. Alternatively, associative structures typically cannot be random-accessed efficiently.

In a UNIX environment, a byte stream provides communication between programs. Every provider program is required to have a routine which linearizes the structure into a byte stream. Every consumer program is required to have a routine which restructures the data from the byte stream. The program designer may linearize his data structures, writing them out in binary form, rendering them incomprehensible as they are passed between programs. Alternatively, he may linearize his data structures into some textually comprehensible form supported by UNIX, reparsing them to reconstitute his data structures. This approach does not lend itself well to graphics data which is no more comprehensible in text form than in binary form.

A second common approach uses a Data Base Management System. Data Base Management Systems sacrifice efficient access to individual tuples to provide several users with access to large amounts of data. This exchange causes severe modeling and efficiency problems. A typical DBMS anticipates substantial data and relatively minor structure. Many interactive applications, which use linked lists, trees and associative access structures, may require complex structures. Associative accesses map well onto the query facilities of a DBMS. However, linked lists and trees are awkward to create and inefficient to use in a relational data base environment. Linked lists and trees are easier to model in a hierarchic or networked data base environment, but inefficient to use nonetheless.

Graphics applications require fast random access to relatively small files. A file manipulated by a graphical editor typically occupies less than 100K of storage and rarely exceeds 1M of storage. Thus, the entire file can be loaded into main storage for manipulation and saved to secondary storage at the conclusion. Virtual memory makes this approach entirely feasible.

A file structure is needed which can be manipulated in main storage, much like a Pascal or C data structure, but with efficient saving and loading to and from secondary storage. If the file structure is self-describing, generalized tools can be written to view and manipulate the data represented in the structure. The concepts of generalized viewing and manipulation are represented in the text utilities in UNIX [Bourne 83], but are encumbered by linearization requirements.

The design and implementation of structured files for interactive programs described in this paper overcome the linearization requirements of UNIX and the inefficiency of a Data Base Management System. Previous work on graphical interfaces to relational data bases has influenced the design [Garrett 80]. Access is provided to linked structures, records, unions and variable length linear sequences. Generalized viewing and manipulation are supported.

DATA TYPING

Every file has a "filetype" defining the type of data the file can contain. For example, a compiler may have one filetype for the intermediate form produced by the parser and another filetype for the parse tables produced by the parser generator. A filetype is defined by a set of "datatypes." A datatype is either a record, a union or a sequence. A representative filetype for a hierarchic picture is shown in figure 1.

```
Picture = Sequence (Primitive)

Primitive = Union
               LinePrim: Line;
               CirclePrim: Circle;
               SubPicture: Picture
            End;

Line =      Record
               EndPoint1: Integer[2];
               Endpoint2: Integer[2]
            End;

Circle =    Record
               Center: Integer[2];
               Radius: Integer
            End;
```

Figure 1
Sample Filetype Definition

In this example, the datatypes Picture, Primitive, Line and Circle constitute a filetype.

A record type contains fields for each "object" of that type. Record types are similar to Pascal, C and Ada record types. A union type defines fields for objects of that type; however, only one field is present for any object of union type. For example, a primitive can be a line, a circle or a text string. Union types are similar to variant records in Pascal and unions in C. A union type is tagged with a discriminant field to identify its contents to the generic tools. A sequence type has one field and consists of a variable length array of such fields.

Every field has a "fieldtype." A fieldtype consists of a type and a dimension. Primitive types are integer, real, Boolean, char, byte and string. Byte is an unsigned integer in the range 0..255. A fieldtype of dimension 1 is a scalar fieldtype. A fieldtype of dimension greater than 1 is an indexed fieldtype. Str is always an indexed fieldtype. Indexed fieldtypes are fixed length, one-dimensional arrays. For example, Integer[2] is an indexed fieldtype and an array containing two integers. Str is an array of characters; the number of characters in the string is stored in the zeroth byte of the array. This implementation is faster than the implementation of true, variable length strings. Sequence (char) may be used for variable length strings.

Two special datatypes, "type" and "field," are defined in every file. When the filetype is created, the set of objects associated with "type" and "field" is fixed. "Type" objects define each of the datatypes belonging to the filetype. One field in a "type" object is

a pointer to a linked list of "field" objects containing the fields for that datatype. "Type" and "field" datatypes provide the self-describing facility of the structured files.

FILE FORMAT

A file can reside either in main storage or in secondary storage. While the formats are similar, there are some important differences. Logically, within the file each datatype is allocated a variable length array. References to objects of a datatype are relative to the base address of the datatype's array. In the case of records and unions, the array can be viewed as an array[1..dimension] of T where T is a type definition of tuples of that type. In these two cases, an object is referenced by an index into the array. Array indices are address space independent, provided the base address of the array is known. In the case of sequences, the index is treated as a byte offset relative to the base address. This array model also allows for associative searches. The search begins with the first index into the array containing objects of the desired type. The search proceeds through the array seeking matches. Relational, associative and pointer-based structures are all supported by this model. In typical dynamic allocation systems, pointer and associative accesses rarely are available simultaneously.

OBJECT ALLOCATION AND DEALLOCATION

Allocation and deallocation of records and unions differs from allocation and deallocation of sequences. Records and unions are of fixed sizes, simplifying the algorithms. A linked list of available objects is kept in the "type" record for each datatype. This list contains the index of the first object in the list. A pool of available objects is kept at the end of each type's array. This pool contains objects that have never been allocated. This pool initially receives one tenth of the array space allocated to the datatype when a file is loaded from secondary storage, reducing the need to resize the array subsequently. The pool is eliminated when the file is saved to secondary storage where no modification of files occurs. Until the request is satisfied, a request for a new object causes first a search of the list, then a search of the pool and finally a resizing of the array. A tag field associated with every record and union indicates its active or deleted status, permitting associative search routines to skip over deleted tuples.

Sequences are handled differently because of their variable lengths. Each sequence in an array is referenced by a byte offset relative to the base location for its datatype. The number of bytes in the sequence is stored in both the header and the trailer of the sequence. Deletion causes the length stored in the header to become negative. The "type" record for the datatype maintains a pointer to one of the sequences. This serves as a starting place for searching through the deleted sequence to find available space. In place of a pool of space that is not yet allocated, a single deleted sequence at the end of the datatype's space is used. Deleted sequences are split as needed; adjacent sequences are merged as they are deleted.

"TYPE" AND "FIELD" DEFINITIONS

Self-description is provided by the special datatypes "type" and "field." Each datatype in a filetype, including "type" and "field," is described in this manner. "Type" records also contain information needed to manage storage for the type. The record "type" is defined in figure 2.

```
Type = Record
          TypeName: Str[20];
          TypeKind: Byte;
          (*   1 = Record      *)
          (*   2 = Union       *)
          (*   3 = Sequence    *)
          FirstField: Field;
          (*   Index to the head of a linked list of fields defining
               this type.  If TypeKind indicates a sequence, there
               is exactly one field in the list. *)
          FirstAvailEl: Integer;
          (*   For a Record or Union, the index in the datatypes
               array of the head of a linked list of deleted
               objects.  For a Sequence, the index indicating where
               the search terminated and will begin for the next
               operation. *)
          AvailPool: Integer;
          (*   For a Record or Union, the index to the last object
               which is either allocated or in the avail list.
               Objects beyond have not been allocated.  For a
               Sequence, this field is ignored. *)
          TypeSize: Integer;
          (*   For a Record or Union, the number of bytes required
               for one object of this datatype.  For a Sequence, the
               size in bytes of one sequence element. *)
          TotalSpace: Integer;
          (*   Space allocated to the datatype including deleted and
               pool space.  For a Record or Union, the number of
               objects.  For a Sequence, the number of bytes. *)
          End;
```

Figure 2
Type Descriptor Definition

A "type" object controls the behavior of the remainder of a file. Therefore, it cannot be modified after a file has been generated except during allocation and deallocation.

The "field" datatype contains information needed to describe and extract the contents of fields. The record "field" is defined in figure 3.

```
Field = Record
          FieldName: Str[20];
          FieldTypeKind: Byte;
          (*   1 = Byte        *)
          (*   2 = Integer     *)
          (*   3 = Real        *)
          (*   4 = Char        *)
          (*   5 = Boolean     *)
          (*   6 = Str         *)
          (*   7 = composite   *)
          CompositeType: Type;
          (*   If FieldTypeKind = 6, an index into "type" defining
               the type of objects to which this field points *)
          Dimension: Integer;
          (*   1 : a scalar field; > 1 : an indexed field specifying
               quantity.  For a Str field, the maximum number of
               characters.  Additional space is provided for the
               length byte. *)
          Offset: Integer;
          (*   The byte offset of the field from the base location
               *)
          Nextfield: Field
          (*   Index of the Next field in the linked list of fields
               to which this belongs *)
        End;
```

Figure 3
Field Descriptor Definition

Figure 4 provides "type" and "field" information for the sample filetype of figure 1.

TYPE

	Type Name	Type Kind	First Field	First FreeEl	Free Pool	Type Size	Total Space
1	Type	Record	1	Nil	6	43	6
2	Field	Record	8	Nil	20	31	20
3	Picture	Sequence	13	Nil	Nil	4	100
4	Primitive	Union	14	Nil	Nil	5	100
5	Line	Record	17	Nil	Nil	17	100
6	Circle	Record	19	Nil	Nil	13	100

FIELD

	Field Name	FieldType Kind	Composite Type	Dimension	Next Field
1	TypeName	Str	Nil	20	2
2	TypeKind	Byte	Nil	1	3
3	FirstField	Composite	2(Field)	1	4
4	FirstFreeEl	Integer	Nil	1	5
5	FreePool	Integer	Nil	1	6
6	TypeSize	Integer	Nil	1	7
7	TotalSpace	Integer	Nil	1	Nil
8	FieldName	Str	Nil	20	9
9	FieldTypekind	Byte	Nil	1	10
10	CompositeType	Composite	1(Type)	1	11

FIELD (Cont.)

	Field Name	FieldType Kind	Composite Type	Dimension	Next Field
11	Dimension	Integer	Nil	1	12
12	NextField	Composite	2(Field)	1	Nil
13	PrSeq	Composite	4(Primitive)	1	Nil
14	LinePrim	Composite	5(Line)	1	15
15	CirclePrim	Composite	6(Circle)	1	16
16	SubPicture	Composite	3(Picture)	1	Nil
17	EndPoint1	Integer	Nil	2	18
18	EndPoint2	Integer	Nil	2	Nil
19	Center	Integer	Nil	2	20
20	Radius	Integer	Nil	1	Nil

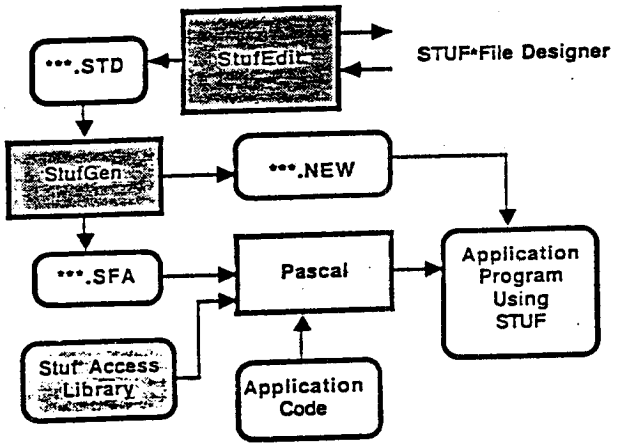
Figure 4
Sample Encoding of Type and Field Information

Every file in main storage has a header record containing information including the name of the secondary storage file from which the file originated and an array of pointers to a data space for each datatype. The first pointer references the "type" datatype space which contains all of the storage allocation information for the file. When a datatype requires additional space, 1) space is allocated, 2) existing information is copied, 3) the previous space is liberated, 4) the pointer to the datatype's space is changed and 5) the appropriate entry in the "type" space is updated. Indices for type pointers match corresponding entries for the datatype in the "type" array.

Writing a file to disk consists in writing the binary image of the datatype arrays to the file in the order in which they appear in the "type" datatype. "Type" and "field" are written first. Reading a file begins by reading the first "type" record in a fixed format. The AvailPool field contains the number of "type" records, permitting the remainder of the "type" datatype to be read. Necessary information about the size and quantity of each of the other datatypes is contained in the "type" datatype. The remainder of the process consists in allocating space for each datatype and in reading the associated binary images from the file.

SUPPORT TOOLS

Because raw files are not suitable for direct manipulation by interactive programs, several support tools have been developed. While the tools are Pascal based, similar tools can be implemented in any language which supports user-defined type declarations and dynamic storage allocation. An overview of the tools is provided below. Figure 5 presents the interaction of the tools.



* STructUred Files

Figure 5
Support Tools

A new application involving structured files uses the StufEdit program to create a definition of the filetype. A "type" definition file which contains a textual definition of the filetype can be created or modified using StufEdit. The textual format assures independence from word size and memory structure. The filetype definition file is then passed to the StufGen program. StufGen must be specific to a language and host processor. StufGen programs have been developed for VMS/Pascal on Vax hosts and Turbo Pascal on IBM/PC hosts. A man week is sufficient to create a StufGen program for a new language and host.

StufGen produces two outputs. The first is an empty structured file for the filetype being created. This file contains values for the "type" and "field" datatypes only. A new structured file can be created only by using StufGen. Other structured files are created by copying existing files and adding or deleting records. This simplifies the problem of maintaining the integrity of "type" and "field" information.

Second, StufGen produces access routines tailored specifically to a programming language and filetype. Frequently, more than one filetype is accessed by a program. Therefore, StufGen allows the specification of a local filetype to be appended to the front of all locally generated names, making them unique. The example shown in figure 6 uses the name PIC. Every object in a structured file is referenced by an index into the file with a private interpretation. Therefore, "type" definitions shown in figure 6 are created for these indices to insulate the code from the implementation.

```
Type
     PicPictureIDX = Integer;
     PicPrimitiveIDX = Integer;
     PicLineIDX = Integer;
     PicCircleIDX = Integer;
```

Figure 6
Index Types

Because the general routines which apply to any filetype use datatype numbers, a constant is assigned to each datatype (figure 7).

```
Const
     PicPictureNum = 3;
     PicPrimitiveNum = 4;
     PicLineNum = 5;
     PicCircleNum = 6;
```

Figure 7
Datatype Constants

The routines in a .SFA file (see figure 5) map data objects stored in the structured files into the type declaration model of the language. Once a particular data object has been accessed, it can be manipulated by the data referencing structures of the host language. Two type declarations are created for each datatype. The first is a description of the data object. The second provides a pointer to the type to be returned by the object referencing routines.

The design of this portion of the system required that some robustness be sacrificed for efficiency. However, the typing facilities of the language were exploited to guard against improper usage.

