TABLE 2 (CONT.)

Training
Organizations in this classification have more on-the-job training than formal classroom training. Most training is informal, including acquaintance with the organization and the systems that are in production. The only formal training is involved with programming, documentation, and testing standards.

Standards
All companies but one in this classification have a very formalized set of programming standards. In one organization, structured walkthroughs check very carefully the adherence of the product to standards. Since no organization at this time has a comprehensive methodology for data processing department development projects, no standards exist for the Systems Analysis or Design phases.

View of the Future
These organizations feel that a structured approach to software creation is needed, but can only be implemented as far as it is cost justified. This classification, with the exception of one organization, does not feel as strongly about the use of fourth-generation languages as the organizations of the previous classification feel.

Education of Potential Personnel
The companies in this classification feel that college graduates are trained very well in the technical aspects of data processing. The increased educational requirements that need to be implemented are in the areas of business education, communication, and psychology. The companies feel that computer science graduates should be businessmen as well as data processing personnel. Also, the graduates should have experience with major business applications sometime in their educational career. In order to communicate with users and solve their problems, graduates should know how to effectively communicate and understand the psychology of the users.

The companies of this class have great awareness of software engineering techniques, and are studying what effects they might have on their operations. They are very concerned with cost justification of implementing the techniques, but are beginning to implement some of the more modern methods to help improve
the quality of the software produced and productivity of data processing personnel.

3. High Development Companies. The companies in the high development classification have either developed their own comprehensive methodologies from combinations of many different techniques, or use a published methodology. Personnel are trained in their respective techniques, and the organizations try to remain on the leading edge of software engineering techniques. The results of the study for this classification follow.

TABLE 3
HIGH-DEVELOPMENT COMPANY ANALYSIS

Data Processing Organization
With the exception of one organization, the employees have the title of programmer/analyst. In all organizations, analysis and programming are viewed to be separate functions. The one company that does not have programmer/analysts have only programmers in the data processing department. The analysts were at one time members of the data processing departments, and have now been moved permanently to the user departments.

Planning and Control
The projects in this classification are generally user-initiated, with data processing taking responsibility after initiation. In one organization, the user department is responsible for the analysis and feasibility (the what of the project) and data processing is responsible for programming and implementation (the how of the project). Responsibility for a project depends on its estimated manpower-time requirements in another organization. If a project is estimated at under six months, it is handled by the maintenance group of programmer/analysts with data processing management sponsorship. An estimate of over six months requires senior management, user, and data processing management sponsorship and is handled by a systems development group with a project manager from outside data processing. Another organization has a steering committee made up of senior management from the different corporate areas that require data processing services. All projects and large modifications go through the committee for approval, prioritizing, and the commitment of resources. This committee also tries
TABLE 3 (CONT.)

to solve conflicts in the assignment of personnel to specific areas and projects. In another organization the divisional data processing department does a systems study before taking a report of estimated costs and potential benefits to a company-wide steering committee for approval. Although most projects in the organization are user-initiated, data processing initiates projects that would help accomplish the organizational goals identified by top management.

User Involvement
In all the organizations of this classification there seems to be a high degree of user involvement. User sign-off is required after almost every phase of application system development. The user is either entirely responsible for the requirements definition and development of systems specification or plays a major working role. The amount of formal documentation for this classification also points to more user involvement, as users need such documentation in order to understand the system and be able to sign-off on the specified phases. The user also plays a large role in testing and implementation of the new system, to be certain that he can control and utilize the system, and that it gives the specified results.

Initiation of Projects
Projects in organizations of this classification are for the most part user-initiated. The process of initiation is very formalized. In one organization, a steering committee is formed in each user department to identify problems for possible solution by an application project. Every project involves much documentation and review, plus formal procedures for approval and commitment of resources.

Methodology or Technique
All the organizations in this grouping have both comprehensive project methodologies and techniques for specific software upgrades. Two of the companies use published methodologies, one company uses a combination of methodologies, and one company uses a methodology that was developed in-house. The first organization uses a customized form of IBM's MADP methodology. Along with its phaseology, the organization also institutes structured programming and structured walkthroughs. The use of structured analysis and HIPO charts is encouraged. There has been some attempt to implement the use of structure charts before coding, but this is not working as well as was hoped.

The second organization uses, in conceptual form, the PRIDE methodology by M. Bryce & Associates. Although the organization does not follow the methodology
TABLE 3 (CONT.)

verbatim, the general breakdown of system activities and sub-system breakdown is followed. PRIDE constraints ensure structured programming, and other techniques are encouraged when working on larger projects. The organization is experiencing a gradual shift in time division among development phases. More time is now being spent in analysis, rather than in coding.

The third organization uses a combination of techniques for software development. In analysis, Tom DeMarco's Structured Analysis and System Specification is used. For design, Constantine and Yourdon's Structured Design provides the optimal characteristics. HIPO diagrams are used after Structured Design, and Nassi-Schneiderman charts are used as the final step before coding. Structured programming is encouraged, and there is a team assigned to each project. This organization has tried many techniques, and changes in the process of software development happen fast and often. After coding, structured walkthroughs are used to help verify the solution to the user problem. The documentation required with the use of these techniques provides a vehicle for communication with non-technical personnel.

