1-1-1983

Teaching Software System Design: An Experiential Approach

Thomas E. Perkins
Southern Methodist University

TEACHING SOFTWARE SYSTEM DESIGN: AN EXPERIENTIAL APPROACH

Working Paper 83-100*

by

Thomas E. Perkins

Thomas E. Perkins
Assistant Professor
Management Science and Computers
Edwin L. Cox School of Business
Southern Methodist University
Dallas, TX 75275

This work was supported by the National Science Foundation under grant SER-7900607.

*This paper represents a draft of work in progress by the author and is being sent to you for information and review. Responsibility for the contents rests solely with the author. Please address all correspondence to Thomas E. Perkins.

TEACHING SOFTWARE SYSTEM DESIGN:

AN EXPERIENTIAL APPROACH

Thomas E. Perkins

Southern Methodist University

1. INTRODUCTION

Software developers are often faced with complex and difficult challenges. Issues they address involve starting with an ill-defined problem, abstracting the underlying processes and data relationships, representing the problem stucture in a manner that can be readily transformed into a programming language, designing and coding the system, testing the code produced, educating user personnel, planning for system conversion, and on and on. No single academic discipline adequately prepares the student for the vagaries of such an experience: software development may require a combination of skills derived from psychology, management science, operations research, organizational behavior, electrical engineering, mathematics, and computer science. Software development involves much more than programming. In fact, programming represents only a small part of the typical development effort [Boehm,1976].

Approaches to teaching software development in university environments tend to emphasize the programming aspects of the software project: ranging from providing realism in the classroom [Freeman, 1978] to structuring the development process [Kant, 1981]. The course described in this paper, however, views the software development project as a joint, participative effort involving both the users of the system as well as the developers. Students learn to apply software design techniques by dealing with real people with real software needs in real organizational settings. The focus of this paper is on the design portion of the project; a subsequent paper will address our experiences in actually implementing these designs.

2. COURSE DESCRIPTION

Software engineering is taught at SMU in a two-semester sequence. The fall semester course, Software Engineering Principles, is open only to students with a senior standing. The course lecture material provides an overview of the development project, discusses project management, and then focuses on the "creative" phases of the software life cycle - requirements engineering and systems design. In conjunction with the lecture material, students work in teams on separate "real-life" projects for the school administration and for local businesses and industry. The teams' goals are the preliminary design of the system's software architecture: each group is also required to prepare a system implementation plan for the next semester.

The spring semester course is called Software Engineering Laboratory. As the name implies, less emphasis is placed on lectures. This semester deals primarily with the experiences of the project teams as they wrestle with the "implementation" portion of the project. However, supplementary lecture material is presented to cover topics such as detailed design representation, coding techniques, software testing, customer education, and conversion considerations. The projects which are implemented during the spring are chosen from those designed during the fall semester. The spring semester course is open to both all levels of undergraduates. Usually seniors serve as project managers and team leaders, while other undergraduates take on team responsibilites commensurate with their experience and initiative.

3. LECTURE TOPICS

The course sequence was initially conceived of as a capstone "projects" course, with a few project management topics added. However, it soon became clear that, even though the students had an excellent background in programming techniques and an understanding of data structures, they were lacking some fundamental skills needed to translate a poorly defined problem into a professional piece of software. As a result, an appreciable portion of the lecture material is spent on "front-end" problems of the development effort: communicating with users with limited knowledge of computers, defining precisely what the problem is or what the user wants, representing the problem so that it can be communicated to both the user and other members of the project team, separating political problems from technical problems, and designing for a changing environment. An outline of the course modules is included in Appendix A.

Project Preliminaries

The lecture topics begin with a brief "history" of software development, tracking it from machine language programming of the early 50's to the multi-billion dollar a year industry it is today. The software crisis [Jensen and Tonies, 1979;Mills, 1977] is examined, as well as some of the pressures brought to bear on the developers of software systems in industry. The software life cycle is presented as one view of the evolution of a software system. Although this concept has come under criticism recently [Gladden, 1982;Jackson and McCracken, 1982], it still serves as an excellent framework within which to address the interdisciplinary nature of software development work. It can also be useful to assure harried students that software projects do eventually "end".

The course then describes the activities within each life-cycle phase in more detail, often using a scenario approach in which class members with previous experience or part-time programming jobs describe what happens in a development effort. The purpose of this module is two-fold: it gives the

student a relevant overview of processes really involved in a development effort, and it sets the stage for the discussion of project planning, scheduling, and monitoring which follows.

The next module introduces the Work Breakdown Structure (WBS) [Boehm,1981] as a means for organizing and estimating a development effort. The approach taken here is that although each software project is different, there are repeating patterns of activity common to all projects. The students are given a list of "standard" activities (Appendix B); later, they will be asked to tailor these activitiy descriptions and add others as they develop a WBS for their own projects. This module also introduces students to PERT and CPM as scheduling tools. Gantt charts are used to represent the schedule, as well as a tool for smoothing resource utilization.