The datatype determines the type declaration. The datatype of figure 1 is a Picture which is a sequence. Implementation of a variable length sequence as a fixed length Pascal array uses an array of maximum size, in which some elements at the end of the array typically are meaningless. Figure 8 shows the generated types for Picture.

```
Type
     PicPicture = Array[1..maxsize] of PicPrimitiveIDX;
     PicPicturePtr = ^PicPicture;
```

Figure 8
Sequence Datatype Declarations

The Primitive type in the example is a union. A union is implemented in Pascal as a variant record. Figure 9 shows an additional, generated Tag type, enumerating variants for each field. The actual record declaration contains a field qqST which is used to manage deleted and active objects and other marking chores. The Tag field is always present. Free unions are not allowed because the general purpose viewing and editing routines require a discriminant field.

```
PicPrimitiveTag =
     (PicLinePrimTG,
      PicCirclePrimTG,
      PicSubPictureTG);
PicPrimitive = Record
                    qqST: Stf2ObjectStatus;
                    Case Tag: PicPrimitiveTag of
                         PicLinePrimTG:
                              (LinePrim:PicLineIDX);
                         PicCirclePrimTG:
                              (CirclePrim:PicCircleIDX);
                         PicSubPictureTG:
                              (SubPicture:PicPictureIDX)
               End;
PicPrimitivePtr = ^PicPrimitive;
```

Figure 9
Union Datatype Delcaration

Record datatypes are modeled in Pascal as shown in figure 10.

```
     PicLine = Record
                    qqST: Stf2ObjectStatus;
                    EndPoint1: Array[1..2] of Integer;
                    EndPoint2: Array[1..2] of Integer
               End;
     PicLinePtr = ^PicLine;

     PicCircle = Record
                    qqST: Stf2ObjectStatus;
                    Center: Array[1..2] of Integer;
                    Radius: Integer
                End;
     PicCirclePtr = ^PicCircle;
```

Figure 10
Record Datatype Declarations

The type declaration generated last defines file variables for a particular filetype. Use of the file with routines other than those created for the filetype is prevented.

```
     Type Pic = Record
                    FileHeader: StfFileHeader;
                    ...
                    ...
                End;
```

Figure 11
Filetype Type Declaration

The only field of interest in the filetype declaration is the FileHeader field. This field is used when calling the generic access routines described below.

The bulk of an .SFA file consists of three types of routines used to obtain access to data objects in a structured file. Access to an object requires its index to be dereferenced. The first two types of routines are generated to perform this function. typeRef routines are used to look at an object. typeMod routines are used to modify an object. Changes to structured files can be monitored using these routines. The third type of routine, typeLength, is generated for every sequence datatype. The routines in figure 12 are generated for the example Picture filetype.

```
Function PicPictureRef (StufFile: Pic; Idx: PicPictureIdx):
     PicPicturePtr;
Function PicPictureMod (StufFile: Pic; Idx: PicPictureIdx):
     PicPicturePtr;
Function PicPictureLength (StufFile: Pic; Idx: PicPictureIdx):
     Integer;

Function PicPrimitiveRef (StufFile: Pic; Idx: PicPrimitiveIdx):
     PicPrimitivePtr;
Function PicPrimitiveRef (StufFile: Pic; Idx: PicPrimitiveIdx):
     PicPrimitivePtr;


Function PicLineRef (StufFile: Pic; Idx: PicLineIdx):
     PicLinePtr;
Function PicLineRef (StufFile: Pic; Idx: PicLineIdx):
     PicLinePtr;

Function PicCircleRef (StufFile: Pic; Idx: PicCircleIdx):
     PicCirclePtr;
```

```
Function PicCircleRef (StufFile: Pic; Idx: PicCircleIdx):
     PicCirclePtr;
```

Figure 12
Generated Reference, Modification and Length Routines

GENERIC ROUTINES

In addition to the generated routines for manipulating files of a particular filetype, a library of routines has been built to manipulate any structured file, regardless of its filetype. Figure 13 lists generic routines.

```
Function StfRead(StufFile: StfFileHeader; FName: InString): Boolean;
     (* Read a file of the specified name into StufFile.  If the
     read fails then a false result is returned; otherwise the
     result is true. *)

Function StfWrite(StufFile:StfFileHeader; FName: Instring): Boolean;
     (*  Write the contents of StufFile to the file named in FName.
     If the write succeeds then true is returned. *)

Procedure StfNew(StufFile: StfFileHeader; TypeNum:Integer; Var
ObjIdx: Integer)
     (* Allocate a new object in StufFile whose datatype is
     specified by TypeNum.  The index of the new object is placed in
     ObjIdx. *)

Procedure StfDispose(StufFile: StfFileHeader; TypeNum: Integer;
ObjIdx: Integer);
     (* Deallocate an object of the specified datatype in StufFile.
     *)
```

Figure 13
Generic Routines

Additional generic routines initialize, clear or dump structured files and provide access to fields of various types.

SAMPLE DATA MANIPULATION

Figure 14 contains a code segment showing how generated and generic routines are used to perform simple manipulations on the sample filetype.

```
Var PicFile: Pic;
     (* the structured file being manipulated *)

Function NewLinePrimitive (X1, Y1, X2, Y2: Integer):
PicPrimitiveIdx;
     (* Create a new Line primitive and return its index *)
     Var  NewPrimIdx: PicPrimitiveIdx;
          NewPrimPtr: PicPrimitivePtr;
          NewLineIdx: PicLineIdx;
          NewLinePtr: PicLinePtr;
     Begin
     (* Create a new Primitive object and retrieve a pointer to it
     *)
          StfNew (PicFil.FileHeader, PicPrimitiveNum, NewPrimIdx);
          NewPrimPtr := PicPrimitiveMod (PicFile, NewPrimIdx);
     (* Create a new Line object and retrieve a pointer to it *)
          StfNew(PicFile.FileHeader, PicLineNum, NewLineIdx);
          NewLinePtr := PicLineMod(PicFile,NewLineIdx);
     (* Fill in the fields in the Primitive object *)
          NewPrimPtr^.Tag := PicLinePrimTG;
          NewPrimPtr^.LinePrim := NewLineIdx;
     (* Fill in the fields in the Line object *)
          NewLinePtr^.EndPoint1[1] := X1;
          NewLinePtr^.EndPoint2[2] := Y1;
          NewLinePtr^.EndPoint2[1] := X2;
          NewLinePtr^.EndPoint2[2] := Y2;
     NewLinePrimitive := NewLineIdx;
     End; (* NewLinePrimitive *)

Function FindCircle (SearchPic: PicPictureIdx; XP, YP: Integer):
Index;
     (* Find the location in SearchPic of the first circle
     which contains the point (XP, YP) *)
     Var  I, PicLen, Distance: Integer;
          PicPtr: PicPicturePtr;
          PrimPtr: PicPrimtivePtr;
          PrimPtr: PicPrimitivePtr;
          CirclePtr: PicCirclePtr;

Begin
     PicLen := PicPictureLength (PicFile, SearchPic);
     I := 1;
     FindCircle := 0;
     PicPtr := PicPictureRef (PicFile, SearchPic);
     While I <= PicLen Do
          Begin
               PrimPtr  :=  PicPrimitiveRef  (PicFile,
               PicPtr^[I]);
               If PrimPtr^.Tag = PicCirclePrimTG Then
                    Begin
                         CirclePtr  :=  PicCircleRef
                         (PicFile, PrimPtr^.CirclePrim);
                         Distance  :=  Sqrt((XP-
                         CirclePrim^.Center[1])**2  +  (YP-
                         CirclePrim^.Center[2])**2);
                         If Distance <= CirclePrim^.Radius Then
                              Begin
                                   FindCircle := I;
                                   I := PicLen;
                              End;
                    End;
               I := I + 1;
          End;
End; (* FindCircle *)
```

Figure 14
Sample Data Manipulation

File usage and object access match closely the usage and access of main storage data structures.

EXPERIENCE WITH TOOLS FOR VIEWING AND MANIPULATING FILES

Three sets of generic tools for viewing and manipulating files have been developed. The first set employed a generalized syntax directed editor [Scott 85]. The editor is driven by a format which is based on left attributed context free grammars. While the approach was effective, it was inefficient in the general purpose algorithms, but not in the structured file routines. Substantial flexibility dictated conservative algorithms which proved inefficient.

The second set employed a forms style editor [MacKay 86]. The editor is driven by formats and proved slightly more efficient. However, generality again reduced interactive efficiency.

The third set employs editing templates [Olsen 86b]. An editing template represents a style of data display such as an alphabetized list, a scrolling linked list or a schematic diagram. Each template is parameterized by the actual structures being manipulated for a particular instance. Each template is hard coded for a particular style of display, incorporating necessary efficiencies in the algorithms. Substantial generality is achieved in creating new editing templates while preserving interactive response times.

CONCLUSION

A data management system has been designed for interactive graphics programs. The system preserves the characteristics of main storage data structures whether they actually reside in main storage or secondary storage. The system supports viewing and manipulation of any file. The system is in continuous use in production programming.

References

[Bourne 83] Bourne, S. The Unix System. Addison-Wesley Publishing Company, 1983.

[Garrett 80] Garrett, M. "A Unified Non-procedural Environment for Designing and Implementing Graphical Interfaces to Relational Data Base Management Systems." Ph.D. dissertation, Dept. of EE & CS Technical Report GWU-EE/CS-80-13, The George Washington University, Washington, D.C. 1980.

[MacKay 86] MacKay, P. "Fixed Forms Display of Arbitrary Data Structures." M.S. thesis, Dept. of CS, Brigham Young University, Provo, UT, 1986.

[Olsen 86a] Olsen, D. "An Editing Model for Generating Graphical User Interfaces." Procedings of Graphics Interface 86, Canadian Man-Computer Communications Society, May 1986.

[Olsen 86b] Olsen, D. "STUF: User's Manual." Internal Report, Dept. of CS, Brigham Young University, Provo, UT, 1986.

[Scott 85] Scott, E. "Generalized Parsing and Display of Textually Represented Data Structures." M.S. thesis, Dept. of CS, Brigham Young University, Provo, UT, 1985.

THE APPROPRIATENESS OF COMPUTER CONFERENCING IN TEACHING ACROSS DISCIPLINES

CAROL SAUNDERS
M. J. NEELEY SCHOOL OF BUSINESS
P. O. BOX 32868
TEXAS CHRISTIAN UNIVERSITY
FORT WORTH, TEXAS 76129
(817) 921-7421

Abstract

This paper studies the appropriateness of computer conferencing in courses across different academic disciplines. It is hypothesized that computer conferencing, since it is relatively low on "information richness", will be more appropriate in disciplines with a high level of paradigm development. There is a high level of agreement among practitioners in these fields concerning key concepts and methods of solving problems. Tests of the hypothesis on a sample of 21 students in a nontraditional, masters-level program are significant, but in a direction opposite from that hypothesized. Implications of the findings are discussed.

USER INTERFACE DESIGN IN THE M.B.A. MIS COURSE: A CASE EXPERIENCE

Neil Jacobs, Gregory Neal, Jack Kant, and Austin Byron
Northern Arizona University

ABSTRACT

From the users' perspective, probably no aspect of information systems design has made such dramatic progress in the last five years as has the design of the user interface. User interfaces are becoming increasingly significant and sophisticated. This paper describes the introduction of user interface design in a graduate MIS course required of all students in a M.B.A. program. The implementation involved no programming and resulted in students developing a pseudo-interactive microcomputer-based order processing system while taking only one and one-half weeks of the semester course. Student and instructor reactions are discussed, and suggestions on the future role of user interface design are provided.

INTRODUCTION

One has only to consider the recent evolution of microcomputer software packages to see the growing significance of user interface design for information systems. For example, consider the evolution of what has been a leading database management system for microcomputers, Ashton-Tate's series of dBASE programs. The change in dBASE's user interface depicts the recent transition in user interface design--and importance. dBASE II's interface was noted for the dot (".") prompt which insisted the user know what to do next; dBASE III added a procedural user interface with pull-down menus and other features to ease use; and dBASE IV is purported to further ease use by incorporating a nonprocedural user interface which lets users execute tasks without walking through a series of command selections (PC Week, Vol. 5, No. 7, February 16, 1988). Similar evolutions are occurring in other software packages, e.g., other data base management systems, fourth generation languages, and word processing programs. In general, these changes have provided more capability while demanding less user "computer" expertise through enhanced user interface designs.

It doesn't require much imagination to predict that users will become more demanding regarding the design of the user interfaces for organizational information systems. The sophisticated interfaces available in common software packages (1) provide users with insight into potential interface features, e.g., pull-down menus and context sensitive help, and (2) start to define de facto standards regarding functionality and ease of use. Another impetus for more sophisticated user interfaces is the expansion of application systems in organizations. This expansion of the user base will add many unsophisticated users with greater need for enhanced interfaces to use systems effectively.

Users need exposure to and designers need experience with more sophisticated user interface designs than are practical with traditional techniques which require programming skills.

This paper describes an approach to provide business students an exposure to sophisticated user interface design utilizing user interface simulation software. The approach requires no programming and was completed in less than two

weeks of a course. Student and instructor reactions are discussed, and suggestions offered regarding the use of simulation software to provide user interface design training.

INTRODUCING USER INTERFACE DESIGN IN THE GRADUATE MIS COURSE

Setting. Twenty-three graduate students in a graduate Management Information Systems course required of all M.B.A. students provided the setting. Only a few of the students intended to pursue the MIS option of the M.B.A. program, so most were pursuing a basic M.B.A. degree. The user interface design task was introduced as a group project about one-third of the way through the semester. An in-class demonstration of the simulation software (DEMO by Software Gardens, Inc.) was given, teams of two assigned, and a user interface design task built around a short case was distributed. The task was completed eight days later.

Preparatory reading assignment. Prior to the user interface task, students were assigned two selected readings. One was on the psychological and behavioral aspects of information systems (Ahituv and Neumann, 1986); the other dealt with more specific user interface design issues (Kozar, circa 1988).

Task assignment. The user interface design task assignment was built around a four page case involving a microcomputer telemarketing firm using rudimentary, nonintegrated systems, e.g., stand alone systems for order processing, inventory, and accounts receivable. Software was not integrated and the hardware was not networked in any way. (The case was developed from a consulting engagement by one of the authors and is included in Appendix A.) Students took the role of a consultant who wanted the job of providing an integrated information system. The consultant planned to demonstrate her ideas and competence by constructing a simulated user interface for the order processing portion of the system. The consultant was committed to prepare an initial user interface design over the coming weekend, discuss the design with the firm's owner/manager on the following Monday, and deliver an updated simulated design by the following Thursday. The order processing operation had to show an integrated capability to: check contents of the product line, help salespeople guide customer consideration of alternative brands or models, check inventory status, provide pricing dependent on customer status, confirm credit acceptability, and enter the order.

User interface simulation software. The DEMO software provides the capability to show how a working program will appear. With DEMO, one can create screen prototypes and simulate the operation of a running program. Screens of text and drawn lines are easily constructed in monochrome or color. These screens are called "slides" as they are much like transparencies used for projection. Each is a complete screen image, i.e., 80 columns by 25 lines. Composite slides are made by stacking one or more slides together; in this way, highlighting, pop-up menus, windows, and data entry are simulated. For example, while an order entry form is displayed and the entry of data simulated, pop-up menus provide help, and windows are opened to check on the customer's credit status and inventory availability of the items being ordered. The running of a program is simulated by having user designated keystrokes control the transfer from one screen image to another. Thus, screens are not merely viewed in a sequential manner. Rather, use of the system is demonstrated since the sequence of screens viewed is contingent on what key the viewer (user) selects. (Student copies of DEMO were obtained from Software Gardens, Inc. for $25 each; list price was $75.)