The fourth organization has developed its own project management and development methodology that is customized to this particular department. The methodology has a specific phaseology, including the proposal phase, feasibility phase, preliminary design phase, detail design phase, implementation phase, conversion phase, performance review phase, and maintenance phase. Each phase has very detailed requirements pertaining to what should be included in the Findings and Recommendations, a report put out in the analysis phases, or in the specifications put out from other phases. The organization is in the process of compiling a manual of procedures, constraints, and expected output from each phase. This methodology requires the use of Structured Analysis and Design and also structured programming. Gane and Sarson's version of DeMarco's Structured Analysis and System Specification is used in this organization, and structured walkthroughs are also used.

Attitude Toward Techniques
All the organizations have a strong commitment to their software engineering techniques, both philosophic and economic. Without exception they believe that the techniques have improved the quality of the software produced and have saved the company money. All are constantly looking to upgrade the quality of the software produced, so are either still looking at other techniques or are enforcing more strictly the techniques already in use.
TABLE 3 (CONT.)

Training
Organizations in this grouping, with the exception of one, have classroom facilities for formal training. Classes with instructors, along with audio-visual courses, are offered in many aspects of software creation from analysis to coding. The training is customized to the individual employee, with some general training about the standards and procedures of the particular organization. On-the-job training is also a part of the training program for all the organizations.

Standards
All the organizations interviewed in the high development classification have very detailed standards for every phase of the software life cycle. These are sometimes provided by the comprehensive methodology in use and other times a result of the company's goals. There is specific output from each phase of the cycle, and the output is reviewed and approved according to company standards. Many times the most time in classroom training is spent teaching the specific standards of the organization, and many manuals are printed for standards reference.

View of the Future
Two of the companies are researching the use of fourth-generation languages and "end-user" programming. One company is concerned with finding a methodology that addresses the purchase and implementation of software packages. The organization sees major application development as a thing of the past because of the availability of advanced software packages. The final organization is looking for more efficient project tracking and estimation techniques.

Education of Potential Personnel
Without an exception, all the organizations in this grouping feel that college graduates need more experience in business applications. There was suggestion for possible separation of the computer science major into applied and pure computer science. The organizations also felt that interpersonal communication skills should be stressed.

Software engineering techniques have been used in high-development companies for many years. Their experiences with the techniques have justified their use, both with increased quality of software and increased productivity. The software produced is
more easily maintained and modified, and solves the user problems more efficiently and completely.

C. RELATIONSHIP OF COMPANY CHARACTERISTICS TO TECHNIQUES CHOSEN

The following sections will attempt to analyze the results of the study with respect to company characteristics. The affects of staff size, hardware size, and percentage of purchased software on the choice and advancement of software engineering techniques will be analyzed and presented.¹

1. Staff size. Each company was questioned on its staff size in the data processing department. Comparisons will be made on size excluding the staff in operations and data entry, as those employees are not pertinent to this study.

The organizations can be divided into three classifications of staff size. They are as follows:

1. Small 20 persons or under  3 companies
2. Medium 21-50 persons  2 companies
3. Large 51 persons and over  4 companies

With the exception of one, companies with small staff size do not use software engineering techniques. They do not separate in function the analysis and programming in a formal manner. They are aware of the modern techniques, but do not feel that they are cost justified. These companies do not engage in formal training classes, but do have basic programming standards. The one exception to this classification does use some structured approach to analysis, but this is not carried through to the entire software life cycle.

¹Refer to Appendix D for relationship charts.
The two companies in the medium classification are very diverse. One company is beginning to implement structured analysis to complement already implemented structured programming and testing. Informal training classes are conducted in these techniques and programming standards are enforced through structured walkthroughs. The other organization in this classification is following through on the "end-user" programming concept. Since no application development takes place in the data processing department (the users control project development) software engineering techniques are not considered as necessary as they would be if major development projects were data processing responsibility. Personnel are trained informally in fourth-generation language skills.

The four companies that are in the large staff classification correspond directly to the four companies that were listed as being highly developed in their use of software engineering techniques. These companies use many of the techniques listed in this paper, plus comprehensive project management methodologies. They have formal classroom training in both software creation techniques and organization documentation, naming, and testing standards. They feel that the implementation of techniques and standards is justified by the increased productivity of the staff, increased quality of the software, and the elimination of some of the more common errors in problem-solving.

2. Hardware size. The criteria used for the breakdown of the participants by hardware size include number of mainframes, number of mini- and personal computers, amount of networking or
distributed processing, and number of peripheral units. Three classifications have been determined.

1. Small 3 companies
2. Medium 3 companies
3. Large 3 companies

Again, all but one of the companies with relatively small hardware configurations are in the low-development class with respect to software engineering techniques. The one exception is a company that is of medium development, using some techniques, but not totally committed to the software engineering discipline.

Two of the companies with medium hardware configurations, as compared to others in the study, are of medium development also. The third company has highly developed software engineering techniques.

All of the companies with very large hardware configurations are in the highly developed classification. The applications developed for these systems are naturally more complex and need more planning and control.

3. Percentage of Purchased Applications Software. The need for advanced software engineering techniques often depends on the amount of software development that is done. If an organization believes in and is able to purchase software packages, the need for implementing specialized techniques and methodologies may not exist. Classifications are listed in the percentage of applications purchased from outside software services organizations. The classifications are as follows:
1. Low  20% or under purchased   4 companies
2. Medium  21%-50% purchased   3 companies
3. High  51% and over purchased   2 companies

These classifications and the division of the companies can be directly related to the companies' use of the software engineering techniques. Those companies purchasing less than 20% of their software must therefore develop over 80%. These companies use more advanced techniques for software development. Those purchasing from 51% to 100% of applications software are low-development companies, using very few techniques.