Since most of the student projects will start with no written description, students are given a series of lecture on interview techniques. One lecture deals with the advantages of interviewing in a "top-down" order in an organization: managers first, then supervisors, then clerical personnel. Other lectures emphasize techniques of the individual interview. An exercise which has proven fruitful is to assign one of the more outspoken teams the goal of obtaining a piece of information about a system, and have the instructor to play the role of a recalcitrant and somewhat grumpy user. After the interview, other class members are asked to analyze the interview and to suggest question sequences which might have been used. Students are also asked to represent graphically the information which was presented verbally during the interview.The class then discusses the different interpretations of what was said, which invariably occur.

Defining the Problem

An initial step in the evolution of a software system is for the development team and the customer/user to reach closure on what the problem really is. This phase is often called "requirements engineering" [Boehm, 1976]. However, an equally important aspect of the project occurs during this phase: a relationship between the system developers and the users is established which will carry forward through the remainder of the project. The dual nature of the requirements

engineering phase is addressed in the course lecture material by presenting a development methodology which: 1) encourages a high degree of user participation and involvement in the definition and design efforts, 2) represents the evolving system in graphical form which the user can understand, 3) invites user changes and creativity early in the design process rather than after the design is complete, and 4) attempts to establish a sense of user "ownership" of the system. (The underlying question of "Whose system is it?" appears to impact not only the relationship between the designers and the users but also the actual structure of the resulting system. If the designers "own" the system, users may wind up with an accounts receivable system which looks suspiciously like a compiler. In this course setting, many students have the idea that the new system "belongs to" the designers and programmers only until the coding is complete, and are somewhat surprised when the users are reluctant to accept the coded system and finish the testing themselves.)

Problem definition is stressed as one of the most important portions of the project. Students are introduced to structured analysis [DeMarco, 1978;Gane and Sarsen, 1979] , a technique for examining information flow throughout the problem. The lectures also stress the importance of identifying and representing the structure of information at various points in the problem. Techniques for representing information structure include Jackson Structure Charts [Jackson, 1975], Warnier-Orr Diagrams [Warnier, 1974;Orr,1977], and Yourdon-Constantine Diagrams [Yourdon and Constantine, 1978]. In a series of class exercises, students are given a verbal description of an information system and copies of the reports it is to produce. Then they are asked to depict the problem structure graphically, showing both information flows (data flow diagrams) and information structures (data structure charts).

The data dictionary is presented as a necessary and integral tool for any development project. Students are introduced to structured walkthroughs [Freedman and Weinberg,1982] and design reviews [Inmon and Freidman, 1982] as a vehicle for presenting the designers' perception of the problem definition to the user and as a means for encouraging user participation and dialogue. The lectures emphasize the necessity for user feedback and input, as well as the need for representing the

problem as it is percieved by the user, not as it is perceived by the designer.

System Design

The students are introduced to two methodologies used to map the problem representation into a software architecture: data flow design approaches [Yourdon and Constantine, 1978;Pressman,1982] and data structure design approaches [Jackson,1975;Warnier,1974;Peters,1982]. A number of small problems are used to demonstrate each technique, and to point out the types of problems and development environments in which each technique has been sucessfully applied.

The issue of program modularity is discussed extensively. Students with part-time jobs (who usually work as maintenance programmers) are asked to describe to the rest of the class the types of problems they have encountered when attempting to modify a portion of a complex software system. Such discussions usually include lamentations about the lack of adequate documentation, the fact that many such modifications are made in stressful situations (often in the wee hours of the morning), the problems of having to "re-think" another individual's code, their lack of appreciation for "clever" programming tricks, how sensitive highly-coupled monolithic coding is to the simplest change, and on and on. We next focus on issues of coupling and cohesion [Stevens, Myers, and Constantine, 1974] using small examples of "pathological" coding for discussion. The focus on modularity issues concludes with an introduction to Halstead's Software Science measures [Halstead, 1977] and McCabe's complexity metric [McCabe, 1976]. The students are asked to analyze a piece of code and calculate values for each of these software metrics.

Throughout the discussions on software design, emphasis is continually placed on the desirability representing the design graphically, in a simple form which can be understood by both users and other members of the design team. Too many software projects have been disrupted by disagreements about "what was said when" - many of these misunderstandings could have been avoided by drawing a simple picture of what was meant. Since communication between team members and the user group will probably be one of the major problems experienced by the students in their project, the students

are encouraged to use understandable graphical representations as a focal point for discussions about the system.

4. THE USERS

Prior to the start of each semester, the instructor identifies a number of potential projects by contacting local businesses and administrative offices in the university. Key individuals are identified who are familiar with the details of the problem. The instructor meets with these individuals to assess the problem and the environment in which the students will be working. Each user group is cautioned that the project should be a small, "back-burner" type of problem which falls into the nice-to-have, rather than the need-to-have category, since there is always a possibility that the student group may not be able to handle the project. Users are encouraged to look at the project as two sub-projects, one for design and another for implemenation. (Sometimes in-house programmers have been able to complete an implementation when a student team couldn't be assembled for the second semester.) Each user group is appraised that the development methodology will require a significant commitment of time and involvement on the part of the key individuals. Even with all these constraints placed on the projects, practically every potential user group contacted was enthusiastic about supporting the course.