Sequence of events. Eight calendar days elapsed from the introduction of the DEMO software to submission of the completed assignments. The key events were:

Day 1 Introduce the user interface design software
- Walk through an example
- Distribute the case and task assignment
- Organize teams
- Give teams copies of software and documentation
- Announce role play with owner manager for Day 4

All On-call expert help available

Day 4 Teams present interim designs to firm's owner/manager as role played by an experienced small business person

Day 5 and
Day 7 Help sessions in microcomputer lab (outside of class time)

Day 8 Task assignments collected

RESULTS

The completed projects. The completed assignments contained varying degrees of comprehensiveness and quality of design and considerable diversity in design approach. Teams constructed between eight and seventy different slides; the average was approximately thirty slides per team. Some completed task assignments were menu driven, some used pop-up menus or windows, and a few used color (color monitors were scarce).

The task assignments were evaluated on the degree to which the completed "system" met the evaluation criteria summarized below. (The evaluation criteria are included in Appendix B.)

EVALUATION CRITERIA

Were the identified order processing activities included? Could they be done in any sequence, and could they be done in the process of entering an order?

How well did the design deal with the problems or opportunities identified in the case?

How well were other customer needs, related to the desired firm objective of providing friendly service, incorporated into the design?

How well did they meet the consultant's stated desire to provide a "classy" demonstration which would obtain the order to provide the firm's information system? (Ease of use, visual appeal, and flexibility were considered here.)

Students' reaction. Students' reactions were positive with over three-fourths recommending inclusion of a similar user interface design task in future versions of the course. Additional information gained from anonymous questionnaires completed at the end of the task indicated:

(1) On average, each participant spent nineteen hours on the task, and the time was roughly allocated as follows:

30%	Learning the software
28%	Analyzing the case and preparing screen designs on paper
42%	Using the software to design screens

(2) At the completion of the task, over eighty percent felt proficient with the software, i.e., they required no more than occasional reference to the manual or help screens. Thirty-three percent were competent enough to rarely use the documentation or help screens. The help screens available in the DEMO software nearly preclude the need to use the accompanying manual.

To make the learning process more efficient, students recommended:

More elapsed (i.e., calendar) time for the assignment

Give students a chance to play with DEMO before assigning the case.

Give more examples or drill exercises, perhaps a couple of "leading by the hand" exercises.

Have a class session on how to control the sequencing and movement from slide to slide.

(3) Responses indicated the team size and case complexity were about right. For most a group size of two or three would be acceptable, although a few suggested groups of four and a few suggested individual work.

(4) Approximately one-half felt the interim presentation of their design to the owner/manager provided important value, helped improve their user interface designs, and added significantly to their understanding of the challenges faced by designers. A number of write-in comments indicated the additional realism provided by the role playing element.

It was good to get feedback from someone outside the group.

It made the prototype work sound professional.

Helped us understand the "real process" of user interaction.

No one commented that the role playing was not realistic, rather the realism of the role playing took some by surprise. One comment was particularly interesting as it captured the lack of real world perspective that some bring to graduate study in business. This student said:

It's irritating. Would rather have the limits--things wanted--laid out in writing.

(5) Three-fourths felt the exercise gave them worthwhile knowledge and understanding of user interface prototyping.

Write-in comments concerning the value of the knowledge and understanding of user interface prototyping gained included:

Excellent "vocabulary" for describing to programmers what needs to be accomplished for users

Appreciate what programmers have to do and the work involved in an application program

It is helpful to understand what a system should look like, without actually building it.

Prototyping is a powerful tool in systems development.

I learned how difficult or intricate the process of prototyping is.

You have to think clearly about what you want.

Write-in comments on ways they might use the skills included:

To design a system which is as explicit and user friendly as can be

Assessing user needs and requirements as a systems analyst

I can use DEMO to prototype my own needs, and then have the system done by someone else.

Write-in comments on ways managers might use the understanding and skills gained included:

Designing interfaces to show a programmer what is needed or how a program might be improved

To make a prototype and play with it until they identify what they really want

Provides an excellent vocabulary for describing to programmers what needs to be accomplished by users

In summary, students felt they gained useful knowledge about user interface design, user-designer interaction, and the process of prototyping. Further, they strongly recommended use of the task in the course in the future and offered suggestions to improve the task assignment and process of learning the software.

DISCUSSION

We found the students' favorable recommendation for future use of the user interface design task particularly significant as the task was introduced in a time of high pressure for the students due both to external and internal factors. First, the task, software, and team organization were essentially dropped on the students on Day 1. Second, the eight days of the task overlapped with midterm exams in other graduate courses. (While the abrupt introduction of the task was intentional--to limit the amount of total commitment to the task--scheduling the task to avoid overlap with midterm exams deserved more attention.) Considering that the students were essentially on their own to learn the software (except for the expert help available by telephone or in the two scheduled lab sessions) and that they had to complete the task during a time of high overall pressure, their favorable recommendation is encouraging.

From our perspective, the task experience offered many potential benefits, including:

(1) For students the user interface design task provides enhanced perspective on important information systems roles and concepts, e.g., users' and designers' differing perspectives, the challenge of establishing working relationships among members of the design team and between designers and users

(2) For managers and users in general, use of

user interface simulation software could significantly increase their power to clearly communicate their "needs" to designers.

(3) To analysts and consultants, user interface simulation software gives the ability to more clearly communicate potential designs or intended designs, i.e., by communicating with pictures instead of words in a specification document.

(4) To organizations, user interface simulation software offers

a. A rapid and inexpensive way to demonstrate the "information solution" to a problem

b. A reduction in specification error since the designer can talk with "live pictures" which simulate the intended design and demonstrate system usefulness

c. Reduced development cost as design errors are identified sooner and before major investment occurs

d. An accelerated process for gaining a common understanding of the design between users and designers

Some had difficulty handling the uncertainty introduced by role play. While it provided a useful experience of what it's like to work with a user, perhaps its value would be greater if participants had stronger advance appreciation of what to expect.

The task experience also identified a number of clear needs for future applications. Students' efficiency of learning the software must be improved so more of their energies can be devoted to completing designs of high quality. Few of the teams completed designs that incorporated all the assigned functions, and it was easy to identify many opportunities for higher quality presentation. Possibilities for accomplishing more efficient learning include:

Providing a 15-20 page concept guide
Providing simplified software documentation
Providing examples
Using a structured experience in the microcomputer lab to introduce the software
Extending the timeframe allowed for completion to approximately three weeks

Some concerns were also identified. The major ones deal with the level of design sophistication. On the one hand students' designs are limited by their knowledge of what constitutes good user interface design and of what can be done. On the other hand, students may not be able to recognize what is impossible, i.e., designs may be too visionary to realistically implement. Important considerations for implementations of user interface simulation software, therefore, are (1) helping students to grasp quickly an appreciation of advanced user interface designs, (2) providing them some guidelines on what the relative costs of constructing systems and with different types of interfaces.

CONCLUSION AND RECOMMENDATIONS

User interface simulation software provides a potentially efficient tool, which requires minimal resources, to give the general non-programming business student exposure to:

(1) Articulating information requirements by challenging them to take a business task, e.g., order processing, and rather explicitly lay out the information needed to perform the task. The real opportunity here is to make "information" a meaningful concept by requiring students to define the content, presentation style, and process of using the information to accomplish a given business function.

(2) The prototyping process by including a role playing-activity which provides them feedback on their design and gives them experience interacting with a user on a critique of their system design.

To achieve these benefits, improved ways are needed to reduce the setup costs, i.e., the student effort, course time, and instructor effort, to more efficiently get students over the hump of learning the software so more of their time is devoted to articulating information requirements. From this experience and the suggestions of those involved, we would propose:

(1) Allowing at least three elapsed weeks for completion of the task (As few as two class sessions devoted to the task during this period seem sufficient.)

(2) Incorporating a hands-on structured learning experience in a microcomputer laboratory to acquaint them with the software's capability to create individual and composite screens

(3) Providing a structured take-home learning experience to show them how to control the flow of screens and simulate a functioning system

(4) Incorporating user interface design into subsequent course assignments to capitalize on the proficiency developed and enhance students' understanding of information

While this implementation involved graduate students, we are confident that incorporation of the lessons learned will lead to successful future implementations with undergraduate students.

REFERENCES

Ahituv, N. and Neumann, S., Chapter 2: Psychological and Behavioral Aspects of Information Systems, in Principles of Information Systems for Management, 2nd Ed., W.C. Brown, 1986, pp. 13-73.

Kozar, K., Chapter 11: Designing for the User-Dialogs and Displays, in Humanized Information Systems Analysis and Design: People Building Systems for People, McGraw-Hill, forthcoming.

Bricklin, D., "Dan Bricklin's DEMO Program User Manual," Software Gardens, Inc., 1985, 29 pages.

Bean, Gary, Dan Bricklin's DEMO Program Tutorial, Software Gardens, Inc., 1987, 96 pages.

Ray, Garry, "A First Look at Ashton-Tate's dBASE IV, PC Week, Vol. 5, No. 7, February 16, 1988, pp. 1, 6.

APPENDICES
(Available from authors)

A Larry Byron Computer Systems Case

B Evaluation Criteria for User Interface Design Exercise

Computer Instructional Support Environment Use in Introductory Information Systems Courses

Dr. Marilynn Carol Hoeke
University of North Texas

ABSTRACT

Instruction in Information Systems fundamentals is rapidly becoming a foundation for undergraduate Business School curricula. Expanding computer use in corporations requires new approaches in academics to support student development of necessary skills and background knowledge. Familarity with the environment through its use in learning aids students in meeting needs in their careers. A series of experiments explores the effectiveness of three Instructional Support Environments in meeting two levels of academic learning objectives.

INTRODUCTION

Challenged to fully prepare undergraduates for futures in business, academic professionals are using Information Systems as study tools in many courses outside the discipline. It remains the responsibility of Information Systems departments to teach the fundamentals and familiarize students with the environment.

Research in educational use of computers reveals a growing concern about the role of computers and computer software in the learning process. Review of published studies reveals a pattern of single topic or complete course examination, primarily at the elementary school level. None of the studies presented, including those in Kulik, Kulik and Cohen's meta-analysis (4), focused on the effects of distinct computerized individual study presentation methods, instead evaluating the impact of the computer as one treatment type, regardless of application. These separate Computer Instructional Support Environments (CISE), CAI modules specifically designed to enhance a traditional lecture, need structured examination with regard to their effectiveness in meeting predetermined learning objectives for specific topics.

The absence of topic classification into learning objectives, and CISE module classification into presentation types in past research confuses statistical findings.

RESEARCH DESIGN

In order to realistically compare the effectiveness of selected individual study support treatments, it is necessary to provide a controlled observation environment. A pretest-treatment-postest design best supports measurement of relative improvement. This design also allows the researcher to check each topic's pretest results for sample population characteristics. For each topic included, the researcher identified its learning objective level, prepared support materials, created measurement pretest and postest questions, and checked all materials for correctness and consistency.

Six introductory MIS course topics, History of Computing, Systems Analysis and Design, File Organizations, Looping, Arrays, and Logic Structures, representing two fundamental levels of learning objectives (6), provide the content for this research. Repetition of each topic occurs in multiple sections of the closely coordinated course, thereby providing a larger sample size, and an opportunity to examine the effect of different instructors on student achievement across treatments.

EXPERIMENT DESIGN

A random lot drawing during each student volunteer's first participation in the continuing set of experiments, assigns treatment group. Each volunteer could attend any combination of the six available topic experiment sessions. Separations in these environments gave the researcher data subgroup identifications for the statistical analysis process.

Each experiment session used a pretest-treatment-postest research design for data collection. Immediately prior to the scheduled topic lecture, participants arrived at the experiment site for the pretest. This set of 25 multiple choice questions provides a measurement of achievement at the identified learning objective level for that topic. These questions also comprise the postest, in a second random order of presentation, with the related answer selection also randomized.

Following the pretest, all participants attend the scheduled class lecture on the topic under observation. After this department defined lecture, participants returned for the individual study session. At this time, students separate into their predetermined treatment groups. Each student evaluated the study support method after completion of the topic.

Three study support methods are under consideration. Textbook and class notes provide the student with a familiar presentation environment. Combinations of text and visual material cover important concepts and terms within each topic. At the conclusion of each chapter, questions give the student an opportunity to review topic information and determine whether s/he understood the presentation.

Computer instructional support environments (CISE) can use several different presentation types. Tutorials use text and graphics, and allow the student some interactive control over material displayed. Step by step construction of complex graphics, and explanations for each change in the display are possible in this environment. Questions, usually at the end of each section use structured review to allow the student an opportunity to evaluate his/her knowledge of a topic.

Computerized drill and practice presents multiple choice questions and related answers to the student in blocks of one to twenty-five, randomly selected from the prepared set. Following each answer evaluation, the system responds with an appropriate message regarding the correctness of the choice, and a short summary of the reasons for that answer.

The postest portion of this experiment provides a final measurement of ability or knowledge. Postest score evaluation, combined with the individual's preliminary score, computes a measure of relative improvement. Each student recorded his/her elapsed time for each individual phase; pretest, individual study, and postest. Elapsed time was a second measurement tool for evaluating learning achievement.

EXPERIMENT RESULTS

Students attended any combination of the six available topic experiments for that section. Average participation for students was three to four of the six topics. Each individual student did not necessarily attend sequentially ordered topics, or all topics within a specific learning objective. For each topic, an average total of 40 students participated across sections. Approximately a third of these student participated in any given study support method for a topic, although the distribution of treatment group sizes varied between sections and topics.

Frequency analysis results, supported by analysis of variance findings, display CISE drill and practice presentation with the highest levels of improvement across all topics. Textbook support of individual study, and the CISE tutorial environment also provide some average improvement, usually at lower levels.

TIME IMPROVEMENT

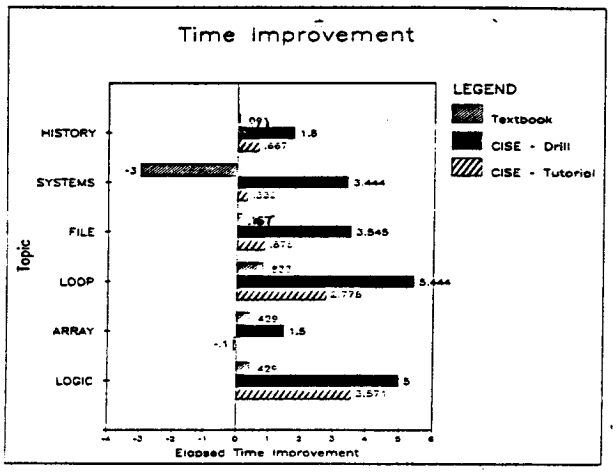


FIGURE 1 : Time Improvement

Improvement between pretest and postest elapsed time shows a distinct advantage in the use of CISE drill and practice study support across all topics. A decrease in the amount of time needed to complete the multiple choice questions indicates a clearer grasp of the definitions and characteristics for each topic at the learning objective level. Use of multiple choice questions to measure demonstration of understanding is less precise, though possible.