D. Conclusions

The factor having the most effect on the use of software engineering techniques is the percentage of software packages that are purchased instead of developed in-house. The amount of development that an organization does dictates the amount of structure and discipline that the development process needs. It is obvious that if an organization only develops a few small applications a year, there would be no cost justification for the implementation of formalized structured techniques with complete documentation and training. The complexity and size of the applications would also play a major role in the justification of software engineering techniques, but as those characteristics are hard to measure consistently from organization to organization, they will not be addressed here.

Hardware size has a definite affect on the development of technique usage, but it is not strong and may be a result instead of a cause of technique implementation. A large configuration may just be indicative of larger company size. Since an
application only views a portion of the system and does not deal with all facilities at once, the hardware size may not be as great a factor as suspected.

Staff size can be a very big consideration when deciding on and implementing structured techniques. When staff size is small, communication between users and data processing personnel, and among the DP personnel is much easier to manage both physically and with respect to paper work. As staff size grows, users and data processing departments may be more physically separated, and may have a harder time getting together. This highlights the need for more formal documentation of the system. With more of the communication done on paper, there is better control of the process and less time is spent in personal communication. Large staff size also implies the need for better project management techniques. There are more projects and more personnel to control and more resources to allocate. A structured approach may be the only way to handle a large staff situation.

In conclusion, those companies that have a combination of larger staff and hardware size, plus a low percentage of purchased software tend to have more software engineering techniques in use. These are also the larger companies that have been involved in data processing for a longer time. The complexity and number of systems needed to run the corporation effectively make structured approaches to software creation essential.

Since all the companies using software engineering techniques agreed that software quality was improved by their
use, and money was saved, why aren't small companies using the same formalized techniques?

The initial implementation cost is a major factor in smaller companies' choices. The smaller, simpler systems produced by these companies may not warrant the formalized approaches. These companies can also usually find the applications they need on the software market and therefore purchase instead of develop. Some of the medium-sized companies are just now moving into the stage of software usage where more complex and integrated systems are needed, and therefore are just beginning to realize the need for such techniques. The small companies do not integrate the systems as much, so the users utilizing "user-friendly" languages can do some of their own programming. In these cases software engineering techniques are not as necessary.

D. LIMITATIONS OF THE SURVEY DATA

The conclusions drawn from the data collected must be tempered by the limitations of the study conducted. There are four major limitations:

1. sample size - due to the limited time available for the completion of this study, only nine companies could be examined. Time was a factor in the number of interviews that could be set up at mutual company and personal convenience, and also in the amount of data that could be organized and analyzed.

2. length of interviews - partially due again to time constraints and partially due to the geographical distance between the parties involved, limited time was devoted to each individual interview. Better conclusions might have been drawn if more than one interview had been conducted with each company. A second interview would have provided for follow through on points brought up in the first interview.
3. geographical distribution - the interviews were conducted in a limited geographical area, northeastern and central Indiana. The conclusions cannot be generalized to all corporate organizations.

4. subjectivity - much of the data concerning the general attitudes of an organization toward software engineering is subjective in nature, therefore making any conclusions drawn from that data sensitive to the subjectivity of all parties involved.

Although these limitations have affected the data and its analysis, the conclusions drawn from the data presented are valid for the sample drawn.

The summary and conclusions of this research paper, along with suggestions for the future will be presented in the next section.
VI. SUMMARY AND CONCLUSIONS

A. SUMMARY

The information presented in this report by no means intends to be an exhaustive study on software engineering techniques. It is intended to present an overview of the impetus for the development of such techniques, and details of some representative techniques for the phases of the software life cycle.

The overall objective of the study of the cross-section of corporate data processing environments was to try and maintain closer contact with industry and its applications of software engineering techniques. It is the responsibility of the academic community to develop new techniques and concepts through research, but unless the results of this research are applicable in business, they are doomed to be caught forever in theory. Thus by studying the techniques that are implemented in industry at the present time, and obtaining industry views of the future direction of data processing, decisions on future research and teaching material for students of computer science can be made with better results.

B. FUTURE TRENDS IN APPLICATIONS DEVELOPMENT

In this section, four trends for the future will be discussed. All are related, for they stem from a common cause: user dissatisfaction with their data processing organizations.

"The application development backlog is overwhelming in many organizations. DP departments may be lagging three to four years behind in implementing user requests for new
applications." [FINK82] For this reason, and because the costs of new developments done by the DP department are rising so rapidly, user departments are doing more of their own applications development. They are installing mini-computers or buying high-level, end-user oriented languages to be used in a time-sharing environment.

The four future trends that will be discussed provide the user with a method for bypassing the traditional role of the data processing department. It must be stressed that while short term results of this bypass may be extremely good, the long-term effects may be harmful if bypass is not limited to use on small, simple systems. The four trends are:

1. fourth-generation applications development
2. the information center concept
3. package purchase
4. decision support systems

1. Fourth-generation Applications Development. Before the advent of fourth-generation systems for applications development, the modern development system was the integration of many development tools. The integration included "extending the standard language [such as COBOL or PL/I], coupling it to a data dictionary/directory and DBMS and supporting the development process in an interactive environment." [KAUF82]

The fourth-generation systems attempt to integrate all of the above characteristics into one package or system. "All have been directed toward end-user involvement in the applications development process . . . ." [KAUF82] The ideal fourth-generation system has an on-line support environment providing:
1. menus and help services to coach the inexperienced user
2. an efficient command language for the experienced developer
3. language-sensitive editors that streamline the programming process
4. a flexible printing capability that permits routing and on-line browsing of lists and reports
5. integrated, active data dictionary/directory control and coordination[KAUF82]

These integrated tools can make applications development by the user very possible, as "user-friendly" prompts and explanations can guide the user through the necessary steps to develop an application.