The benefits the user organization derives from the project are 1) a possibility of a completed piece of inexpensive software, 2) an opportunity to evaluate students for possible employment later, 3) exposure of in-house personnel to state-of-the-art development techniques, and 4) a sense of having contributed to software engineering education.

5. THE STUDENT PROJECTS

The student projects are conducted along the lines of a real life software development project, as though the students were programmers and analysts in a MIS department assigned to work on a user

problem. In general, the students try to follow a slightly modified version of the software development life cycle. In order to make the project a public, rather than private, process [Weinberg, 1971], the instructor places heavy emphasis on on-going communication with users through simple graphical representations of the project at each step along the way.

Problem Assessment

The first phase of the student project is a "problem assessment phase". Here, the students and the users attempt to define just what the problem is, come up with a preliminary analysis of the structure of the problem, and agree to a graphical representation the problem. The definition of the problem necessitates a period of intense interaction with the users, usually in the form of individual interviews. The interviews are conducted at both managerial and supervisory levels, as the team attempts to determine who are the key personnel and if the problem being addressed has political overtones. All notes taken during the interview become part of the project documentation. The results of the interview are discussed in class; other class members are asked to critique the interview and to offer suggestions of interviewing strategies the team might follow.

During this phase, the team begins develop a graphical representation of its understanding of the problem in the form of a Leighton diagram [Scott,1978], a HIPO Visual Table of Contents [HIPO;Stoy,1976], or a Structured Analysis Level I Data Flow diagram [DeMarco,1978]. The team can choose the type of representation it wants - the criteria is that it must be simple, non-technical, and not intimidating to the user. The graphical problem representation is presented to the user in a design review. In this meeting, the users are asked to point out any parts of the problem which they perceive to have a different structure than that developed by the designers. Although the users are given no formal training in any of the above techniques, by the end of the design review they often converse freely about the partitioning of the problem and parts of it which the students left out. We believe that this early dialogue is essential in constructing a system which will reflect both the user's needs and his perception of the problem structure, and which will also be adaptable as these needs

change in the future.

Rapid Prototyping

While dealing with the abstract nature of the problem assessment phase, the project teams and the users also focus on a tangible portion of the system about which both groups can converse - the system output. The students are encouraged to prepare layouts of all system reports and screens mentioned by the users. These layouts and screens are presented to the user in the form of a "mock-up", using values which would appear on the report if it had been generated by the operational system. Hard-copy reports are printed on a line printer; the students use a simple display program to present samples of interactive screens on a CRT terminal.

This simple form of "rapid prototyping" gives the teams and the users something tangible to discuss, yet the reports aren't "cast in concrete". In fact, the students are instructed to encourage the users to change the reports by using stock questions such as "Is this the report you requested?" "Would you like this report better if it were rearranged?" "Would other information be useful to you on this report?" Encouraging change at this point in the system serves two purposes. First, the prototypes are very easy to change - much easier to change than coded modules. Secondly, the process of making changes (and thereby participating in the development effort) seems to instill a sense of "ownership" of the system in the user group. To overcome the students' natural defensiveness when the users don't find their reports just letter perfect, the instructor and several students with work experience demonstrate in the classroom several scenarios of user changes at different points in the system development cycle.

Project Planning

Once the student teams and the users have developed a basic understanding of the problem to be solved, the teams undergo a period of high-level project planning. Each team develops a Work Breakdown Structure (WBS) hierarchy of the activities foreseen for its project. This list of high-level milestones is developed by tailoring a standard software development WBS hierarchy (Appendix B). Students are asked to estimate the duration of each activity, as well as the resourses required: student team involvement, user involvement, number of compilations, meeting areas, etc. The milestones are somewhat broad at this point, usually covering a duration of one week. During the project, more detailed activities are developed for a two-week planning horizon. The purpose of the initial planning exercise is to start the students thinking of the project as a whole, and not just concentrate on those activites to be performed next.

The teams are then asked to schedule their project activities and represent these schedules graphically, using PERT, CPM, or Gantt charts. The schedules are then formally presented to the users. Although most experienced users take the optimistic schedules with a grain of salt., the schedule charts (displayed in the computer science department student lounge) tend to give the projects a high visibility.

Requirements Analysis

During the planning phase, the student teams work primarily by themselves, except for the presentation of the schedules to the users. The next phase, however, again sees an extensive amount of communication between the students and the user groups. During this phase, a more detailed analysis of the problem is undertaken, using the highly graphical methodology called Structured Systems Analysis [De Marco, 1968]. The students represent the flow of data through the problem in data flow diagrams, refine the reports and screens the system is to produce, name all the data elements in the system, define required file structures, and document organizational policy through

Structured English Process Descriptions or decision tables. As each part of the system evolves from an intellectual concept into a more tangible form, the component is documented in a graphical form so that it can be discussed with the user group. This discussions take place in both informal work sessions and during more formal review presentations. As the documents which comprise the system requirements are completed, they are filed in the Project Notebook and kept on public display.