Textbook supported study has little apparent effect on the amount of time required to complete the test questions. CISE tutorial environments show little effect for the first three topics, requiring a demonstration of knowledge, although the results for two of the understanding topics are better than the mean results from the textbook group.

Analysis of variance results compute no significant interaction effects between combinations of topic and individual study treatment or learning objective and treatment. Study support tools have a 0.000 significance level of F, while topic and learning objective have significance levels of 0.023 and 0.039 respectively.

SCORE IMPROVEMENT

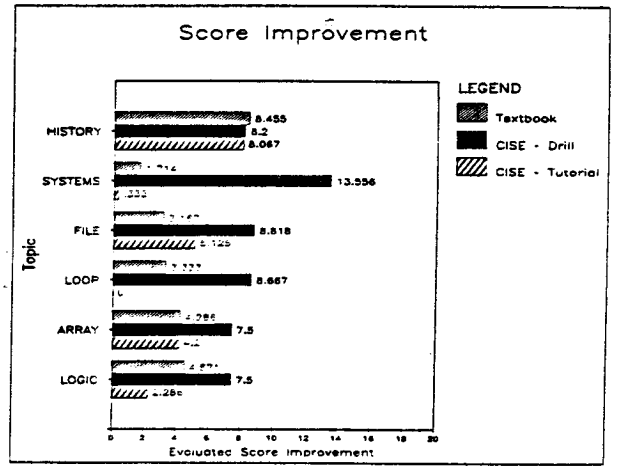


FIGURE 2 : Score Improvement

Mean values for each study treatment show little variation in score improvement for History of Computing. This may be due to students' familiarity with general history and how to approach learning new dates and developments. Of the remaining topics, textbook and CISE tutorial study support show smaller improvements than drill and practice.

Differences between treatment types are smaller for topics requiring demonstration of understanding than for demonstrating knowledge. This may indicate a problem with the measurement tool, which requires further investigation.

Results from the analysis of variance process indicate a significant interaction effect between topic and treatment type, while the interaction between learning objective and study treatment is not significant. Learning objectives have an individual significance level of F value at 0.001, with treatment has a level of 0.000.

STUDY TIME

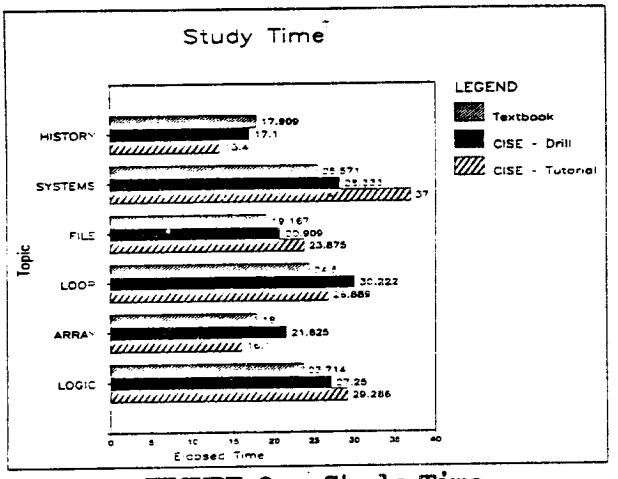


FIGURE 3 : Study Time

Differences in the treatment group means for individual study time are less extreme than those for time improvement. Observations during experiment sessions indicate that textbook and CISE tutorial participants tended to study the topic material without review or pause, while the CISE drill and practice users continued to repeat the random question sets until they attained a personally satisfactory level of achievement.

Analysis of variance calculations show a borderline significance level for the independent variables. The largest factor on time spent in individual study is the topic under consideration.

These initial patterns in data set results provide preliminary identification of treatment effects for topics and learning objectives. Further experiments are necessary to precisely identify contributions to the academic goals.

CONCLUSIONS

Because of the small sample size for treatment/topic data combinations, results from the statistical analysis remain only indications of underlying patterns. With this in mind, observed results support previous research findings regarding the effectiveness of computer supported study environments. Examination of the patterns provided by a multiple classification analysis revealed a tentative ranking of study support treatments from CISE drill and practice, to textbook and class notes, to CISE tutorial presentations. Knowledge, as a learning objective classification, showed positive deviations from the mean, however, topics again displayed no consistent characteristics.

Additional research into advanced topic and higher learning level study support methods, plus a thorough examination of the existing interactions between topics and treatments, or learning objectives and treatments, is needed. CISE presentations are providing useful learning achievement tools, but the best possible combinations of tool and goals are still waiting to be discovered.

SELECTED BIBLIOGRAPHY

(1) J. Aaron Hoko, "What is the Scientific Value of Comparing Automated and Human Instruction?", Educational Technology, February 1986, pp 16-19.

(2) D. Jamison, P. Suppes, and W. Wells, "The Effectiveness of Alternative Instructional Media: A Survey", Review of Education Research, 1974, pp 1-67.

(3) T. F. Jones, "Learning for and from the Micro: The use of Microcomputers by Non-specialists", Journal of the Operations Research Society, April 1983, pp 345-348.

(4) James A. Kulik, Chen-Lin C. Kulik, and Peter A. Cohen, "Effectiveness of Computer-based College Teaching: A Meta-Analysis of Findings", Review of Educational Research, Winter 1980, Vol 50, pp 525-544.

(5) R. G. Montanelli, "Using CAI to Teach Introductory Computer Programming", ACM SIGSUE Bulletin, 1977, Vol 7, pp 14-22.

(6) Jay F. Nunamaker, "Educational Programs in Information Systems", Communications of the ACM, March 1981.

(7) San-Yun Tsai and Norval F, Pohl, "Student Achievement in Computer Programming: Lecture versus Computer Aided Instruction", Journal of Experimental Education, Vol 46, No 2, Winter 1977, pp 66-70.

INCORPORATING INSTRUCTION ON QUALITY CONTROL ISSUES PERTAINING TO DOCUMENTATION INTO COMPUTER APPLICATIONS COURSES

Dr. J. K. Pierson, James Madison University
Dr. Karen A. Forcht, James Madison University
Dr. Faye Teer, James Madison University
Dr. Jack D. Shorter, University of Wisconsin--Eau Claire

Quality control issues related to the use of application packages for systems development include data validation, documentation, testing, operating control, audit requirements, and backup and recovery. This paper presents the results of a recent study on factors affecting the level of documentation requirements and the types of documentation recommended for user-developed applications and presents recommendations for incorporating quality control guidelines on documentation into courses requiring applications development with common user development tools.

INTRODUCTION

As the phenomena of end-user computing and user-developed applications continues to grow, concerns have increased regarding the management of such applications, concerns that include quality control. Quality control issues related to the use of application packages for systems development include data validation, documentation, testing, operating control, audits, and backup and recovery. The purposes of this paper are: (1) to present the results of a recent study on documentation of user-developed applications regarding factors affecting the level of documentation requirements and the types of documentation recommended for user-developed applications and (2) to make recommendations for incorporating quality control guidelines on documentation into courses requiring applications development.

STUDY ON DOCUMENTATION OF USER-DEVELOPED APPLICATIONS

A study was recently completed to identify the factors affecting the level of documentation required for user-developed applications and the types of documentation recommended. Fifteen factors were selected as candidate factors that might affect the level of documentation requirements of user-developed applications, and a questionnaire was developed to elicit reactions to those factors. Thirteen types of documentation were also chosen after reviewing the literature on user-developed applications and discussing them with information systems practitioners. Participants were asked to check the types of documentation they thought were important for each of six different categories of user-developed applications.

The questionnaire was field tested by members of a regional association of information center personnel from the public and private sectors. The final version of the questionnaire was then distributed to 135 management personnel associated with user computing in the New York City area.

Fifty-four questionnaires were returned, a response rate of 40.0 percent. Of those, 46 were completed to an extent considered valid. The majority of the respondents work in manufacturing organizations or in financial, insurance, or real estate institutions that employ over 5,000 people. Over

half of the respondents listed their job title as Information Center manager, assistant manager, project leader, or consultant.

FINDINGS (PART I): FACTORS AFFECTING LEVEL OF DOCUMENTATION

Respondents were asked to rate the fifteen candidate factors that might affect the level of documentation required for a user-developed application. A five-point Likert scale ranging from 1 for "very unimportant" to 5 for "very important" was utilized. The responses were used to produce an "importance score" for each factor by totalling responses of 3, 4, and 5 on the scale. The importance scores, ranking, and standard deviations of the factors is shown in Table I.

The mean response for all factors but one fall above the midpoint on the Likert scale, an indication that those factors are considered to have an impact on the level of documentation requirements for user-developed applications. The rankings and response means provide an indication of the relative importance placed on the factors by the user-computing managers who participated in the study.

FINDINGS (PART II): TYPES OF DOCUMENTATION RECOMMENDED

The questionnaire described above was also used to gather opinions on thirteen types (forms) of documentation. The candidate documentation gleaned were gleaned from literature reviews and from suggestions by information systems practitioners and information center personnel. The thirteen documentation types are listed in Table II.

The classifications of user-developed systems developed by Rivard and Huff were used for this study. The taxonomy has six application categories also shown in Table II.

Respondents were asked to check the types of documentation (user's instructions, narrative description, etc.) that they recommended for user-developed applications with high documentation requirements in each of the six application categories.

Rankings of the documentation types for each of the six application types (modelling, data analysis, etc.) were constructed from frequency data. In addition, Cochran's Q tests were used to determine (1) if there are significant differences in recommendations for the thirteen types of documentation within each of the six application categories, and (2) if the recommendations for each documentation form differ significantly across the six application categories. Table II contains frequency data, rankings of recommended documentation types for each of the six application categories (in parentheses), Cochran's Q statistics, and significance levels.

Cochran's Q test of significance was used to determine if the frequency of recommendations of documentation types vary significantly in each of the six application categories. The Cochran Q and significance statistics at the bottom of the column for each application type indicate that not all documentation types are recommended at the same frequency---that some documentation types are preferred over others in each of the six categories of applications.

Further analysis of the data in Table II indicates that the mean number of documentation types recommended varies from one category of application to another. Transaction processing applications have the greatest number of documentation types recommended; simple query applications have the least.

The rankings of documentation types favored for each of the application categories are given in parentheses. User's instructions ranks first or ties for first in all application types except graphics. Hardware/software requirements is ranked

in first place for graphics applications.

In last place (13th) or tying for last place (12th) is history of development and maintenance. Other types of documentation have different rankings across the six applications categories.

Cochran's Q test of significance was also used to determine whether or not the number (frequency) of recommendations for each documentation form differed significantly at the one percent level from one application category to another. An asterisk beside the significance level indicates that the frequency of recommendations for the documentation form differ significantly across application types.

Recommendations for four types of documentation--user's instructions, narrative descriptions, input data descriptions, and output descriptions--do not vary significantly from one category of application to another. Recommendations for the other documentation types do vary significantly from one application application to another, indicating that not all documentation types are deemed equally appropriate for all application categories.

RECOMMENDATIONS

The following recommendations relate to the inclusion of instruction on documentation guidelines in collegiate courses that require applications development by students similar to applications developed by end users in the workplace.

1. Courses should include instruction on documentation as one of the quality control issues associated with user-developed applications.
2. Because of the importance placed on factors affecting the level of documentation identified in this study, application development assignments similar to applications developed by end-users in the workplace should carry an additional requirement--that of evaluating the application's level of documentation requirements according to the thirteen factors identified in this study.
3. Students should be required to document their applications that carry a high level of documentation requirements with the following: user's instructions, narrative descriptions, input data descriptions, and output descriptions. Graphics applications should always include hardware/software requirements as part of their documentation.

RANKING OF
FACTORS AFFECTING LEVEL OF DOCUMENTATION REQUIREMENTS

Ranking	Importance Score	Response Mean	Std. Dev.	Factor
1	185	4.13	.909	Maintenance requirements
2	181	4.13	1.087	Use of output by another system
3	180	3.98	.954	Financial impact
4	174	3.93	.986	Scope of application--personal, departmental, or organizational
4	174	3.96	1.134	Complexity of application
6	165	3.74	.999	Turnover of user personnel
7	163	3.84	1.261	Security requirements
8	162	3.74	1.250	Audit requirements
9	160	3.69	1.221	Type of application--modelling, data analysis, simple query, report generation, transaction processing, or graphics
10	141	3.49	1.100	Number of locations of use
11	135	3.33	1.076	Number of users
12	132	3.30	1.030	Life expectancy of application
13	131	3.24	1.139	Development tool used
14	128	3.28	1.047	Anticipated reuse of application
15	111	2.98	1.085	Frequency of use of application

DOCUMENTATION RECOMMENDED FOR APPLICATIONS WITH HIGH DOCUMENTATION REQUIREMENTS
(Rankings Given in Parentheses)

Documentation Type	Application Category: Modelling	Data Analysis	Simple Query	Report Prep.	Transact. Processing	Graphics	Across Application Category: Cochran's Q	Sig.
User's Instructions	42 (1)	40 (1)	40 (1)	43 (1)	43 (1)	42 (2)	3.3333	.6487
Narrative Description	42 (1)	39 (2)	33 (3)	42 (2)	41 (6)	41 (3)	13.5859	.0185
Input data source, description, and/or layout	39 (3)	36 (3)	34 (2)	39 (3)	43 (1)	38 (4)	9.2202	.1006
Output destination, description and/or layout	39 (3)	36 (3)	32 (4)	39 (3)	43 (1)	38 (4)	15.0000	.0104
Program code, procedure and/or formula listings	37 (5)	34 (7)	24 (7)	34 (7)	41 (6)	29 (6)	28.3862	.0000*
Internal documentation such as remark statements or labels	34 (6)	35 (5)	20 (11)	35 (5)	39 (11)	27 (7)	32.0647	.0000*
Testing procedures and/or test data	33 (7)	35 (5)	24 (7)	31 (10)	42 (5)	26 (9)	34.7872	.0000*
Hardware/software requirements	32 (8)	33 (9)	28 (6)	33 (9)	36 (12)	43 (1)	23.6093	.0003*
Audit trail requirements	32 (8)	33 (9)	23 (9)	34 (7)	41 (6)	24 (10)	30.2792	.0000*
Security requirements	30 (10)	34 (7)	29 (5)	35 (5)	43 (1)	27 (7)	25.2424	.0001*
Design documents such as data flow diagrams, flowcharts, pseudocode	28 (11)	31 (11)	20 (11)	31 (10)	41 (6)	23 (11)	29.9145	.0000*
Impact Analysis	27 (12)	29 (12)	21 (10)	30 (12)	40 (10)	22 (12)	34.3401	.0000*
History of development and maintenance—authors, dates, etc.	25 (13)	29 (12)	18 (13)	24 (13)	34 (13)	21 (13)	24.0659	.0002*
Within Application Category: Cochran's Q	53.3428	43.9850	72.2832	70.5205	58.1013	166.2509		
Within Application Category: Sig.	.0000*	.0000*	.0000*	.0000*	.0000*	.0000*		
	N = 45	N = 43	N = 46	N = 45	N = 45	n = 44		
Mean Number of Forms Recommended	9.78	10.32	7.52	10.0	11.71	9.11		

* $p < .01$

Predicting Success In Introductory Computer Information Systems Courses

James A. Nelson

College of Business Administration & Economics
New Mexico State University, Las Cruces, NM 88003
Telephone: (505) 646-4901 BITNET: BSA047 @ NMSUVM1

ABSTRACT

Grades in an introductory Business Computer Systems course of 128 students were predicted using a discriminate analysis of factor analyzed semantic differential scales. The analysis successfully classified 78% of the students with passing examination grades (A, B, or C) from their responses to semantic differential scales of the concept "COMPUTER". Suggestions are made for further research.