How does the use of fourth-generation systems affect software engineering concepts? As was mentioned previously, fourth-generation applications development may cause more problems than benefits in the long run. If fourth-generation development is used extensively and exclusively in a larger organization with complex systems, the result could be the development of many incompatible, non-integrated systems. Each user area may have its own set of programs, often duplicating the functions of another area's programs. When all the programs use the same database, backup and recovery decisions may not be standard throughout all user areas. Also, a tremendous amount of data processing support services are required for the use of fourth-generation systems.

Based on the considerations made above, software engineering techniques will still be needed with fourth-generation systems. Large applications with company-wide significance will have to be
developed centrally, and software engineering techniques can help control the cost, schedule, and complexity of these systems. The documentation techniques will also be helpful in communicating complex explanations to users from technical personnel. [KAUF82]

2. Information Center. Another trend in data processing is the Information Center concept. In order to decrease the backlog of new applications, the users would begin programming their own needs, and the data processing center would be used for support, expertise, and custodians of the data base or data files. Many times fourth-generation systems are used with the Information Center concept. The responsibility for applications development is shared between management, user, and data processing personnel.

Since this concept is so similar to the fourth-generation system implementation, the effects on software engineering principles are also similar. [KAUF82]

3. Package Purchase. The third future trend is also a result of the need to bypass the traditional role of the data processing department. Organizations are looking closely at the many varied software packages offered on the market today. Many times an application package costs much less to purchase than the in-house development would cost. Trade-offs must be made between cost of the package and modification cost, and the cost of in-house development. Another cost sometimes left out is the price of lost flexibility with a purchased package.
Even though many of the software engineering techniques would not be needed for a purchased package, there are some that would still be applicable. The Systems Analysis phase would still have to be executed, for proper identification of the problem and the requirements for solution is needed before the proper package can be purchased. Also, testing and implementation techniques would still be necessary. There is a great need for the development of a methodology directed specifically at purchasing software. A structured approach to software purchase could save time and money, and ensure that the proper package is purchased.

4. Decision Support Systems. The fourth future trend is the implementation of decision support systems (DSS). These are advanced systems that can create models to simulate business situations and can provide relatively instantaneous summaries of business data. There are five basic capability characteristics of decision support systems:

1. a fully-integrated, multiple source data base containing information about the economy, merger and acquisition data, markets and products data, and internal and external data on marketing, finance, and operations.

2. ad hoc analysis of data base information

3. modeling capabilities

4. rapid cross-functional communication; information from all departments is tied in

5. user-friendliness; a manager is not a computer programmer, so a DSS should be English-based with help facilities[LANE82]

The implementation of this type of DSS can help relieve data processing's backlog of work by putting the power of information
retrieval and analysis into the manager's hands. It will also provide more complete and timely information to the manager in a form he can understand because he designed it.

Again, software engineering techniques are not excluded when a DSS is installed. The development of the DSS and other applications not covered by its functions may still benefit from the cost, quality, and timing control provided by the use of software engineering. [LANK82]

C. SOFTWARE ENGINEERING EDUCATION

Before what to teach software engineers can be decided upon, the functions of a software engineer must first be established.

From an industry point of view, excluding software houses, the following is a list of the software engineer's major functions.

1. expressing oneself in clear English, since over half of the total systems development effort is in producing documentation.

2. developing and validating software requirements and design specifications, which are very important to avoid design errors.

3. developing applications software, as opposed to concern with linkers, loaders, compilers, or operating systems.

4. performing economic analyses, including cost/benefit analyses, sensitivity analyses, and risk analyses.

5. working with project management techniques such as milestones, design reviews, and activity nets.

6. working in groups with emphasis on group communication, interface definition, and system integration.

7. performing software maintenance, as in using, fixing, or modifying a program written some time ago. [BOEH76]
The first section covered will be an outline of general requirements for a computer science major in the applications development option. This outline will not attempt to develop any specific class structures, but will give the basic topics that should be covered for completion of the major. The second section will discuss a new major for computer science using classes that are currently offered at Ball State University, and new classes which have Ball State classes as their bases.

1. **General Requirement.** The general requirements for a computer science major in applications development would be divided into three categories: business requirements, general requirements, and computer science requirements.

The following is a list of the business courses that should be taken for better skills in applications development:

1. managerial and financial accounting
2. principles of management
3. principles of finance
4. principles of economics
5. organizational behavior

The accounting courses are useful in two ways. First, many of the business applications programmed in the corporate environment are based on or get their input from accounting systems. It is therefore important to know and understand accounting principles. Secondly, the estimates used in cost/benefit analysis and also cost/benefit analysis procedure have their basis in accounting procedures. Since cost/benefit analysis and estimates are used in feasibility studies, it is
necessary to understand their concepts before attempting the Systems Analysis phase.