Software Architecture

After the system requirements have been documented and discussed with the users, each team then constructs a preliminary system module architecture. This process involves mapping the data flow diagrams into a module hierarchy [Pressman,1982;Yourdon and Constantine, 1978] or developing the system structure concomitant with the information structures suggested by the system's data [Jackson, 1975]. The teams are free to choose the methodology they prefer - usually the teams go through several design iterations, often mixing methodologies to understand some part of a system better. During the design sessions, the teams are encouraged to represent the system component under consideration graphically. Once the preliminary design is complete and module functions have been identified, the preliminary design is presented to the users in an informal design review.

Final Deliverables

The final deliverables for the design portion of the project consist of the project notebook (Appendix C), a large module hierarchy chart,, report and screen mockups, and a CPM chart showing the team's estimates and plans for next semester's effort. In lieu of a final exam, the students make a final, formal presentation to the user group, preferably at the user's site. Usually, the students put considerable effort into these presentations: they are well rehearsed, colored charts and overhead foils abound, and the students often sport new suits and haircuts. The purpose of this

final presentation is not so much evaluatative as it is for both the students and the users to reach a sense of closure on the design portion of the project. In fact, the students are instructed to try to create an experience similar to an apparent underlying philosophy in several major motel chains - "No surprises!".

6. EXPERIENCES

The software engineering course sequence has been taught for four years at SMU, covering a wide variety of projects, student teams, and outside user groups. The fall class size is usually between twenty-five and thirty students. Currently, the class is involved in nine projects, ranging from a microcomputer-based weather monitoring station to a decision support system for an administrative office on the campus. Although each of these projects has presented a unique set of experiences (and problems), possibly some general observations on the course can be drawn.

Lectures versus Projects

During the first semester the design portion of the course was taught, the instructor tried to interleave the lecture material with the project effort - always trying to stay just ahead of what the teams would be doing next. This proved to be a frustrating experience, since each project moved at a different pace. Our current approach is to not assign the projects until we are at least one-third into the semester, using the first five or six weeks to concentrate on lectures. During the remainder of the semester, lectures are interspersed with project discussions and reviews. This is still somewhat frustrating for the students, who usually want to begin coding as soon as the semester starts. A better approach might involve adding a course in systems analysis and design as a prerequisite.

Project Team Sizes

An interesting phenomenon is that the accomplishments of the team seem to be more related to how well the team works together, rather than the number of students on the team. Teams seem to function well when there are not more than four students on them, particularily if the students are of equal ability. The *prima donna* approach - surrounding one strong producer with team members of lesser abilities, doesn't seem to work well in this environment. Neither do large teams work well - large groups (over five individuals) tend to break into two or more factions who spend most of their time intellectualizing over the number of bits which can be placed on the head of an RS232 pin. Some of the most surprising project efforts have come from two-person teams, regardless of the size of the problem.

Project Notebook

During the teaching of this course, the project notebook (Appendix A) has become an indispensible tool. It serves to lift the software development effort to the level of being highly visible, although we are still quite a ways from a "public science"[Wienberg, 1971]. The notebook has a standard set of forms which allow the students to "cookbook" their way through the first stages of requirements definition. It also contains the data flow diagrams, the data element descriptions, the file descriptions, and the module narratives. The project notebooks stay on display in the computer science department, where they can be inspected by other teams and students. During the design phase, the teams are periodically graded on how complete their project notebooks are. At the completion of the project, users are given a copy of the project notebook.

7. CONCLUSIONS

In general, the real life project experience works well. Considering the size and nature of the projects and the students' lack of experience, the resulting designs have been reasonably well structured, modular, and functional. The preliminary designs are usually overly ambitious. During the fall semester, the design teams tend to add functions to the system; during the spring semester, the implementation teams tend to delete functions. Seldom does the development reach the program product level [Brooks, 1975]; usually the students wind up implementing a bare-bones prototype of the version of the system which was designed.

Support for the course and the student projects remains strong in the local academic and business communities. Many students have been hired by the organizations for which they did a project. (In one case, however, a user manager informed us that he was so impressed by the students he wasn't going to make them any offers - he was afraid that they were so high-powered that they would find his type of application uninteresting.) At a minimum, satisfied users are asked to send each individual team member a letter acknowledging his or her accomplishments; these letters quickly find their way to the students' resumes. Both students and recruiters have indicated that the project experience seems to equip students with the software development *patois* necessary for effective peer level interviews on plant visits.

The course seems to be well accepted by the students, judging from their enthusiasm, the amount of effort put into the projects, and the high percentage of seniors who enroll for the optional spring semester course.

Hopefully, as we learn more about the process of developing software, courses such as this can produce professionals who experience software development projects not as protracted periods of exigence, but rather as intellectual challenges to be met with energy, excitement, and enthusiasm.