As our society has become more computer dependent, institutions across the country have begun to offer specialized curricula for various computer applications. One can readily find baccalaureate granting colleges and universities that offer degrees in computer science, computer engineering, human-factors engineering, computer aided design and manufacturing, and computer information systems. With such expanded emphasis and opportunities, more college students are choosing to take courses in computer related disciplines. One such program is the Data Processing Management Association's Computer Information Systems (CIS) curriculum for schools of business incorporating the American Assembly of Collegiate Schools of Business (AACSB) common body of knowledge. This increase in demand, combined with a continuing concern over student retention and limited institutional resources, is causing university personnel to seek valid ways of predicting which students are most likely to succeed in computing related fields.

Alspaugh (1) predicted performance in a programming course with independent variables of: scores on the IBM Programmer's Aptitude Test, the Thurstone Temperament Schedule, the SCAT Quantitative and Verbal Subsets, the Watson-Glaser Critical Thinking Appraisal, and a coding system for mathematical background. Alspaugh used 50 students in his study and three measures of programming proficiency: a FORTRAN score, an assembly language score, and a total of the two scores. Using 9 or 10 independent variables selected from a total of 18, he was able to explain from 33 to 40 percent of the variance in programming proficiency. For all the dependent variables, the mathematical background code that attempted to interpret level of high school and college mathematics appeared to be the most important independent variable.

Peterson and Howe (21) found general intelligence score and college GPA to be important predictors as to the grade received in an introductory computer science course. Mazlack (16) concluded in his study that "future programming skill is not predictable" by the IBM Programmer's Aptitude Test. The simple correlation between total PAT score and FORTRAN grades would only explain about 11 percent of the variation in course grade. Such a small percent validates Mazlack's assertion that the PAT score is not a valid predictor of skill in and of itself.

Several studies reveal that students of high general ability perform well in programming classes. One study of the relationship between ability and performance in a programming class was carried out by Kurtz (14). He based his study on the Piagetian theory of intellectual development. He used several standard measures of intellectual development to assess student abilities. Kurtz found that students' intellectual ability did correlate with performance in the course.

A logistical classification model was developed by Fowler and Glorfeld (9) in an attempt to classify students in an introductory programming course.

High aptitude students were those receiving an "A" or "B" grade while low aptitude students received "C", "D", or "F" grades. The classification model used college grade point average, number of math courses, SAT math score, and age as independent variables. The model correctly classified 81 percent of the students. Results indicated that college GPA was the most important independent variable.

Konvalina, Stephens, and Wileman (13) were able to account for 19 percent of the variation in final exam scores for an introductory computer science course. They concluded in their study that high school mathematics background, overall high school performance, and exposure to high school computer course work were directly related to performance in college level computer science courses.

Recently, Butcher and Muth (3) used first-semester freshmen in a study designed to predict both performance in an introductory computer science course and first-semester college grade point average. High school coursework, GPA, class rank, and ACT scores were used as independent variables in the study. Their results showed that 36.6 percent of the variation in course grade could be explained by the relationship of any two of the following three variables: high school GPA, ACT Math, and/or ACT Composite. Most of the variation in exam scores (63.4%) remained unexplained.

One area that has been relatively neglected is the relationship of attitudes towards computing and success in using computers. Thoughtful scholars, such as Jacques Ellul (8), have reasoned that there is a difference between technology and "la technique". According to Ellul, "la technique" is a strong set of underlying values which most people hold, and will be induced to hold, in order to survive in the organization. Technology today is almost synonymous with computers. Perhaps students need an underlying set of values about computers in order to be successful programmers. Ellul does not attempt to measure or operationalize his concept of "la technique". One widely used measure of "meaning" or underlying values is the semantic differential. We attempt to measure these underlying values through factor analysis of semantic differential scales of the concept "COMPUTER". If these values of "la technique" are necessary to survive in the organization (CIS course), then they should relate to grades in the course.

Osgood, Suci, and Tannenbaum (19) developed the semantic differential to measure what meaning a concept might have for people by rating the concept on bi-polar adjective scales, then extracting dimensions of meaning through factor analysis. Early studies demonstrated the effectiveness, reliability, and efficiency of using the semantic differential for a wide range of applications. The present study attempts to predict examination grades from students responses to semantic differential scales for the concept COMPUTER.

METHOD

A total of 128 university students enrolled in Business Computer Systems (BCS) 338, "Introduction to Business Computer Systems", were used as subjects. Of the 128 subjects, 80 were Anglo, 37 were Hispanic, and 11 were Native American, 63 of the subjects were male and 65 female. The median age was 24 years. BCS 338 is an extension of the material presented in DPMA CIS 1 with equal emphasis on mainframe and microcomputer applications. Students study hardware, software, and systems concepts and learn Lotus 1-2-3, WordPerfect on IBM PCs and on the mainframe, CMS, XEDIT, Script/DCF, and SAS. Junior standing and CIS 1 (taught in the computer science department) are prerequisites for all students taking BCS 338, and the students must have also completed, or demonstrated competency in, intermediate algebra, calculus, finite math, and freshman English composition.

On the first day of the semester responses were made by the subjects using 32 seven-point semantic differential scales. The scales are listed in the first column of Table 1. The polar adjectives that composed the scales were chosen on the basis of open-ended responses to technology and previous research using the semantic differential (19). The order of presentation of the scales and the polarity of the adjective pairs was randomized.

The first page of the measuring instrument provided a detailed set of instructions on using the semantic differential scales. At the top of the next page was the stimulus word COMPUTER. Immediately following were the 32 scales for evaluating the stimulus word. Each subject recorded their evaluations on a mark-sense form, that was subsequently scored by computer.

The results of the evaluations on the 32 semantic differential scales were analyzed using principal components factor analysis with varimax rotations and examination grades as the dependent variable.

RESULTS and DISCUSSION

To determine the dimensionality of the evaluations of the concept COMPUTER, the intercorrelations among the 32 semantic differential scales were factor analyzed. An analysis of the factor structure

showed that a five factor solution exhausted the meaningful variance. The results of the factor analysis appear in Table 1.

The first factor is defined at one end of a continuum by valuable, good, active, honest, safe, fresh, and fair. At the other end of the continuum for the first factor are negative attributes of worthless, bad, passive, dangerous, stale, and unfair. This appears similar to the evaluative factor reported by Osgood and Suci (19).

The second factor is defined by informed, educated, strong, happy, and rugged at one end of the continuum and by uninformed, uneducated, weak, sad, and delicate at the other end. This dimension appears to reflect Ellul's (8) concept of "technical rationality"

Factor three is an aesthetically pleasant dimension defined by the scales sweet-sour, beautiful-ugly, new-old, pleasant-unpleasant, open-closed, and deep-shallow. It evokes an image of "hi-tech" as not just equipment, but as art.

Table 1

ROTATED FACTOR PATTERN

	FACTOR I	II	III	IV	V
VALUABLE-WORTHLESS	0.63	-0.29	0.03	0.00	0.09
GOOD-BAD	0.63	-0.05	-0.10	0.33	0.03
ACTIVE-PASSIVE	0.53	0.01	-0.05	-0.18	-0.37
HONEST-DISHONEST	0.52	0.08	-0.38	-0.13	-0.05
SAFE-DANGEROUS	0.52	0.08	0.07	0.08	0.08
FRESH-STALE	0.49	0.28	-0.10	0.07	0.18
FAIR-UNFAIR	0.39	0.13	-0.12	-0.04	0.06
INFORMED-UNINFORMED	0.02	0.79	0.02	-0.10	0.00
EDUCATED-UNEDUCATED	-0.02	0.77	-0.02	-0.10	-0.08
STRONG-WEAK	0.28	0.50	0.02	-0.05	0.44
HAPPY-SAD	0.14	0.45	-0.24	-0.10	-0.11
RUGGED-DELICATE	-0.05	0.30	0.17	0.18	-0.23
SWEET-SOUR	0.17	-0.05	0.71	-0.01	0.17
BEAUTIFUL-UGLY	-0.14	0.06	0.69	-0.08	-0.16
NEW-OLD	-0.12	0.16	0.61	0.05	-0.07
PLEASANT-UNPLEASANT	-0.04	-0.15	0.60	0.14	0.07
OPEN-CLOSED	-0.23	-0.20	0.47	-0.33	0.26
DEEP-SHALLOW	-0.01	-0.27	0.39	-0.26	-0.15
EXPERIENCED-INEXPERIENCED	-0.21	-0.37	-0.04	0.57	0.09
OBJECTIVE-SUBJECTIVE	0.06	-0.17	0.15	0.52	0.04
TRAINED-UNTRAINED	-0.24	-0.12	-0.17	0.48	0.11
FANCY-PLAIN	0.17	-0.13	0.01	0.47	0.09
LIGHT-HEAVY	0.01	0.24	-0.00	0.43	0.00
RELAXED-TENSE	0.14	0.28	-0.01	0.40	-0.03
MASCULINE-FEMININE	0.26	-0.16	0.13	0.35	-0.15
SOFT-HARD	0.15	-0.26	0.14	-0.34	0.24
FAST-SLOW	0.31	0.15	0.18	-0.10	0.54
CLEAN-DIRTY	0.12	-0.03	-0.29	0.14	0.53
EXTROVERTED-INTROVERTED	-0.16	-0.05	-0.07	0.21	0.52
HOT-COLD	-0.19	-0.04	0.08	-0.04	0.44
FRANK-RESERVED	0.18	0.03	0.17	-0.24	0.40
COMPLEX-SIMPLE	0.14	-0.18	-0.04	0.07	0.36

VARIANCE EXPLAINED BY EACH FACTOR

I	II	III	IV	V
0.08	0.08	0.08	0.07	0.06

PERCENT OF COMMON VARIANCE

I	II	III	IV	V
21.9	21.5	21.5	18.2	16.8

Factor four reflects a dimension of scientific objectivity as defined by the scales experienced-inexperienced, objective-subjective, trained-untrained, fancy-plain, light-heavy, relaxed-tense, masculine-feminine, and hard-soft.

The final factor is defined on one end of the continuum by fast, clean, introverted, hot, frank, and complex. At the other end of the continuum are slow, dirty, extroverted, cold, reserved, and simple. This factor appears to reflect a high technology dimension.

The five factors accounted for only 38 percent of the total variance, with each individual factor accounting for less than 10 percent.

Based on the factor analysis the following scales are suggested for use in measuring the concept COMPUTER. The three highest loading scales for each of the five factors are used and the averages are listed in Tables 2.

Table 2

FACTOR	MEAN	S.D.
I	2.17	0.90
II	2.82	1.46
III	4.30	0.97
IV	4.15	1.43
V	2.80	0.99

I. Evaluative: valuable-worthless, good-bad, and active-passive.

II. Technical rationality: informed-uninformed, educated-uneducated, strong-weak.

III. High-tech aesthetics: sweet-sour, beautiful-ugly, new-old.

IV. Scientific objectivity: experienced-inexperienced, objective-subjective, trained-untrained.

V. High-tech: fast-slow, clean-dirty, introverted-extroverted

Scores from a multiple-choice examination over class lecture and textbook materials (hardware concepts) were then used in a discriminant analysis with the five factor scores. Examination scores were separated into two categories, students with grades of A, B, or C (ABC) and students with grades of D or F (DF). Table 3 shows the results of this analysis.

Table 3

NUMBER OF OBSERVATIONS AND PERCENT CLASSIFIED INTO EXAM GRADES

		ACTUAL EXAM GRADES ABC	DF	TOTAL
PREDICTED EXAM SCORES	ABC	71 78%	20 22%	91 100%
	DF	8 29%	15 71%	21 100%
	TOTAL PERCENT	77 69%	35 31%	112 100%

Grades of A, B, or C were correctly predicted for 78% of the subjects on the basis of their five factor scores. For subjects with an examination grade of D or F, the factor scores correctly classified 71%. These results support Ellul's thesis that a common set of underlying values towards technology are necessary to survive in modern organizations. If "survival" is defined as passing an examination and "organization" as the classroom, then Ellul is correct.

Further research will extend this study to programming courses and to courses of a less technical nature.

REFERENCES

(1) Alspaugh, C.A. (1972). Identification of some components of computer programming aptitude. Journal of Research in Mathematics Education,3, (9), 89-98.

(2) American College Testing Program, The. (1979). ACT Evaluation Survey Service. Iowa City, Iowa.

(3) Butcher, D.F. and Muth, W.A. (1985). Predicting performance in an introductory computer science course. Communications of the ACM,28, (3), 263-268

(4) Campbell, P.B. (1984, November). Computers in bilingual classes: Effects of sex and language dominance. Paper presented at the American Educational Research Association: Special Interest Group on Women and Education Annual Meeting, Long Beach, CA.

(5) Campbell, P.F. and McCabe, G.P. (1984). Predicting the success of freshmen in a computer science major. Communications of the ACM,27, (11) 1108-1113.

(6) Capstick, C.K., Gordan, J.D. and Salvadori, A. (1975). Predicting performance by university students in introductory computing courses. SIGCSE Bulletin,7, (3), 21-29.

(7) Dalby, J. and Linn, M.C. (1985). The demands and requirements of computer programming: A literature review. Journal of Educational Computing Research,1, (3), 253-274.

(8) Ellul, J. (1965). The technological society. Knopf, New York.

(9) Fowler, G.C. and Glorfeld, L.W. (1981). Predicting aptitude in introductory computing: A classification model. AEDS Journal,14, (2), 96-109.

(10) Hess, R. and Muira, I. (1983). Sex differences in computer access. The Forum for Academic Computing and Teaching Systems,2, 91-97.

(11) Hoc, J.M. (1977). Role of mental representations in learning a programming language. International Journal of Man-Machine Studies,15, 87-105.

(12) Hostetler, T.R. (1983). Predicting student success in an introductory programming course. Proceedings of the NECC5. Silver Spring, MD.

(13) Konvalina, J., Stephens, L., and Wileman, S. (1983). Identifying factors influencing computer science aptitude and achievement. AEDS Journal,16, (2), 106-112.

(14) Kurtz, B. (1980). Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. ACM SIGCSE Bulletin,12, 110-117.

(15) Maddala, G.S. (1977). Econometrics. New York: McGraw-Hill.

(16) Mazlack, L.J. (1980). Identifying potential to acquire programming skill. Communications of the ACM,23, (1), 14-17.

(17) Miller, L. (1981). Natural-language programming: Style, strategies, and contrasts. Perspectives in Computing,1, 22-33.

(18) Nelson, J. A. & Scranton, B "Predicting success in the introductory COBOL course", Proceedings of the Information Systems Education Conference,San Francisco, 1987, pp.

(19) Osgood, E.E., Suci, G.J., & Tannenbaum, P.H. The measurement of meaning.University of Illinois Free Press, 1957.

(20) Ott, L. (1978). Admissions management with the focus on retention. New Directions for Student Services,3, 23-28.