Management principles are important for every applications development student, even those who do not intend to become part of the formal management of an organization. Several courses can be taken in the management area, including large and small office management, small business management, and production and operations management, depending on what environment in which the student wishes to operate. Those students who do not aspire to formal management positions will still find these courses beneficial, as they may become team managers or may benefit from knowing why certain procedures are carried out by their managers.

The principles of finance are again important for two reasons. Financial systems are very common in large organizations, and better understanding of the principles involved in their functions will lead to better designed systems. Also, financial analyses are necessary to justify a systems' development or continued use.

Economic principles are very useful in conducting economic analyses of alternatives. These can be hardware purchase alternatives, software purchase alternatives, or alternatives involving programming considerations. Accounting and financial principles are also helpful in this area.

The study of organizational behavior can help students understand and deal with the human aspects of a business environment. The knowledge provided in this course encompasses the behavior of people in groups inside the organization, with emphasis on leadership and motivation. As many projects are
developed as team efforts, learning how to be a good group leader or functioning member can be very important.

There are three subject areas that should be included in an applications development major under the "general" category. They are:

1. psychology
2. speech
3. technical writing

The psychology course taken should be one dealing in interpersonal relationships. Improved skills in this area will help in dealing with members of a team who must work closely together. Also, the interface with the users can be improved in the student understands how to communicate with them on a human level, taking into account their feelings and biases.

Speech courses can help a student gain confidence in presenting his ideas to a group. The special considerations of small group and technical presentations should be emphasized. These skills may help the student present his product more clearly when involved in feasibility analysis or a structured walkthrough.

Communication with group members and also with the user is enhanced with technical writing skills. The skills can help personnel develop clear, precise specifications and documentation in every phase of the software life cycle. Also, the conversion of a very technical design document into language understandable by the user is much easier if technical writing skills are employed.
The following is a list of topics that should be covered in the computer science courses offered to an applications development major. Although several of the topics could be combined in one course, no attempt has been made in this section to accomplish such combinations to help retain generality.

1. Basic courses in PL/I and other high level languages
2. Assembly language
3. Debugging
4. Problem solving
5. Data Structures
6. Business applications in COBOL
7. Performance evaluation
8. Systems Analysis
9. Notation
10. Abstraction
11. Design
12. Modularization
13. Implementation and testing
14. Modification

This list is not exhaustive, but should provide a good basis for the decision on the content of CS courses. The following paragraphs will highlight some of the topics on the list.

Implementation, testing and modification are three important items on the list. Although they are sometimes hard to simulate in an academic environment, the students who have had exposure to techniques and strategies concerning these areas will be much better prepared for the corporate environment. Sometimes
modification can be as much as 75%–80% of the work done, and implementation and testing take up a major portion of the software life cycle.

Often students are unfamiliar with the notation used to represent flowcharts, data flow diagrams, program constructs, or the documentation involved with the use of special techniques. Exposure and use of specialized notations would not constitute a separate course, but could be integrated into many different courses.

Note should be taken of item number 6 on the list. All of the companies interviewed for the study used in this paper have COBOL as a primary language. Many universities do not teach COBOL using non-trivial applications because of its well-known deficiencies: lack of structure and "wordiness." The decision on how much COBOL to teach should not be based on its evaluation as a language in comparison with some of the more modern languages. Most of the companies agreed that COBOL was not the ideal language, but its use has been established in these organizations and it is economically not feasible to change. Also, most companies agreed that entry-level personnel did not have enough experience with actual business applications. Therefore, it is suggested that non-trivial business applications be programmed using COBOL at some time in a student's academic career.

If a computer science-applications development major takes suggested courses from all three areas outlined, he will be ready to begin his training in industry. He will be better prepared
for what industry needs, and will hopefully also have fresh approaches to problems that he can give to industry.

2. Model Curriculum. The model curriculum developed in the following sections is based on the current computer science major curriculum at Ball State University (BSU), in Muncie, Indiana. It will also include specifics of courses in other departments that would complement the CS major. First, an overview will be given of Ball State's current program. The new curriculum will then be presented, and the section summarized by the statement of problems in initiating such a curriculum.

The computer science program at BSU stresses the software side of data processing. Three prime areas of specialization are suggested. They are:

2. Operating Systems  Computer operating systems, compilers.

Data Systems specialists are expected to have a major in computer science and a minor in business. It is on this area of specialization that the model curriculum will concentrate.

The current CS major at BSU requires 83-84 quarter hours of math and computer science courses, with 68 hours of required studies and 15-16 hours of CS and math electives. The suggested business minor consists of 29 quarter hours split between accounting, economics and management.
All CS majors enter the program by taking CS220, an introductory programming course using FORTRAN. The student may then take any course from CS442, CS462, CS330, CS324, or CS466. The required courses are CS330, a course in machine language programming, and CS324, a course in applied combinatorics. CS330 serves as a pre-requisite for two other required courses, CS331 and CS332. CS331 is an extension of machine language programming skills, and CS332 is a data structures course utilizing PL/I for implementation. CS331 serves as pre-requisite for CS476, CS472, CS433, and CS436, with the latter three being required courses. CS472 is a mini-computer class with a "hands-on" laboratory; CS433 deals with basic hardware concepts; CS436 is a basic database course which also has CS332 as a pre-requisite. CS332 is also the pre-requisite for CS440, which is a course on data processing techniques utilizing COBOL. CS433 is a pre-requisite for the Systems Analysis (CS441) and computer software laboratory (CS498) courses. These last two classes are grouped as a software laboratory lasting two quarters.