APPENDIX A

SOFTWARE ENGINEERING PRINCIPLES

COURSE OUTLINE

Module 1: Introduction

- Software Development Evolution
- The Software Crisis
- Software Life Cycle: An Introduction
- Software Development Teams: Job Titles and Responsibilities
- Software Life Cycle: A Second Look

Module 2: Project Organizing, Planning, and Estimating

- Work Breakdown Structures
- PERT/CPM Scheduling
- Estimating Lines of Code, Activity Durations, and Resources
- Project Monitoring Techniques

Module 3: Communicating with the User

- Interviewing Techniques
- Making Formal Presentations
- Structured Walkthroughs and Design Reviews
- Preparing the User Manual

Module 4: Defining the Problem

- High Level Problem Representation Techniques
- Layouts: Screens, Reports, and Files
- Establishing System Objecitves

Module 5: Defining System Requirements

- Structured Systems Analysis
- Report Prototyping
- Data Dictionaries
- Structured English
- Decision Tables

Module 6: Designing Software Architectures

- Modularity and Functionality
- Coupling and Cohesion
- Representing the Design Graphically
- Data Flow Design Techinques
- Data Structure Design Techniques
- Module Descriptions

Module 7: Signing Up

Project Descriptions
Project Selections
Selection of Team Leaders

Module 8: Working Sessions

Module 9: Final Presentations

APPENDIX B

WORK BREAKDOWN STRUCTURE

FOR

STANDARD PROJECT ACTIVITIES

S1 - PROJECT MANAGEMENT

- S11 - Developing Project Work Breakdown Structure
- S12 - Project Planning
- S13 - Project Scheduling
- S14 - Project Estimating
- S15 - Completing Project Notebook Entries

S2 - PROBLEM ASSESSMENT

- S21 - Meetings with user personnel to define problem
- S22 - Meetings with user personnel to establish requirements
- S23 - Drafting input, file, or output layouts
- S24 - Team discussions of problem or requirements
- S25 - Preparation of graphical problem representations
- S27 - Presentation of problem representations to users

S3 - REQUIREMENTS ANALYSIS

- S31 - User interviews to understand problem details
- S32 - Preparation of data flow diagrams
- S33 - Refinement of Reports, Screens, Files
- S34 - Preparation of Data Dictionary entries
- S35 - Preparation of data structure charts
- S36 - Preparation of Structured English Process Descriptions
- S37 - Preparation of Decision Tables
- S38 - Presentation of problem structures to users
- S39 - Preparation of User Manual (rough draft)

S4 - SOFTWARE ARCHITECTURE DESIGN

- S41 - Constructing Module Hierarchy Chart
- S42 - Preparing module function narratives
- S43 - Preparation of module pseudo-code
- S44 - Preparation of HIPO diagrams
- S45 - Team meetings to discuss design
- S46 - Design reviews with users

S5 - CODING

- S51 - Module coding
- S52 - Reading module code by other than writer
- S53 - Entry of modules into computer
- S54 - Module compilations
- S55 - Structured Walkthroughs for code

S6 - TESTING

- S61 - Development of system test plan
- S62 - Development of system test data
- S63 - Development of acceptance test plan
- S64 - Development of acceptance test data
- S65 - System testing
- S66 - Acceptance testing
- S67 - Reviewing test results

S7 - IMPLEMENTATION

- S71 - Development of conversion plan
- S72 - Conversion Activities
- S73 - User Manuals (Final form)
- S74 - Prototype Implementation
- S75 - Model Office
- S76 - User Training
- S77 - Parallel Runs

S8 - MAINTENANCE

- S81 - Fixing design errors
- S82 - Fixing coding errors
- S83 - Enhancements to adapt to change in user's needs
- S84 - Fine tuning for speed or efficiency

APPENDIX C

PROJECT NOTEBOOK CONTENTS

Section 1		
	Brief Problem Narrative	
	Graphical Problem Representation (Leighton, HIPO, or Data Flow)	
Section 2		
	Project Description	SE Form 1.1
	System Description	SE Form 1.2
	User Statement of Objectives	
	User Organization Description	SE Form 2.1
	Key User Personnel	SE Form 2.2
Section 3		
	System Input Requirements	SE Form 4.1
	System Output Requirements	SE Form 4.2
	System Processing Requirements	SE Form 4.3
Section 4		
	Output Layouts	
	Input Layouts	
	File Layouts	
	Jackson Data Structure Charts for System I/O	
Section 5		
	Leveled Data Flow Diagrams	SE Form 6.1
	Data Element Descriptions	SE Form 6.2
	Data Flow Descriptions	SE Form 6.3
	Data Structure Descriptions (Structured English)	SE Form 6.4
	Process Descriptions (Structured Engligh)	SE Form 6.5
Section 7		
	Software Architecture Charts	
	Module Functional Narratives	
	HIPO Charts	
	Module Pseudo-code	
Appendix A		
	Project Work Breakdown Structure	
	Project PERT/CPM Chart	
	Weekly Planned Activity Report	
	Individual Student Time Cards	
Appendix B		
	Contact Worksheets (Raw notes from interviews, meetings, design reviews, etc.)	SE Form 3.0

REFERENCES

Boehm, B.W.,*Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

Boehm, B.W.,"Software Engineering," *IEEE Transactions on Computers*, C-25:1976.

Brooks, F.P., *The Mythical Man-Month*, Addison Wesley, Reading, MA, 1975.

DeMarco, T., *Structured Analysis and System Specification*, Yourdon Press, New York, 1978.