(21) Peterson, C.G., and Howe, T.G. (1979). Predicting academic success in introduction to computers. AEDS Journal,12, (4), 182-191.

(22) Ralston, A. and Shaw, M. (1980). Curriculum 78 - is computer science really that unmathematical? Communications of the ACM,23, (2), 67-70.

REFLECTIONS ON THE PSYCHOLOGY AND EDUCATION OF PROGRAMMERS
Notes from a Graduate Seminar

Judith D. Wilson
Department of Mathematics and Computer Science
Xavier University, Cincinnati, OH 45207

ABSTRACT

Informal discussions in a graduate seminar on programmer productivity and psychology are described. These discussions were based on experimental research that investigates how programmers work and how they should be trained. While this research is providing new insight into the psychology of how programmers work, there is still little understanding of how skilled programmers solve the analysis and design problems that are central to large programming tasks outside the classroom.

INTRODUCTION

The general problem of programmer productivity and the psychology of programmers was the agenda for a recent graduate Information Systems seminar. A primary objective of the seminar was to find answers to the question: Is there a programmer psychology and, if so, can it explain differences in the quality of programming performance? Answers to this question would be relevant to how programmers should be educated. Readings included a number of papers from the empirical literature on programmers and a collection of informal interviews of highly successful programmers in industry. The seminar format was discussion-driven with one or two papers from the readings designated as focal reading for each session.

Half of the seminar participants were MBA students engaged in developing software as programmers or managers for local industries. The remaining participants were information systems instructors or Ph.D. students. This mix of participants led to informative discussions about programmer psychology and training that involved both academic and nonacademic perspectives. Several recurrent discussion topics are reported in the following sections, with an overview of the seminar readings provided first.

THE LITERATURE

The empirical literature surveyed in the seminar can be grouped according to several general categories. The discussion here concerns research of the following two types:

1. Development of cognitive models of programmers. The studies by Adelson [1], Brooks [3], McKeithen, et al [7], Ratcliffe and Siddiqi [8], Shneiderman [13], Soloway [12], Vessey [14] and Weiser [16] are included in this category.

2. Pragmatic or pedagogical recommendations based on these models. The papers by Mayer [6], Sheppard [10], and Soloway [13] come under this category.

Collectively, this research is beginning to converge on a model of how programmers work, and it is also providing some clues about how background and experience can effect programming performance.

MODELS OF HOW PROGRAMMERS WORK

The research provides evidence that experienced programmers, like experts in other fields, have a large knowledge base, or repertoire, of domain-specific "chunks" which are used to organize the input stimulus and solve problems. For programmers, these chunks are abstract procedures, production rule sets or "plans" [1, 3, 9, 11, 13]. Such abstractions, which I will refer to as "procedural templates", are organized hierarchically so that they can be accessed quickly for the purpose of building and understanding computer programs [8, 14]. There is also some evidence that the expert programmer's ability to tap this knowledge base can be undermined by programs that violate the programmer's expectations (the tacit "rules of programming discourse" [12]) and by convoluted or unstructured programs [10].

By contrast, novice programmers are found to attend to literal or syntactic features of programs and problem descriptions [11, 14, 16] and to organize their knowledge according to common language associations with program features [6].

The study by Vessey indicates that expert programmers are sensitive to context when reading a program, while novice programmers rely on preconceived models of program function [14]. Vessey also finds that experts discard hypotheses about program error more readily and are thus more flexible when searching for program errors than are novices. This kind of flexibility is shown by expert information systems designers as well [15]. The importance of context sensitivity to effective error-tracking is also indicated by a study described by Littman and others which finds that those programmers who are most successful at error-tracking proceed by first developing a systematic, global view (or "simulation") of how the entire program works [5].

HOW EXPERIENCE EFFECTS SUCCESS

Several experimental studies indicate that variations in performance among experienced programmers is associated with the quality and not the amount of programming experience. Ratcliff and Siddiqi find that specific training in decomposition techniques (such as Jackson structured programming) can be effective in developing abstraction skills in programmers [8]. Such skills are needed to develop the expert programmer's repertoire of procedural templates. Sheppard and others find an association between variety of programming experiences and languages, and successful performance which indicates that novice programmers benefit from exposure to a diversity of languages, software development tools, and problem types [10].

Mayer describes several experiments which imply that verbalization can help beginners extract conceptual meaning from syntactic structures [6]. The benefits for beginners of verbalization may be linked to the benefits of systematic, global program simulation for experienced programmers. In both cases, programmers construct a model of program behavior, but the novice must rely on common language. This may be why one researcher finds verbal aptitude correlated with the ability of beginning programmers to successfully create programs [2].

THE DISCUSSIONS

Many issues regarding the training of programmers were addressed during seminar discussions. Most of these issues fall under three general topics: defining the programming task, defining who the programmer is, and determining the most effective methods for training programmers for better performance.
Before discussing these topics, it is important to observe that two distinct educational concerns occupied seminar

participants. The nonacademic participants were interested in the psychological differences between very successful (above average) and less successful (below average) experienced programmers, and often found fault with the experimental research on the ground that relatively inexperienced programmers, students with little or no real world experience, were used as subjects.

In contrast, the problem of achieving programming literacy had greater interest for those who had taught undergraduate lower division programming courses. These instructors focused on the problem of helping novice programmers perform more as experienced programmers do, and their main concern was to discover systematic differences in cognitive approaches to problem solving between novice and more experienced programmers.

Problems of definition and focus were common in seminar discussions due to these differences in concern. In many cases, seminar participants could not reach a consensus on the implications of particular experiments, or on how the central concepts should be defined, or even on what the underlying problems really are.

1. The Programming Task

Seminar participants soon realized the need to define what is meant by the programming task. Unless this is done, it will not be possible to determine what specific experimental studies can tell us about the psychology of programmers. Few researchers explicitly identify the aspect of the programming activity that their experiments address, or distinguish this aspect from others which the research does not examine. In many of these cases, the experimental results have implications for only a small part of the overall activity.

It is possible to distinguish at least three interrelated yet distinct programming tasks, (1) problem analysis, (2) program synthesis and (3) program comprehension (which is required for program modification and error-tracking). Most experimental investigations of how programmers work isolate skills used primarily in program comprehension tasks.

Less work has been done on how programmers handle problem analysis and program synthesis tasks. Brooks' mechanical model addresses the code generation phase of programming and permits speculation about why expert programmers may be better and faster coders than non-experts based on the size of their code-specific knowledge base [3]. Seminar participants agreed, however, that Brooks' cognitive model has limited explanatory value for distinguishing among experienced programmers. Experienced programmers who are competent at coding when given a set of specifications may not be competent at designing solutions to problems. In other words, experienced programmers can be "good coders" without being "good designers". Seminar participants with nonacademic experience reported that experienced programmers differ most in their ability to understand problems and to map this understanding into solution methods.

Weber hypothesizes that the expert designer is assisted by a large propositional network of "declarative knowledge" and by mental imagery ("analog representations") as well as by extensive procedural knowledge [15]. It may not be possible or enlightening to model all such skills in a mechanical way.

The work by Ratcliff and Siddiqi concerns the design phase of the programming task. The experiments they describe indicate that problem analysis and program design are closely related [8]. Weber describes these two activities as two stages in a single linked process which occur concurrently for expert problem solvers [15]. The

experienced programmer is skilled at analyzing programming problems in terms of a hierarchy of goals and subgoals that can be accomplished using procedural templates that are also hierarchically organized. However, the empirical studies surveyed do not provide much insight into how these precoding activities are accomplished, and hence how programmers can be better trained to handle them.

2. The Programmer.

Much of the empirical research surveyed in the seminar is concerned with explaining how programmers solve problems. Yet it is not clear in most cases whether the research objective is to model the average programmer or the unusually talented programmer. Many researchers attempt to achieve their objective by developing contrasting models of the novice and the more experienced programmer. Moreover, much of this research appears to assume that there is a gradual continuum from novice to expert programmer and that differences between more and less experienced programmers on the novice end of this continuum can be generalized to account for differences between all groups of programmers.

In part this is a problem of definition. Are expert programmers, by definition, the few in the profession who excel or should the label refer to any experienced programmer with average or better performance? Seminar participants could not agree on this question. Perhaps this kind of question can be answered only in reference to experimental objectives, and these seem to point to identifying qualitatively superior performance and not simply average performance [1, 7].

The next problem is how expert programmers are to be identified. One strategy is to rely on the judgment of managers or teachers. In such cases, subjective or irrelevant factors usually play a dominant role. For example, a manager may favor a programmer who has been with the company for many years, or one who quickly solves domain-specific problems using unmodifiable code. Or a teacher may favor a student who with great, and transparent effort, complies with every assignment detail.

Another approach is to use objective criteria, such as the time to finish a programming task, or the number of errors made, or the quality of the algorithm used. Some of these may be problematic in experimental settings, however. For example, time constraints imposed by an experiment may effect novice and experienced programmers differently. Under the pressure of time constraints, novices might be more prone to select suboptimal algorithms than they would otherwise.

Only Vessey addresses the problem of defining and identifying expert programmers [14]. She compares a classification scheme based on evaluations made by managers with one based on evidence of skills generally recognized in the literature as associated with programming expertise. Not surprisingly, she finds that the performance factors are significantly better predictors of expert behavior in error-tracking problems.

3. Training Programmers.

The empirical research has undeniable implications for how academic programs can help develop programming expertise in novice programmers, at least in the coding phase of program synthesis. A first step would be to provide good examples of procedural templates, perhaps identifying them with descriptive labels as Mayer [6] and Soloway [13] advise. It is also likely that exposure to different programming languages and a variety of programming problems will help the novice programmer abstract procedural templates from the syntactic features of programs written in languages like COBOL, Pascal and C.

Seminar participants who had taught programming to college undergraduates remarked that novices tend to fixate on the syntax of their first programming language. Since informal verbalization appears to help beginners learn programming abstractions more effectively [6], it is reasonable to expect that novice programmers in general will acquire a knowledge base of quality procedural templates more quickly if they are encouraged to explain verbally, and in their own terms, both the goals to be achieved and the mechanism by which they are to be achieved.

These pedagogical techniques may help train novice programmers how to efficiently code and debug small programs. However, novice and experienced programmers alike need to become more proficient at problem analysis and high level program design. Here is where the nonacademic participants noted the greatest difference between good and poor programmers in industry. The better programmers are invariably better at problem analysis and program design tasks.

But how do expert programmers approach these tasks? Testimonials from the best programmers suggest that an extensive memory, intuition and, perhaps, the ability to visualize, play important roles in the architectural phase of programming [4]. Weber hypothesizes that expert designers may use non-sequential mental imagery (or pattern recognition) as a compact way to manipulate complex information in consciousness. It is possible, then, that the best programmers do something different from, and not something more than, other programmers, and that this expertise may not be possible to model precisely or to achieve routinely through academic training with relatively small and isolated programming projects. Nonacademic seminar participants noted that student programmers with "real world" experience tend to be much better at abstraction and program design than those who have had no such experience.

CONCLUSIONS

The different backgrounds and interests of seminar participants forced a concern with definition and led to the distinction of two educational issues: the training of professional programmers and programming literacy for novices. The empirical research surveyed in the seminar seems to offer more insight into the problem of how to train novice programmers.

Nonetheless, these studies collectively identify differences between how novice and experienced programmers solve programming problems and find program errors, and several of them find that expert programmer behavior is associated with the type of experience not the amount of experience a programmer has had. These results suggest that novice programmers can acquire some of the skills associated with experienced programmers if they are provided with good examples of procedural templates and encouraged to verbalize program goals and mechanisms. Yet, such techniques do not appear sufficient to train successful software engineers, where success applies to the entire software development process including high-level program design.

Informal reports of experiences by the nonacademic seminar participants suggest that new trainees from academia who have had previous nonacademic experience generally perform better than those who have not, and that some who have been academically successful turn out to be poor performers on the job. The implication is that educational approaches based on the cognitive differences between novice and experienced programmers are not likely to directly address differences between more and less successful programmers in industry.

A more effective approach might be to include activities that approximate

those required by large-scale industrial projects. The utility of these activities would be enhanced by (1) complete projects where students carry a project from requirements analysis through coding and testing, (2) projects complex enough to span several academic terms, and (3) industry cooperation where possible. Moreover, the empirical evidence examined above suggests that such projects should be included in early programming courses as well as in a capstone sequence that comes at the end of a student's academic career.

REFERENCES

1. Adelson, B., "Problem Solving and the Development of Abstract Categories in Programming Languages", MEMORY & COGNITION, 9, 4 (1981), 422-433.

2. Austin, H.S., "Predictors of Pascal Programming Achievement for Community College Students", ACM SIGCSE BULLETIN, 19, 1 (Feb. 1987), 161-164.

3. Brooks, R.E., "Towards a theory of the cognitive processes in computer programming", INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES, 9 (1977), 737-751.

4. Lammers, S., PROGRAMMERS AT WORK (1st Series): INTERVIEWS. Redmond, Washington: Microsoft Press, 1986.

5. Littman, D.C., J. Pinto, S. Letovsky & E. Soloway, "Mental Models and Software Maintenance", EMPIRICAL STUDIES OF PROGRAMMERS, (Soloway & Iyengar, Eds), Norwood, New Jersey: Ablex Publishing Corp., 1986.

6. Mayer, R.E., "The Psychology of How Novices Learn Computer Programming", ACM COMPUTING SURVEYS, 13, 1 (March 1981), 121-141.

7. McKeithen, K.B., J.S. Reitman, H.H. Rueter & S.C. Hirtle, "Knowledge Organization and Skill Differences in Computer Programmers", COGNITIVE PSYCHOLOGY, 13 (1981), 307-325.

8. Ratcliff, B., & J.I.A. Siddiqi, "An empirical investigation into problem decomposition strategies used in program design", INTERNATIONAL JOURNAL OF MAN- MACHINE STUDIES, 22 (1985), 77-90.

9. Sheil, B.A., "The Psychological Study of Programming", COMPUTING SURVEYS, 13, 1 (March 1981), 101-120.

10. Sheppard, S.B., B. Curtis, P. Milliman & T. Love, "Modern Coding Practices and Programmer Performance", COMPUTER, 12, 12 (Dec. 1979), 41-49.

11. Shneiderman, B., & R. Mayer, "Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results", INTERNATIONAL JOURNAL OF COMPUTER AND INFORMATION SCIENCES, 8, 3 (1979), 219-238.

12. Soloway, E., & K. Ehrlich, "Empirical Studies of Programming Knowledge", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SE-10, 5 (Sept. 1984), 595-609.

13. Soloway, E., "Learning to Program = Learning to Construct Mechanisms and Explanations", COMMUNICATIONS OF THE ACM, 29, 9 (Sept. 1986), 850-858.

14. Vessey, I., "Expertise in debugging computer programs: a process analysis", INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES, 23 (1985), 459-494.

15. Weber, E.S., "Cognitive Processes Involved in Solving Information Systems (IS) Design Problems", PROCEEDINGS OF THE SIXTH INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS, Indianapolis (Dec. 1985), 305-312.

16. Weiser, M., & J. Shertz, "Programming problem representation in novice and expert programmers", INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES, 19 (1983), 391-398.