The new curriculum would not change the structure of the major, but would change the content of some of the courses offered. In the first introductory programming class, Pascal would replace FORTRAN as the programming language. Although Pascal is not widely accepted in industry, its features steer students toward structured programming more readily than does FORTRAN. Complete documentation of all programs should be stressed in this class as it sets the programming habits for the rest of the major.
CS330, the next required course, would remain as is. The basic knowledge in assembly language that it provides is essential to more efficient programming in high-level languages. On the same level, CS324, applied combinatorics would stress symbolic logic along with its already established other functions.

The second assembly language course, CS331, would cover debugging with the use of core dumps more extensively. Other debugging techniques could be used and discussed. One of the most important goals of this course would be to make the student proficient in a command language, in Ball State's case, IBM JCL. One of the most repeated complaints heard from companies involved in the study was recent graduates' inexperience with a command language.

Minicomputer Systems (CS472) would remain intact, as would the Computer Hardware Systems (CS433) course. In CS436, the database course, the curriculum would suggest the investigation of modern Data Base Management Systems and query languages, and also the introduction of fourth-generation languages.

CS435, Computer Programming Languages, would be revised to include FORTRAN along with other languages offered. The concept of performance measurement of computer programs could be introduced to be used in the comparison of the many different languages.

The major COBOL course at BSU, CS440, should be split into two separate courses. The first course would do much the same as the present course does. The second course would be a laboratory where a business application would be developed by students
grouped into teams. The application would be of a size that would make development possible in the first five weeks of the 10 week quarter. During the second five weeks, each team would modify another team's project according to specific criteria supplied by the instructor.

The two quarter Systems Analysis and Software Laboratory would do much the same activities, but would introduce the concepts of software engineering more strongly and give the students specific experience with the more modern techniques. In the first course, an introduction of software engineering techniques and concepts would be given. The introduction would include explanation of the software life cycle; each phase and its input-process-output would be explained. Introduction of some of the established techniques would be given, plus details of their technical aspects. Exposure to some of the techniques still in development would also be given. The case study would then commence, with the output at the end of the course being a set of structured systems specifications. Each group's specifications would be subject to peer and instructor review for approval.

The second course of the laboratory would include the design and implementation of the system specified. After completion of these two phases, students of the class would evaluate the system and documentation acting as an organization wishing to purchase the system. This would give the students exposure to the analysis required for purchasing software systems.

Along with electives already offered by BSU, courses on Decision Support Systems, information resource management, and
information systems planning should be offered to especially supplement the applications development specialist.

The current suggested business minor for CS majors at BSU includes one year of general accounting, the elements of micro- and macro-economics, the principles of management, production and operations management, and office management and control. To supplement this minor, it is suggested that CS-applications development majors take Principles of Finance (FIN350) which is "designed to study different areas of financial decision making in a business firm."[BALL82] One of the areas studied is investment decisions under uncertainty, featuring economic analyses. Another important business course would be Organizational Behavior (MGT385). This course concentrates on group behavior, motivation, and leadership.

Other courses that would be added as suggestions are Business and Professional Speech (SPCH450) which emphasizes small audience speaking with discussion handling, and Technical Writing and Editing (JOURN418), which teaches writing clearly using technical terminology, and writing papers for lay people from highly technical documents.

3. Problems. There are many problems associated with implementing a curriculum for computer science with a strong software engineering basis in a university. Some are listed below:

1. Short Time commitment of Part-Time people--Following the software life cycle through with non-trivial examples requires much time and this can cause scheduling problems in some environments.
2. Many students on same problem--this provides opportunities for cheating and extreme machine congestion.

3. Absence of end-user--students cannot see the final implementation and use of their systems, or perform modifications when something is not right.

4. Lack of efficient accountability--student and machine time used cannot be accurately measured, leading to programs that may be inefficient.

5. Tensions of the academic environment--where to strike the balance between teaching research material or practical applications.[SHAW76]

E. CONCLUSIONS

The following conclusions can be drawn from this research:

1. Software engineering techniques are necessary for the development and operation of software systems.

2. Each organization must pick the proper techniques to use according to the organization's goals.

3. In the cross section of corporate environments studied, development in software engineering ranged from low to high.

4. Three major factors affecting software engineering development in the corporate environment are staff size, hardware size, and the percentage of applications software that has been purchased. The highest development companies had large staff size, large hardware configuration, and a low percentage of purchased applications software.

5. The corporate environments studied felt computer science graduates need more education in business, communications, and psychology. Also, more experience is needed in programming actual business applications in COBOL.

6. Software engineering education should be more interdisciplinary, with business, communications, and computer science courses.

This study is only a beginning. It is important that more studies are conducted in this same vein, so that the two-way communication can be opened between industry and academic
environments. Universities need to know how best to prepare their students for the jobs they will fill. Industry depends on academic environments for help in developing new and improved techniques and methodologies. It is hoped that this study will help in such communication.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALL81</td>
<td>Ball State University. &quot;Ball State University Computer Science Majors,&quot; (May 13, 1982), pg. 3.</td>
</tr>
<tr>
<td>BALL82</td>
<td>Ball State University. Ball State University Undergraduate Catalog, Vol. LVII, Number 1, (March, 1982), pg. 118.</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY (CONT.)


BIBLIOGRAPHY (CONT.)