Freedman, D.P. and Weinberg, G.P., *Handbook of Walkthrus, Inspections, and Technical Reviews, 3rd edition*, Little, Brown, and Company, Boston, MA, 1982.

Freeman, P., "Realism, Style,and Design: Packing It into a Constrained Course, *Proceedings, ACM SIGCSE-SIGCUE Technical Symposium on Computer Science and Education*, May, 1978.

Gane,C. and Sarsen,T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, New York, 1979.

Gladden, G.R.,"Stop the Life-Cycle, I Want to Get Off," *ACM Software Engineering Notes*, vol. 7, no. 2, April, 1982.

Halstead, M.H., *The Elements of Software Science*, Elsevier, New York, 1977.

HIPO - A Design Aid and Documenation Technique, Order Number GC 20-1851, IBM Corporation, White Plains, New York.

Inmon, W.H., and Friedman, L.J., *Design Review Methodology for a Data Base Environment*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

Jackson, M.H., *Principles of Program Design*, Academic Press, New York,1975.

Jackson, M.H. and McCracken, D.D.,"Life Cycle Concept Considered Harmful", *ACM Software Engineering Notes*, vol.7, no. 2, April, 1982.

Jensen, R. W. and Tonies, C.C., *Software Engineering*, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Kant, E., " A Semester Course in Software Engineering," *ACM Software Engineering Notes*, vol. 6, no. 4, August, 1981.

McCabe, T., "A Complexity Measure,", *IEEE Transactions on Software Engineering*, SE-2, 1976.

Mills, H.D.,"Software Engineering", *Science* 195: 1201, 1977.

Orr, K.D., *Structured Systems Development*, Yourdon Press, New York, 1977.

Peters, L.J., *Software Design:Methods and Techniques*, Yourdon Press, New York, 1981.

Pressman, R. S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, New York, 1982.

Scott, L.R., "An Engineering Methodology for Presenting Software Functional Architecture," *Proceedings of the Third International Conference on Software Engineering*, IEEE Computer Society, New York, 1978.

Stevens, W. P. , Myers, G. T., and Constantine, L.L.,"Structured Design," *IBM Systems* Journal 13: 1974.

Stoy, J.F., "HIPO and Integrated Design," *IBM Systems Journal* 15: 1976.

Warnier, J.D.,*The Logical Construction of Programs*, Van Nostrand, New York, 1976.

Weinberg, G.M., *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.

Yourdon, E., *Structured Walkthroughs*, Yourdon Press, New York, 1978.

Yourdon, E. ,and Constantine, L. L., *Structured Design*, Yourdon Press, New York, 1978.

The following papers are currently available in the Edwin L. Cox School of Business Working Paper Series.

79-100 "Microdata File Merging Through Large-Scale Network Technology," by Richard S. Barr and J. Scott Turner

79-101 "Perceived Environmental Uncertainty: An Individual or Environmental Attribute," by Peter Lorenzi, Henry P. Sims, Jr., and John W. Slocum, Jr.

79-103 "A Typology for Integrating Technology, Organization and Job Design," by John W. Slocum, Jr., and Henry P. Sims, Jr.

80-100 "Implementing the Portfolio (SBU) Concept," by Richard A. Bettis and William K. Hall

80-101 "Assessing Organizational Change Approaches: Towards a Comparative Typology," by Don Hellriegel and John W. Slocum, Jr.

80-102 "Constructing a Theory of Accounting--An Axiomatic Approach," by Marvin L. Carlson and James W. Lamb

80-103 "Mentors & Managers," by Michael E. McGill

80-104 "Budgeting Capital for R&D: An Application of Option Pricing," by John W. Kensinger

80-200 "Financial Terms of Sale and Control of Marketing Channel Conflict," by Michael Levy and Dwight Grant

80-300 "Toward An Optimal Customer Service Package," by Michael Levy

80-301 "Controlling the Performance of People in Organizations," by Steven Kerr and John W. Slocum, Jr.

80-400 "The Effects of Racial Composition on Neighborhood Succession," by Kerry D. Vandell

80-500 "Strategies of Growth: Forms, Characteristics and Returns," by Richard D. Miller

80-600 "Organization Roles, Cognitive Roles, and Problem-Solving Styles," by Richard Lee Steckroth, John W. Slocum, Jr., and Henry P. Sims, Jr.

80-601 "New Efficient Equations to Compute the Present Value of Mortgage Interest Payments and Accelerated Depreciation Tax Benefits," by Elbert B. Greynolds, Jr.