CURRICULAR DIFFERENCES BETWEEN GRADUATE AND UNDERGRADUATE PROGRAMS IN INFORMATION SYSTEMS

Jennifer L. Wagner, Ph.D.
Roosevelt University

Issues and concerns in graduate MIS programs differ greatly from those in undergraduate programs. Developing the curriculum for a Master of Science in Information Systems program requires more than simply upgrading the undergraduate course content. Several of these underlying curricular concerns and examples from the curriculum of one particular MSIS program are described here.

While many graduate business programs permit a concentration in management information systems (MIS), a complete Master of Science in Information Systems (MSIS) program is much less common [8]. Since there are few such programs, curricular guidelines are far from standardized. Issues relevant to curriculum development for a graduate program differ greatly from those for an undergraduate major. The experiences, expectations, and needs of graduate students cannot be accurately generalized. Preparing a graduate version of a course requires more effort than simply upgrading undergraduate course content, more than adding a research paper, and more than just covering more material.

GRADUATE LEVEL CURRICULAR CONCERNS

Although MIS includes a great deal of technical detail and rests on a technological foundation, the MSIS program must maintain a conceptual view, while meeting the needs of diverse students.

Graduate students bring a variety of majors, industry experiences, and motivations. They have chosen their careers, industries, and even organizations. Many study on a part-time basis while continuing full-time employment. Such students demand that course material be immediately applicable to their present workday problems.

The goal of the MSIS program is to provide students with the knowledge which they need to become effective MIS managers. The MSIS program serves two groups of students: those with technical or computer-related undergraduate degrees, who wish to increase their knowledge of managerial topics; and those with business or other undergraduate degrees, who wish to broaden their knowledge of the somewhat-technical MIS topics. Both groups want career paths leading to the position of chief information officer (CIO).

Managers need skills in three areas: conceptual, technical; and human [6]. The major goal of all graduate education is a research-based orientation so that students "keep up with the field" even after the conclusion of their coursework. A deep understanding of the conceptual foundations is essential. But MIS tends to be technology-based, if not technology-driven. It is difficult to hold a conceptual perspective in the face of such extensive technology.

A major concern is the level of coverage of technical components. Which, if any, specific languages, tools, or techniques should be taught? To be an effective CIO, a manager must have enough knowledge of the technology to avoid being "snowed" by vendors. The CIO must also retain the respect of the MIS staff. Since these specialists have a technical orientation, preferring jobs with technical challenge, technically interesting work, and opportunities to learn new technological skills [2], the CIO needs

knowledge of the fundamental technological concepts. Also, the CIO's knowledge of the more popular tools and techniques will facilitate communication with the MIS staff. Some experience with programming languages is likewise helpful, both to increase the candidate's value in the job market and to enhance the understanding of programmers' problems.

CURRENT CURRICULAR MODELS

The Data Processing Management Association (DPMA), the Association for Computing Machinery (ACM), and the International Federation for Information Processing (IFIP) have prepared undergraduate standards, but they have largely ignored the graduate curriculum.

While DPMA has developed a widely accepted undergraduate curriculum [3], the graduate curriculum remains a "preliminary report" [5]. This curriculum assumes that MSIS students have completed all the business courses at the undergraduate level and that they have also completed these MIS courses: Introduction to Computer Information Systems; Introduction to Business Applications Programming; Systems Development Methodologies - A Survey; and Datafiles and Databases. The MSIS curriculum itself includes: Seminar in Information Analysis; Seminar in Decision Support Systems; Seminar in Distributed Processing and Telecommunications; Seminar in Database Management and Administration; Seminar in Audit, Control and Legal Aspects of MIS; Seminar in Office Systems; Seminar in Project Management; Seminar in Information Resource Management; Seminar in MIS Planning; and Capstone Research Project.

ACM sees graduate education as greater depth but not breadth, so their curriculum proposal [7] suggests only two additional courses for graduate students. These (Modeling and Decision Systems; Information Systems Policy) join: Computer Concepts and Software Systems; Program, Data, and File Structures; Database Management Systems; Data Communication Systems and Networks; Information Systems in Organizations; Information Analysis; Systems Design Process; and Information Systems Projects.

The IFIP curriculum guidelines [4] provide no distinction between the undergraduate and graduate levels, nor are individual courses identified. The curriculum is partitioned into three areas: Source Studies; Major Contributory Studies; and Mainstream Studies. The IFIP approach is more humanistic and sociological than either DPMA or ACM.

In addition to these guidelines, a program within a business school must also follow the guidelines of the regional accreditation agencies. Furthermore, voluntary associations, such as the American Assembly of Collegiate Schools of Business (AACSB), provide curricular guidelines. AACSB requires, for example, "organization theory, behavior, and interpersonal communications; ... administrative process under conditions of uncertainty including integrating analysis and policy determination at the overall management level" [1, p. 29].

IMPLICATIONS FOR THE GRADUATE CURRICULUM

The MSIS curriculum as a whole must synthesize the guidelines described above, while simultaneously meeting the needs of the graduate students with respect to their background characteristics and career objectives. An appropriate balance between technical detail and fundamental theory must be struck. Thus the graduate courses must include some basic technical detail while emphasizing the conceptual basis and overall framework of the technology. Undergraduate students learn how to use various MIS tools and techniques; graduate students must learn why as well as how. Managerial issues must be covered. Each course must ease the students' assimilation of theory into their prior experiences.

THE RESULTING CURRICULUM

The MSIS curriculum described here con-

sists of ten required courses. No prior computer-related courses are required. Programming is not required; it is, however, strongly recommended. In addition to seven MIS courses, students must complete three business courses: Production Management; Statistical Inference; and Managerial Accounting. Each is closely related to MIS and will help the MSIS holder be accepted by the organization's upper management team. The seven required MSIS courses are:

Introduction to Information Systems: basic hardware and software; methodologies for designing business systems; concepts of data and information; introduction to programs, including methods for describing data and program logic; office applications of computers, including word processing, databases, and spreadsheets. (This course is prerequisite to all other MSIS courses.)

Systems Analysis: structured analysis and logical design; techniques for stating and analyzing requirements; logical design and specification of system output, input, and processing; procedures for system cost benefit analysis; the life cycle concept; user relationships. Techniques and tools such as data flow diagrams, data dictionaries, structured English, decision trees, and decision tables. Group projects are emphasized.

Systems Design and Implementation: program and physical system design; alternative system structures; design of program structures, subsystems, and user interfaces; hardware and software evaluation and specification; testing procedures; implementation and conversion problems; project management techniques. Techniques and tools such as structure charts, modules, coupling, and cohesion. Group projects are used.

File and Database Systems: direct access file organization; logical database organization, analysis, and design; evaluation of file and database organization alternatives; relational databases; database management with emphasis on security and the database administrator; database management software.

Accounting and Financial Information Systems: applications in accounting and finance; information needs, decision requirements, processes, techniques, and data flows. Oral presentations are stressed. Managerial Accounting is prerequisite to this course.

Decision Support Systems: the role of MIS in assisting management decision-making at all levels in all functional areas; design and development of decison support systems; cognitive style; expert systems. This course emphasizes the interactions between managers and computers, as well as the interactions between managers and MIS specialists.

Integrated Information Systems: management policy; information management; integration of communication and information; business strategy; the management of change; managing technical workers and other human resource issues in MIS. This course is the capstone course in the MSIS program and, as such, includes team projects and oral presentations.

UNDERGRADUATE COURSES

Four typical undergraduate MIS courses are described to demonstrate the curricular differences between graduate and undergraduate programs.

Introduction to Business Computing (in the business school): the fundamentals of modern business data processing and computing; computer hardware, systems analysis and design, business software applications, the impact of computers on the business organization and society; introductory hands-on experience with microcomputer applications.

Systems and Procedures (in the business school): theory of systems development; procedures charting, form design, control and standardization; computer logic and organization.

Systems Analysis and Design I (in computer science): introduction to the methods and procedures for development of computer-based information systems; the life cycle concept; documentation requirements; project control.

Systems Analysis and Design II (in computer science): a case study in the development of an information system; tools and techniques used in systems analysis and design.

COMPARISON

In each case, comparing the graduate course to the corresponding undergraduate course shows the graduate to be more theoretical and managerial, while the undergraduate is more procedural.

Introductory Course (Introduction to Information Systems vs. Introduction to Business Computing): The graduate course emphasizes the concepts of data and information, as well as the concepts of decision-making, planning, and controlling. It also includes the methodologies for designing systems and describing data. The undergraduate course emphasizes individual elements (hardware and software) of the computer system. The hands-on applications account for 20% of the undergraduate course and 10% of the graduate course.

Systems Analysis and Design: At the graduate level, the two courses include tools and techniques, but they emphasize the managerial issues, such as communication, conversion, project management, and testing. The business undergaduate course is strictly "how-to", providing methods for accomplishing a set of analysis tasks. The computer science courses also primarily teach tools and techniques, although the students complete a project as well.

The MSIS program is based on a combination of the professional and business curricular guidelines, as well as the demands of a particular student body and educational setting. It incorporates the technical details of analysis, design, and implementation, while maintaining a conceptual and managerial perspective. The graduate course content is more theoretical than the content of corresponding undergraduate courses. In addition, the teaching approach at the graduate level places a greater emphasis on group projects and oral presentations. Most of the graduate courses require research or "thought" projects, unlike the undergraduate courses.

REFERENCES

[1] AACSB 1986-87 Accreditation Council Policies, Procedures, and Standards, AACSB, 1986.

[2] Bartol, K. M. and D. C. Martin, "Managing Information Systems Personnel: A Review of the Literature and Managerial Implications," MIS Quarterly, volume 6, 1982, pp. 49-70.

[3] The DPMA Model Curriculum for Undergraduate Computer Information Systems (First Edition), DPMA, October 1985.

[4] Draft Revision of International Curriculum for Information Systems Designers, IFIP, November 1984.

[5] A Graduate Model Curriculum in Computer Information Systems: A Preliminary Report, DPMA, 1986.

[6] Katz, D., "Skills of an Effective Administrator," Harvard Business Review, volume 52, 1974, pp. 90-102.

[7] Nunamaker, J. F., Jr., J. D. Couger, and G. B. Davis (editors), "Information Systems Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs," Communications of the ACM, 25:11, November 1982, pp. 781-805.

[8] Wagner, J. L., "A Survey of MIS Faculty and Programs," 1987 Midwest Business Administration Association Annual Meeting, Chicago, March 1987.

The Master of Science in MIS: A Model Curriculum Which Complies with AACSB Standards

Dr. Thomas A. Pollack

Duquesne University
School of Business and Administration
Pittsburgh, PA 15282

ABSTRACT

This paper presents the Duquesne University Graduate Management Information Systems (MIS) curriculum. The program, as presented, leads to the degree of Master of Science in MIS. The program design is in compliance with AACSB guidelines and standards. Input from faculty, the corporate community, and former graduate students was synthesized prior to the design of this program. The content of the courses in the program reflect topics which are considered important by the faculty as well as the corporate sector.

INTRODUCTION

Over the past one-half dozen years, the DPMA Model Curriculum has attracted a great deal of attention among educators involved with programs in information systems. The model curriculum has served as the focal point for many of the curriculum discussions which are interesting only if you happen to be wrestling with the problem being discussed.

The original model curriculum and the revision have been extremely valuable to undergraduate information systems education programs. The true value, however, is difficult to define. The obvious value of the document is realized by those institutions which adopted the model curriculum in whole. Then, of course, others realized value by adopting in part. Perhaps the model curriculum has been most valuable to the masses because it provokes discussion. It focuses discussion for Management Information Systems (MIS) educators. It provides a focal point, whether one agrees or disagrees with its content. The interaction which results is healthy in that new thoughts and ideas often result. Nearly all MIS educators have benefitted either directly or indirectly, and their programs are probably better programs as a result of the model curriculum.

During the past several years, there has been considerable discussion concerning graduate study in MIS. What are various institutions doing with curriculum? Should there be a model curriculum for graduate study in MIS? How are the accrediting standards established by the American Assembly of Collegiate Schools of Business (AACSB) affecting the curriculum? These are very interesting questions, and the MIS faculty of Duquesne University discussed each as we collaborated to assemble a proposal for a Master of Science Degree in Management Information Systems (MS-MIS).

PROCEDURE

As is the case at most universities, curriculum development and program change is a rather lengthy process at Duquesne University. A lead time of approximately one year is necessary for the implementation of major changes. Thus, the bulk of the curriculum design work described in this paper was completed during the Summer of 1987 for implementation in the Fall of 1988.

During the 1970's, Duquesne University's School of Business and Administration was a pioneer of sorts in that it offered a program which awarded the degree of Master of Science in Business Information Systems. Unfortunately, the program was discontinued in the late 1970's due to the School's inability to provide funding in the form of qualified faculty and computer resources. The current MIS faculty, which numbers six, was charged with the task of reinstitution of a master's level program in management information systems. The author served as overall co-ordinator of this project.

A logical starting point seemed to be a justification of need for this program. Duquesne's MBA program is rather healthy with a stable enrollment of about 400 students, mostly part-time. However, a review of demographics and a perusal of the

literature indicates that, in the future, we may have to depend more and more on the non-traditional student to maintain steady-state enrollment. Universities must address the needs of the non-traditional student. Many feel that there will be an increasing emphasis on understanding the college choice process of graduate students as the declining applicant pool of traditional college students passes through its life cycle (4, p. 305). Additionally, the traditional MBA degree has come under extensive criticism in recent years (6, p. 72). It appears that some graduate students have become somewhat disenchanted with the rather broad and general exposure offered by the typical MBA program and have expressed a desire for the more specialized master of science (MS) degree. Another factor considered is competition from other universities. Only one other local AACSB-accredited university is currently offering a graduate program in MIS, and it is billed as a concentration. Finally, Pittsburgh houses a significant number of corporate headquarters, and the demand for well-trained MIS professionals is substantial. Thus, it seemed as though sufficient justification for the graduate MS-MIS program exists.

The next consideration is the accreditation guidelines provided by AACSB. Any program within an accredited school which offers the MS-MIS degree must be tailored to suit the guidelines provided by AACSB. This is particularly true in the defined common body of knowledge courses (1, p. 30). Being that Duquesne's School of Business and Administration has been accredited for years, faculty standards and admission standards are currently in place for the existing MBA program (1, pp. 22-23). Thus, careful attention was given to AACSB guidelines for curriculum in arriving at the mix of coursework proposed for our graduate MS-MIS model.

The curriculum design process itself was also an area of concern. That sometimes controversial document, the DPMA Model Curriculum provided a starting point for discussion (2). Although we have no interest in duplicating our undergraduate program in MIS, conceptually, the DPMA Model Curriculum is applicable. In our discussions, we agreed that everything has to be offered at a higher theoretical level than the undergraduate courses with a substantially increased research and managerial component. We thought it to be especially important to arrive at a balance between conceptual, theoretical, and practical content of courses. Another factor which warranted a great deal of discussion on our part concerned the diverse backgrounds of the students entering the program. A wide range of incoming competencies must be accommodated through prerequisite requirements. We are not at the stage in our development where we feel totally comfortable with this very significant variable.