APPENDIX A
FIGURE 1

- REQUIREMENTS ANALYSIS
- SYSTEM SPECIFICATION
- ARCHITECTURAL
- DETAILED
- CODING
- TESTING
- AUDIT
- PERFORMANCE MEASUREMENT
- MAINTENANCE
- MODIFICATION
APPENDIX B
1. **SET UP BOW**
   - Assembled Bow Section
   - Rubber Skin
   - Floor Boards

2. **ASSEMBLE HULL**
   - Hull
   - Rolled Up Stern Section

3. **SET UP STERN**
   - Stern Assembly

4. **BRACE HULL**
   - Braced Hull
   - Gunwales

5. **ASSEMBLE RUDDER GEAR**
   - Guy Lines
   - Rudder

6. **INFLATE**
   - Secure Hull

7. **FINISH**
   - Pedal Helm
   - Rudder Sub Assembly
   - Fully Assembled Kayak
Monitor Patients

1. Obtain a Patient's Factors
   4. Find next patient to monitor
      7. Convert patient no. _bed address
         9. Notify station of bad terminal
         10. Write line to station
   5. Read valid set of factors
   6. Store factors in data base

2. Find unsafe factors
   11. Obtain patient's safe ranges
   12. Determine if factor is unsafe

3. Notify stations of unsafe factors
   13. Format output lines

4. Read factors from terminal

FIGURE 4
5.2.1.1.1 VALIDATE RECEIPT ITEMS

5.2.1.1 COMPUTE ITEM EXPENDED PRICE

5.2.1.1.2 COMPUTE PAYABLE AMOUNT

5.2.1.1.3 COMPUTE DISCOUNTED PAYABLE AMOUNT

5.2.1.1.4 SUM GROSS PAYABLE AMOUNT

5.2.1.1.1.1 DISPLAY PURCHASE ORDER ITEMS

5.2.1.1.1.2 ACCEPT QUANTITY FROM TERMINAL

5.2.1.1.1.3 COMPARE QUANTITY TO ORDER BALANCE

FIGURE 5
TY PDL

1. PUSH "SCE" (START OF EXPRESSION) ONTO OPERATOR STACK
2. PROCESS OPERAND
3. DO WHILE NEXT TOKEN IS AN OPERATOR
4. IF OPERATOR IS NOT SAME AS OPERATOR ON TOP OF OPERATOR STACK
   AND ITS PRECEDENCE IS LESS THAN OR EQUAL TO PRECEDENCE OF
   OPERATOR ON THE TOP OF THE OPERATOR STACK
5. BUILD TOP NODE
6. POP OPERATOR STACK
7. ENDDO
8. IF NEW OPERATOR IS SAME AS TOP OPERATOR ON OPERATOR STACK
9. INCREMENT OPERAND COUNT IN TOP OF OPERATOR STACK BY ONE
10. ELSE
11. PUSH NEW OPERATOR AND OPERAND COUNT OF 2 ONTO OPERATOR STACK
12. ENDF
13. PROCESS OPERAND
14. ENDDO
15. DO WHILE TOP OF OPERATOR STACK IS NOT "SCE"
16. BUILD TOP NODE
17. POP OPERATOR STACK
18. ENDDO
19. POP OPERATOR STACK
20. (TOP OF OPERAND STACK CONTAINS TOP NODE IN EXPRESSION)

FIGURE 6
INITIALIZE AND OPEN FILES

WRITE REPORT HEADINGS TO PRINT FILE

READ PAYROLL FILE

DO WHILE NOT END OF FILE (ON PAYROLL FILE)

YTD PICA < MAXPICA

NO

YES

YTD PICA = YTD PICA + DEDUCTION

DEDUCTION = MAXPICA - YTD PICA

YTD PICA = YTD PICA + DEDUCTION

DEDUCTION = PICA x GROSSPAY

YTD PICA + DEDUCT. > MAXPICA

NO

YES

DEDUCTION = 0

NETPAY = GROSSPAY - DEDUCTION

CREATE OUTPUT REPORT RECORD

WRITE REPORT TO PRINT FILE

READ PAYROLL FILE

CLOSE FILES

FIGURE 7
CASE CONSTRUCT

REPEAT-UNTIL

DO-WHILE WITH BREAK
Late Payments Procedure (Narrative)

Begin

for each customer

Select overdue invoices and update overdue invoice file

Sort overdue invoices by customer, age (descending), amount (descending)

Sort customer groups by total dollars (descending)

Customer contacted

- or -

Customer not contacted

Agreed to pay

Did not agree to pay

Note time agreed upon for payment on follow-up report

Note reason and next review date on follow-up report

Note on follow-up report that customer could not be contacted

Return invoices to overdue file

FIGURE 9
INTERVIEW SHEET

GENERAL INFORMATION

Company name

Name and title of manager interviewed

Name of division (if applicable)

Date and time

DATA PROCESSING SECTION INFORMATION

Hardware currently in use

Primary languages

Estimated percentage of gross sales of revenues spent on software development and maintenance

Estimated percentage of applications software currently in use purchased from outside software houses

NOTES:

FIGURE 1 (1)
ORGANIZATION AND FUNCTION

Are analysis and programming handled by the same or different people?

Who initiates new development projects and how are they handled?

Who initiates modification requests and how are they handled?

What percentage of your work is modification and enhancement?

NOTES:

FIGURE 1 (2)
SOFTWARE LIFE CYCLE

What do you consider the phases of your software life cycle?

How is time divided among these phases?