80-800 "Mortgage Quality and the Two-Earner Family: Issues and Estimates," by Kerry D. Vandell

80-801 "Comparison of the EEOCC Four-Fifths Rule and A One, Two or Three σ Binomial Criterion," by Marion Gross Sobol and Paul Ellard

80-900 "Bank Portfolio Management: The Role of Financial Futures," by Dwight M. Grant and George Hempel

80-902 "Hedging Uncertain Foreign Exchange Positions," by Mark R. Eaker and Dwight M. Grant

80-110 "Strategic Portfolio Management in the Multibusiness Firm: An Implementation Status Report," by Richard A. Bettis and William K. Hall

80-111 "Sources of Performance Differences in Related and Unrelated Diversified Firms," by Richard A. Bettis

80-112 "The Information Needs of Business With Special Application to Managerial Decision Making," by Paul Gray

80-113 "Diversification Strategy, Accounting Determined Risk, and Accounting Determined Return," by Richard A. Bettis and William K. Hall

80-114 "Toward Analytically Precise Definitions of Market Value and Highest and Best Use," by Kerry D. Vandell

80-115 "Person-Situation Interaction: An Exploration of Competing Models of Fit," by William F. Joyce, John W. Slocum, Jr., and Mary Ann Von Glinow

80-116 "Correlates of Climate Discrepancy," by William F. Joyce and John Slocum

80-117 "Alternative Perspectives on Neighborhood Decline," by Arthur P. Solomon and Kerry D. Vandell

80-121 "Project Abandonment as a Put Option: Dealing with the Capital Investment Decision and Operating Risk Using Option Pricing Theory," by John W. Kensinger

80-122 "The Interrelationships Between Banking Returns and Risks," by George H. Hempel

80-123 "The Environment For Funds Management Decisions In Coming Years," by George H. Hempel

81-100 "A Test of Gouldner's Norm of Reciprocity In A Commercial Marketing Research Setting," by Roger Kerin, Thomas Barry, and Alan Dubinsky

81-200 "Solution Strategies and Algorithm Behavior in Large-Scale Network Codes," by Richard S. Barr

81-201 "The SMU Decision Room Project," by Paul Gray, Julius Aronofsky, Nancy W. Berry, Olaf Helmer, Gerald R. Kane, and Thomas E. Perkins

81-300 "Cash Discounts To Retail Customers: An Alternative To Credit Card Performance," by Michael Levy and Charles Ingene

81-400 "Merchandising Decisions: A New View of Planning and Measuring Performance," by Michael Levy and Charles A. Ingene

81-500 "A Methodology For The Formulation And Evaluation Of Energy Goals And Policy Alternatives For Israel," by Julius Aronofsky, Reuven Karni, and Harry Tankin

81-501 "Job Redesign: Improving The Quality of Working Life," by John W. Slocum, Jr.

81-600 "Managerial Uncertainty and Performance," by H. Kirk Downey and John W. Slocum, Jr.

81-601 "Compensating Balance, Rationality, and Optimality," by Chun H. Lam and Kenneth J. Boudreaux

81-700 "Federal Income Taxes, Inflation and Holding Periods For Income-Producing Property," by William B. Brueggeman, Jeffrey D. Fisher, and Jerrold J. Stern

81-800 "The Chinese-U.S. Symposium On Systems Analysis," by Paul Gray and Burton V. Dean

81-801 "The Sensitivity of Policy Elasticities to the Time Period Examined in the St. Louis Equation and Other Tests," by Frank J. Bonello and William R. Reichenstein

81-900 "Forecasting Industrial Bond Rating Changes: A Multivariate Model," by John W. Peavy, III

81-110 "Improving Gap Management As A Technique For Reducing Interest Rate Risk," by Donald G. Simonson and George H. Hempel

81-111 "The Visible and Invisible Hand: Source Allocation in the Industrial Sector," by Richard A. Bettis and C. K. Prahalad

81-112 "The Significance of Price-Earnings Ratios on Portfolio Returns," by John W. Peavy, III and David A. Goodman

81-113 "Further Evaluation of Financing Costs for Multinational Subsidiaries," by Catherine J. Bruno and Mark R. Eaker

81-114 "Seven Key Rules For Successful Stock Market Speculation," by David Goodman

81-115 "The Price-Earnings Relative As An Indicator of Investment Returns," by David Goodman and John W. Peavy, III

81-116 "Strategic Management for Wholesalers: An Environmental Management Perspective," by William L. Cron and Valarie A. Zeithaml

81-117 "Sequential Information Dissemination and Relative Market Efficiency," by Christopher B. Barry and Robert H. Jennings

81-118 "Modeling Earnings Behavior," by Michael F. van Breda

81-119 "The Dimensions of Self-Management," by David Goodman and Leland M. Wooton

81-120 "The Price-Earnings Relatives - A New Twist To The Low-Multiple Strategy," by David A. Goodman and John W. Peavy, III.

82-100 "Risk Considerations in Modeling Corporate Strategy," by Richard A. Bettis

82-101 "Modern Financial Theory, Corporate Strategy, and Public Policy: Three Conundrums," by Richard A. Bettis

82-102 "Children's Advertising: The Differential Impact of Appeal Strategy," by Thomas E. Barry and Richard F. Gunst

82-103 "A Typology of Small Businesses: Hypothesis and Preliminary Study," by Neil C. Churchill and Virginia L. Lewis

82-104 "Imperfect Information, Uncertainty, and Credit Rationing: A Comment and Extension," by Kerry D. Vandell

82-200 "Equilibrium in a Futures Market," by Jerome Baesel and Dwight Grant

82-201 "A Market Index Futures Contract and Portfolio Selection," by Dwight Grant