In the actual curriculum design, we employed a model quite similar to that advocated by Lane, Harpell, and Mansour. Their model integrates three major interest groups, namely the university, business and industry, and students (3, p. 367).

The MIS faculty of Duquesne University are an extremely competent group who blend technical competency and managerial expertise. They are both research and teaching-oriented. They are current in the MIS field and they have an excellent sense of articulation for courses within the curriculum. As an instructional designer, the author deems the above qualities as being critically important. The faculty group has a strong sense of what belongs in the curriculum and also what outcomes will result. Perhaps, most significantly, this group possesses an uncanny ability to discuss, argue, deliberate and eventually arrive at some compromise of a consensus.

The author surveyed the corporate sector last year and sought the corporate sector's ranking of MIS competencies (5, pp. 2-3). These rankings were synthesized and integrated into the coursework of the proposed model.

Finally, the faculty was able to gather input and ideas from three graduates of Duquesne's former graduate Business Information Systems program. Strengths and weaknesses of the previous program were discussed and considered as the current model was designed.

The funds to design this graduate MS-MIS curriculum were made possible by a program development grant to the faculty. The money was well-spent as many collective hours of brain-storming and individual hours of course proposal writing resulted. The author encourages others aspiring to write or rewrite curriculum to pursue potential grants for this purpose if finances are a problem. An excellent case and rationale can be stated.

THE PROGRAM OF STUDIES

The primary objective of the MS-MIS program is to develop those competencies which prepare the graduate for productive careers in such positions programmer/analyst, systems analyst, systems designer, project manager, database administrator, data communications specialist, expert systems developer, information center consultant, and information systems manager. Students are trained in the management of modern organizations (AACSB Common Body of Knowledge Core Curriculum for MBA candidates) as well as the technical and conceptual aspects of computer-based information processing systems. The student is also taught to manage information flow and to treat information as a valuable organizational resource. Some of the specific technical competencies which the student will possess upon completion of the program include the abilities:

1. To analyze strategically, tactically and economically the information needs of an organization;
2. To design computer-based information systems;
3. To effectively use structured tools and techniques;
4. To develop a solid conceptual understanding of systems software and its interaction with hardware;
5. To assess and select appropriate computer hardware and software;
6. To perform conceptual, logical, and physical database design and implementation;
7. To understand the role of expert systems and decision support systems and be capable of applying them to a given situation;
8. To analytically apply modeling software techniques;
9. To manage information system resources; and
10. To implement system projects.

A breakdown of the courses which comprise the MS-MIS program of studies follows:

CORE I-A - Conventional MBA Prerequisites
(Prerequisite for Core I-B)

		6 Credits
501	Mathematical Foundations for Management	2
502	Computer Fundamentals for Quantitative Management Techniques	2
503	Statistics as a Basis for Managerial Decision Making	2

CORE I-B - Foundations, Prerequisite: Core I-A

		14 Credits
511	Quantitative Analysis for Management Decisions	2
512	Foundations of Accounting and Control	2
513	Economics for Managers	2
514	Financial Control of Organizations	2
515	Management Information Systems	2
518	Law for the Executive	2
519	Managerial Marketing Strategies	2

CORE II - Required MBA Courses Total of 12 Credits

521	Environment of Business	3
523	Operations and Production Management	3
524	Organizational Behavior in Management	3
528	Strategic Management	3

CORE III - Area of Concentration Courses
Total of 24 credits

REQUIRED COURSES — 15 Credits

541	Computer Concepts and System Software	3
544	Systems Analysis	3
545	Database Management Systems	3
642	Decision Support/Expert Systems	3
648	Information System Design Project	3

ELECTIVE COURSES — 9 Credits

542	Advanced Software Support Systems	3
641	EDP Auditing and System Controls	3
643	Modeling and Simulation	3
644	Decision-Making Heuristics and Expert Systems	3
645	Management of the Information Resource	3
647	Distributed Information Systems	3
649	Research Problems	3

Students entering the MS-MIS program will therefore be required to complete a minimum of 36 and a maximum of 56 credits depending on their undergraduate training and/or their ability to demonstrate competency by examination in a given area. In adtion, proficiency in COBOL programming is a required entry-level competency. Each student's transcript will be evaluated when the student is accepted into the program. The total number of credits required to complete the degree will be established, and the student will be notified of the availability of competency examinations.

MBA with MIS Concentration

In conjunction with the MS-MIS program offerings, the School of Business and Administration will also offer added flexibility in its offerings to MBA students by offering a "concentration" in MIS. By devoting nine credits within the current MBA structure to MIS electives, students can earn the MBA with a concentration in MIS.

Students pursuing an MBA with a concentration in MIS are encouraged to take:

541 Computer Concepts and System Software,
544 Systems Analysis

and an additional elective from among 542, 641, 642, 643, 644, 645, and 647.

Corporate Advisory Board

In an effort to provide a meaningful program of instruction in MIS at both the undergraduate and graduate levels, the School of Business and Administra-ion has formed an MIS Corporate Advisory Board. The purpose of the advisory board is to make realistic recommendations regarding overall curriculum, hardware and software support. The advisory board will meet regularly and will provide both the insight and the impetus to keep the program current. It is our opinion that this close tie and interaction with the corporate community will be a significant factor in establishing a reputation of excellence for our MS-MIS program.

CONCLUSION

This paper has presented the collective thoughts of the Duquesne University MIS faculty in the form of a model graduate curriculum leading to the degree of Master of Science in Management Information Systems. The curriculum design complies with AACSB standards and guidelines. It is the author's hope that this model curriculum may serve as a document which initiates discussion and subsequent refinement suggestions much as has been the case with the DPMA Model Curriculum. The ultimate result will then be better graduate MIS programs from which we all can benefit.

REFERENCES

1. AACSB Accreditation Council "Policies, Procedures, and Standards," American Assembly of Collegiate Schools of Business.
2. CIS `86 "The DPMA Model Curriculum for Undergraduate Computer Information Systems," Second Edition, Data Processing Management Association, July 1986.
3. Lane, Michael S., Harpell, John, and Mansour, Ali, "An Integrated Approach to Curriculum Design/Redesign," Journal of Education for Business, May 1986, pp. 367-370.
4. Olson, Carol and King, Milton A., "A Preliminary Analysis of the Decision Process of Graduate Students in College Choice," College and University, Summer 1985, pp. 304-315.
5. Pollack, Thomas A. and Shepherd, John C., "Business/Education Collaboration: Competencies Ranked by CIS Professionals," ISECON `87 Proceedings, pp. 63-66.
6. Windsor, Duane and Tuggle, Francis, "Redesigning the MBA Curriculum," Interfaces, August 1982, pp. 72-77.

INSTRUCTIONAL GROUPING THROUGH ATTENTIVE LOCALE ASSESSMENT

Floyd D. Ploeger, Ph.D.
Department of CIS/ADS
School of Business
Southwest Texas State University
San Marcos, TX 78666-4616

The results of this three year study suggest that an instrument has been developed which identifies the "Attentive Locale" of participants in a program. As a function of prior knowledge, participants exhibit differential tendencies to be receptive to instruction and to assimilate information. It is believed participants progress through the five hierarchical levels of "Gather", "Internalize", "Assess", "Share", and "Expand". The "Attentive Locale" instrument has been developed to identify the hierarchical level of a participant for any program or content. The information from the instrument developed by this research permits informed decisions to be made regarding instructional programs. Measuring the "Attentive Locale" facilitates the decision making process by providing a uniform measure of the level of reception participants may exhibit toward instruction. With this information, educators may make inferences which allow them to more effectively group participants, design projects, or alter existing programs.

INTRODUCTION

The performance of an individual and, to a great extent, the effectiveness of a program is affected by participant attention. The degree to which participants are ready to take part in a program can be valuable information for educators. By assessing participants' potential attentiveness, educators can begin to tailor both the method of presentation and the type of information provided.

The results of this three year study suggest that an instrument has been developed which identifies the "Attentive Locale" of participants in a program. The information gathered from the instrument developed by this research permit informed decisions to be made regarding potential participant levels of attention. Identification of the "Attentive Locale" facilitates the decision making process by providing a uniform measure of participant's readiness to be receptive to a program's content. This may help educators make inferences which allow them to affect the focus and pace of a program.

RESEARCH BASIS

It has been suggested that the persistence of the process of change has caused significant concern among educators. A model was developed based upon concerns which educators express during the time in which a program is undergoing change. The concept of "Stages of Concern" is a significant underpinning of the "Concerns Based Adoption Model". The model suggests that individuals developmentally progress from a stage of concern about becoming more aware of

the "innovation" at hand through expressing a desire for "refocusing" on other related "innovations". After determining an individual's "Stages of Concern", indicated "interventions" may be suggested in an attempt to remedy or alleviate the level of concern. The model is founded on the concept that concerns about a change or innovation will inhibit the process of change. By intervening appropriately, educator's concerns are removed and the intended innovation can take place.

Another model suggests that the decision making process is based upon a progression through stages referred to as the intelligence stage, design stage and choice of solution. Essentially the model is based upon a progression of participant information stages from awareness through evaluation of identified alternatives. It is believed that the participant must effectively complete each successive stage in order to arrive at an accurate and valuable decision.

As a result of these studies, the "Attentive Locale" study was begun. The "Attentive Locale" concept is based upon some of the combined principles of these two stage oriented models. The "Attentive Locale" concept is learner and program directed as opposed to the educator or decision oriented approach of prior research. The "Attentive Locale" model measures the perceptions of the participant learner in an instructional setting. Data gathered through use of the "Attentive Locale" questionnaire provides a measure of the degree to which the participant's perceptions are coincidental with the goals of an instructional program. Instead of attempting to identify concerns the participant has toward an innovation or decision, the "Attentive Locale" concept measures the participant's perception of similarities between themselves and the goals of a program. As a result, a positive approach is taken in which the methodology of instruction may be altered to more nearly meet a participant's needs. The "Attentive Locale" suggests that the learning process may be represented by a continuum along which a participant may be located. The continuum is identified by five hierarchical processes which are referred to as Descriptive Categories.

Descriptive Categories

1. Gather: information flow from the environment to the learner.
2. Internal: integration of external information with gathered information.
3. Assess: prioritization of internalized information.
4. Share: outflow of information which has been assessed.
5. Expand: extension of shared information into other realms.

PURPOSE

The purpose of this study is to report the development of an instrument designed to determine the degree to which participants perceive themselves to be consistent with the thrust of a program. It is believed that the "Attentive Locale" or alignment with an instructional content differs among individuals over time. The measure of "Attentive Locale" is a reflection of the individual's perception of their similarity to the program's goals. The goal of participants who desire to "Gather" information about the content of a program will differ from those wanting to "Share" their knowledge with others. Likewise, participants desiring to move on to other areas and "Expand" upon the goals of the program will differ in "Attentive Locale" from those who want to "Assess" the program's goals. By knowing the "Attentive Locale" or focus of attention of participants in a program, instruction may be targeted to meet their needs. Or, the participants having a similar "Attentive Locale" may be grouped together in order to facilitate the instructional process. It is valuable to recognize that the "Attentive Locale" is based upon an individual's perception of the alignment between themselves and

their perception of a program's content or goal at a given time. In addition to identifying the "Attentive Locale" of participants in order to more effectively disseminate information, the value of a program may be assessed. The "Attentive Locale" of a group or sub-group may be determined prior to beginning instruction and immediately following completion of a program. A "pretest-posttest" change in "Attentive Locale" may be interpreted as an effect which the program has on the participants. If a program's goal is to foster awareness, a measured shift from "Gather to Internal" may be interpreted as indicative of the program's effectiveness. However, a measured "Attentive Locale" change from "Gather to Expand" may suggest that saturation and perhaps boredom has overcome the participants' desire to remain attentive. A large shift in "Attentive Locale" may further suggest that the participants' content knowledge is high but that they were bewildered by the introductory jargon when the "Attentive Locale" was measured.

QUESTIONNAIRE DESIGN

A Likert type scale is used in which zero (0) represents the non-applicable category and one (1) through six (6) indicate increasingly greater degrees of agreement between a statement and the participants understanding of the thrust of a program. The instrument is composed of 30 statements, each of which is logically associated with one of the five (5) listed Descriptive Categories. The thirty (30) statements have been randomized on the basis of logical association with these terms. As a result, it is believed that a logical association between the thirty (30) statements and the Descriptive Categories is not apparent to the participant. The questionnaire may be scored by adding the points assigned by the participant to the questions in each of the five (5) Descriptive Categories.

RESEARCH DESIGN

The "Attentive Locale" questionnaire was administered to full-time students attending class in the School of Business at a central Texas state supported university. A total of 558 subjects were used to develop the thirty (30) question set and the five (5) Descriptive Categories. Class content involved Computer Information Systems, Management Information System, and Management Policy courses. A pretest-posttest design was used during the three (3) year development process. The questionnaire was administered to subjects attending class on the first day of class, after the syllabus was provided by the professor concerning the topical outline of the class content. After completion of the course of study but before final examination, the posttest administration was completed. The subjects were instructed to rank each of the questions according to "...how well the statement is descriptive of you (the subject)..." as they perceive the content of the course of study. The subjects were informed that the questionnaire was part of a research project and had no effect upon their grade. Subjects were identified by using only the last six (6) digits of their social security number in order to pair pretest and posttest answers.

RESULTS

The Statistical Package for the Social Sciences (SPSS^{X}) employing a Pearson Correlation (PEARSON CORR) and a T-test (T-TEST) in a two-tailed test for statistical significance are believed to be appropriate measures. The ranking number on the Likert type scale (0-6) which the subject assigned to each question was used as input data for PEARSON CORR statistics. The mean rankings for all statements within each Descriptive Category was used as input for T-TEST statistics. Correlation coefficients were determined for ques-

tions in each Descriptive Category. Within Descriptive Categories, item correlations of $.6500 > r > .9800$ at a confidence level of $p<.05$ have been observed for all questions. However, most questionnaire items correlate with other items in the same Descriptive Category at values greater than $r = .7500$ and $p<.001$. Further, T-TEST results indicate significant differences between the means of Descriptive Category rankings with confidence levels of $p<.01$.

DISCUSSION

The statistical analysis of the data indicate the uniqueness of each of the Descriptive Categories because the statements included in each category correlate well with one another. Thus, they independently indicate the degree to which participant perceptions coincide with the goals of a program. It is believed that the results of this study suggest that the "Attentive Locale" questionnaire is a valuable instrument which may be used to facilitate the instructional decision making process. The "Attentive Locale" questionnaire may be used in at least three broad areas by professional educators. The concept of "Attentive Locale" is believed to be valuable for measuring the readiness of participants to be receptive to instructional programs, the implementation of a program, and the alteration of "inplace" programs.

REFERENCES

Fuller, F., "Concerns of Teachers: A Developmental Conceptualization", American Educational Research Journal, vol. 6, pp. 207-226, 1969.

Hall, Gene E., "The Concerns-Based Approach to Facilitating Change", Educational Horizons, vol. 57, pp. 202-209, 1979.

Hall, Gene. E., "The Study of Individual Teacher and Professor Concerns About Innovations", Journal of Teacher Education, vol. 27, pp. 22-23, 1976.

Simon, H.A., The New Science of Management Decision, Harper & Row Pub., 1960.