Do you have a specific methodology concerning each phase of the software life cycle?

When was it introduced?

What is a rough estimate of the costs of initiating this methodology?
What features did it offer that were attractive?

Were any other methodologies studied before this one was selected?

How much orientation and training does its use require?

Does the methodology provide for project management, or is some other procedure used?

Has it saved the company money? How?
Has it improved the quality of software? How?

Is the company satisfied with this methodology?

Did you make any modification to the methodology after implementation?

Are you currently studying any other methodologies?

What are the attitudes of the employees towards the methodology?

Any other specific experiences concerning the methodology?
SOFTWARE CREATION TECHNIQUES

Does your company require the use of specific software engineering techniques such as structured analysis, PSL/PSA, HIPO charts, PDL, structured programming, or any other technique?

Do you have training classes in these techniques?

Do you have a set of standards to be followed in software creation (especially programming)?

Do you have a training program for employees that teaches the rules of structured programming?

Is there a stepwise programming procedure that is followed?
PERSONNEL

Do you feel that the majority of the recent college graduates that are hired at entry-level positions are lacking in any area of education?

Do you have any suggestions on how college curricula for computer science could be improved?
September 7, 1982

Dear (participant): 

Robin Ridgeway, a senior honors student at Ball State University in computer science, is beginning a comprehensive thesis on software engineering techniques to be completed in ten weeks. Robin has spent the last two academic years as an intern in Ball State's computer center and has also gained experience working at Marathon Oil Company as a summer intern in business programming.

The purpose of her study is to document some of the current software analysis, design, and implementation techniques available to data processing professionals today. Also planned is a survey of a cross section of corporate environments to determine which techniques are currently in use, why they were selected, and specific experiences with the techniques.

I am writing this letter to ask for your cooperation in this study. If your company decides to participate, an interview would be set up with someone in your data processing department. A copy of all questions to be covered would be sent to you prior to the interview. I feel your participation in this study would provide Robin with unique and interesting information about your organization, and also provide your company with information about some of the latest software engineering techniques. A copy of the report would be sent to your company upon completion. I assure you that the privacy of your organization would be maintained in every facet of this report.

The interview would be scheduled at your convenience between mid-September and mid-October. Robin will call you during the week of September 13th. If your company is interested in participating, she will set up an appointment at that time. If you would like more information, please feel free to contact me.

I appreciate your time and effort on Robin's behalf, and hope you will decide to be a part of this study.

Sincerely,

William F. Brown
STAFF SIZE

LARGE

MEDIUM

SMALL

STAGE OF DEVELOPMENT

FIGURE 1
HARDWARE SIZE

LARGE

MEDIUM

SMALL

LOW MEDIUM HIGH

STAGE OF DEVELOPMENT

FIGURE 2
PERCENTAGE OF SOFTWARE PURCHASED

STAGE OF DEVELOPMENT

FIGURE 3
APPENDIX E
FIGURE 1
CS220. **Scientific Computer Programming.** Computer organization and facilities. Basic computer programming skills using examples, exercises, and topics which emphasize applications in mathematics and science. Time-sharing techniques are introduced.

CS324. **Applied Combinatorics.** Techniques of network and combinatorial analysis with applications to algorithms and structures in computer and information science. Course assignments will include analysis, design, and implementation of various algorithms and structures.

CS330. **Machine Language and Systems Programming 1.** Introduction to computer architecture, machine language, and assembler language; fixed point and decimal arithmetic; character manipulation; subroutine linkage; looping and branching.

CS331. **Machine Language and Systems Programming 2.** Advanced topics in machine and assembler languages; bit manipulation, floating point arithmetic, macros, conditional assembly, core dumps and debugging, input/output device programming, interrupt programming.

CS332. **Advanced Programming Techniques.** Topics such as structures, blocks, dynamic storage allocation, list processing and pointer variables, compile time instructions, organization and manipulation of data files, and interrupt handling are treated using the PL/I language. Operating system features are included as required by these topics.

CS433. **Computer Hardware Systems.** Introduction to computer logic and register-transfer level of systems description is presented. The basic building blocks of a computer system are examined. Various computer architectures and hardware trends are studied. A general methodology for computer selection is developed.

CS435. **Computer Programming Languages.** Emphasis is on important structural characteristics of high level programming languages. Additional languages studied are APL, ALGOL-68, LISP, and SNOBOL to show such features as dynamic array manipulation, language extensibility, list and string processing. Some compiler techniques are also considered.
COURSE DESCRIPTIONS

CS436. Information Structures. Data Structures used in computer storage to represent the information involved in solving problems. Emphasis is placed on data structures independently of applications in which they are embedded. Storage structures are described according to their characteristics.

CS440. Data Processing Techniques. Commercial data processing techniques. Problems typical to DP centers such as file maintenance, report writing, sorting and merging are treated using COBOL. Tape oriented and direct access processing techniques.

CS441. Systems Analysis. The role of and techniques used by the Information processing specialist. Includes a case study laboratory.

CS472. Minicomputer Systems. The minicomputer considered from the viewpoints of its architecture, operating system, and application. Topics included are processor organization, peripheral devices, I/O programming, system programs, monitor services, and file organization. The departmental minicomputer, PDP-11, will be used for hands-on laboratory projects.

CS498. Computer Software Laboratory. A software system will be designed, developed, and implemented in an environment simulating a software development organization. The latest methods and discipline of software engineering will be used.