82-202 "Selecting Optimal Portfolios with a Futures Market in a Stock Index," by Dwight Grant

82-203 "Market Index Futures Contracts: Some Thoughts on Delivery Dates," by Dwight Grant

82-204 "Optimal Sequential Futures Trading," by Jerome Baesel and Dwight Grant

82-300 "The Hypothesized Effects of Ability in the Turnover Process," by Ellen F. Jackofsky and Lawrence H. Peters

82-301 "Teaching A Financial Planning Language As The Principal Computer Language for MBA's," by Thomas E. Perkins and Paul Gray

82-302 "Put Budgeting Back Into Capital Budgeting," by Michael F. van Breda

82-400 "Information Dissemination and Portfolio Choice," by Robert H. Jennings and Christopher B. Barry

82-401 "Reality Shock: The Link Between Socialization and Organizational Commitment," by Roger A. Dean

82-402 "Reporting on the Annual Report," by Gail E. Farrelly and Gail B. Wright

82-403 "A Linguistic Analysis of Accounting," by Gail E. Farrelly

82-600 "The Relationship Between Computerization and Performance: A Strategy For Maximizing The Economic Benefits of Computerization," by William L. Cron and Marion G. Sobol

82-601 "Optimal Land Use Planning," by Richard B. Peiser

82-602 "Variances and Indices," by Michael F. van Breda

82-603 "The Pricing of Small Business Loans," by Jonathan A. Scott

82-604 "Collateral Requirements and Small Business Loans," by Jonathan A. Scott

82-605 "Validation Strategies For Multiple Regression Analysis: A Tutorial," by Marion G. Sobol

82-700 "Credit Rationing and the Small Business Community," by Jonathan A. Scott

82-701 "Bank Structure and Small Business Loan Markets," by William C. Dunkelberg and Jonathan A. Scott

82-800 "Transportation Evaluation in Community Design: An Extension with Equilibrium Route Assignment," by Richard B. Peiser

82-801 "An Expanded Commercial Paper Rating Scale: Classification of Industrial Issuers," by John W. Peavy, III and S. Michael Edgar

82-802 "Inflation, Risk, and Corporate Profitability: Effects on Common Stock Returns," by David A. Goodman and John W. Peavy, III

82-803 "Turnover and Job Performance: An Integrated Process Model," by Ellen F. Jackofsky

82-804 "An Empirical Evaluation of Statistical Matching Methodologies," by Richard A. Barr, William H. Stewart, and John Scott Turner

82-805 "Residual Income Analysis: A Method of Inventory Investment Allocation and Evaluation," by Michael Levy and Charles A. Ingene

82-806 "Analytical Review Developments in Practice: Misconceptions, Potential Applications, and Field Experience," by Wanda Wallace

82-807 "Using Financial Planning Languages For Simulation," by Paul Gray

82-808 "A Look At How Managers' Minds Work," by John W. Slocum, Jr. and Don Hellriegel

82-900 "The Impact of Price Earnings Ratios on Portfolio Returns," by John W. Peavy, III and David A. Goodman

82-901 "Replicating Electric Utility Short-Term Credit Ratings," by John W. Peavy, III and S. Michael Edgar

82-902 "Job Turnover Versus Company Turnover: Reassessment of the March and Simon Participation Model," by Ellen F. Jackofsky and Lawrence H. Peters

82-903 "Investment Management By Multiple Managers: An Agency-Theoretic Explanation," by Christopher B. Barry and Laura T. Starks

82-904 "The Senior Marketing Officer - An Academic Perspective," by James T. Rothe

82-905 "The Impact of Cable Television on Subscriber and Nonsubscriber Behavior," by James T. Rothe, Michael G. Harvey, and George C. Michael

82-110 "Reasons for Quitting: A Comparison of Part-Time and Full-Time Employess," by James R. Salter, Lawrence H. Peters, and Ellen F. Jackofsky

82-111 "Integrating Financial Portfolio Analysis with Product Portfolio Models," by Vijay Mahajan and Jerry Wind

82-112 "A Non-Uniform Influence Innovation Diffusion Model of New Product Acceptance," by Christopher J. Easingwood, Vijay Mahajan, and Eitan Muller

82-113 "The Acceptability of Regression Analysis as Evidence in a Courtroom - Implications for the Auditor," by Wanda A. Wallace

82-114 "A Further Inquiry Into The Market Value and Earnings' Yield Anomalies," by John W. Peavy, III and David A. Goodman

82-120 "Compensating Balances, Deficiency Fees and Lines of Credit: An Operational Model," by Chun H. Lam and Kenneth J. Boudreaux

82-121 "Toward a Formal Model of Optimal Seller Behavior in the Real Estate Transactions Process," by Kerry Vandell

82-122 "Estimates of the Effect of School Desegregation Plans on Housing Values Over Time," by Kerry D. Vandell and Robert H. Zerbst

82-123 "Compensating Balances, Deficiency Fees and Lines of Credit," by Chun H. Lam and Kenneth J. Boudreaux

83-100 "Teaching Software System Design: An Experiential Approach," by Thomas E. Perkins