COMPUTER-AIDED SOFTWARE ENGINEERING'S IMPACT ON THE SOFTWARE DEVELOPMENT PROCESS: AN EXPERIMENT

Mary J. Granger

College of Business and Management
Morgan State University
Baltimore, Maryland 21239 USA

Roger Alan Pick

College of Business Administration
University of Cincinnati
Cincinnati, Ohio 45221-0130 USA

ABSTRACT

Organizations today invest enormous resources in Computer-Aided Software Engineering (CASE) technologies with the hope of gaining significant increases in programmer productivity. In an empirical study, eleven three-person teams, composed of undergraduate Information Systems majors, developed versions of the same software system, a Pascal Pretty Printer; four teams used the same CASE tool and seven teams did not use any CASE tools. The major results of this study are: 1) most of the CASE projects were coded in less time than the non-CASE projects; and 2) all of the CASE projects met more of the requirements than the non-CASE projects. Each of these results was a statistically significant effect.

INTRODUCTION

CASE (Computer-Aided Software Engineering) is perceived by many software developers as the answer to their software problems and the "software crisis" that has plagued the computer industry [8, 18, 21]. The term "software crisis" describes the current state of software development: systems that do not meet the client's specifications; systems that are over budget and/or late; systems that are extremely complex; and systems that are difficult to maintain. The increased demand for applications software has created an estimated backlog of four years [31].

One reason for the software crisis is that programmer productivity has only been growing at a rate of 5% per year. The last major breakthrough in programmer productivity was in the 1950s with the introduction of language compilers [12]. Programmer productivity has been a pressing issue for Management Information System managers for the last four or five years [15]. Techniques for improving programmer productivity include structured programming, structured design methods, requirements analysis techniques, and software engineering. CASE is considered a software development environment that supports these methods and processes. CASE, the automation of software development [19, p. 4], is considered a major step toward improving programmer productivity and therefore a partial solution to the software crisis.

Very little research about the effectiveness of CASE technology exists [7, 23]. Although recent studies [1, 10] report increased programmer productivity, few formal measurements exist. Other affirmatory reports such as [17] and [9] have come from the developers of commercial products. [19] reports programmer productivity gains at DuPont but does not indicate where or how these increases were developed. Another practitioner article [28] discusses increased programmer productivity after adoption of CASE technology, but these articles are anecdotal and lack quantitative productivity measures. Often productivity gains are reported using the actual implementation time with CASE and the estimated implementation time without CASE.

The only empirical study that we are aware of is one done by Norman and Nunamaker, which is reported in the Communications of the ACM [23], the Proceedings of the Ninth International Conference on Information Systems [22] and the Proceedings of the Twenty-second Annual Hawaii International Conference on Systems Science [24]. The study used a questionnaire to evaluate managers'

0073-1129/91/0000/0028$01.00 © 1991 IEEE

perceptions of the relative productivity improvement of a specific CASE tool's components over manual methods. They did not attempt to measure the degree of productivity improvement using CASE technology.

This research tests the belief that the use of CASE technologies increases the productivity of the system analyst/programmer. That is, an improved product is produced in the same amount of time or the same product is produced in less time.

The formal hypothesis used to test this belief is: there is no difference in the productivity of the system analyst/programmer who uses CASE technologies and the productivity of the system analyst/programmer who does not use CASE technologies.

THE EXPERIMENT

Excelerator is the CASE tool selected for this research. Excelerator was named Software Product of the Year in 1987 by the American Federation of Information Processing Societies [14] and also has been the most widely used CASE product [11].

This study is a controlled experiment using student subjects in a software engineering course implementing a classroom project. A classroom project is used for two reasons. First, controlled experiments in an organizational environment are too costly and time consuming [20]. Second, a better experimental design can be achieved in a classroom setting than in a commercial setting.

There are a number of benefits to conducting the study in a classroom setting. Commercial projects will not be replicated by software developers because of financial and practical considerations. Given two commercial projects, neither the system nor the programming teams are the same. Data collection is easier with student subjects; practitioners are reluctant to change their way of doing business [6]. This means it is difficult to isolate and evaluate the effect of the technology being studied [4, 13, 2].

In this experimental study, the same task was given to a number of teams of university students enrolled in two sections, across two quarters, of an Information Systems course titled Software Engineering. Students are a convenient sample and since these students are Information Systems majors, they are also a representative sample of future users of CASE technologies.

The main objective of the software engineering course is for the student to gain knowledge of structured methods and to become familiar with "programming in the large" by working in a programming team environment and implementing a small (600-2000) line system. Although 600-2000 lines is not "programming in the large," this is an appropriate size for the limited amount of time in a ten-week quarter. The students used structured methodologies to implement and complete the assigned project.

THE TASK

The experiment was conducted over two quarters using the same task each quarter. The project consisted of designing, coding, testing, debugging, and documenting a Pretty Printer for Pascal programs. The project was of moderate difficulty and length; it was a non-trivial problem, resulting in an average of 1500-2000 lines of Pascal source code.

As defined for the students' project, a Pretty Printer was a computer program that reformatted computer programs [5, 25, 30]. The new reformatted version of a computer program should be easier to understand and read.

The original version of the computer program was stored in a text file on a Digital Equipment Corporation VAX 6350 under the VMS operating system and was treated as input in the form of character strings. It was not the purpose nor the responsibility of the Pretty Printer to detect syntax errors. The Pretty Printer assumed that the input was a text file containing a syntactically correct Pascal program. The output also was to be a syntactically correct program with the same execution behavior as the input. The formatting activities the Pretty Printer should include were adding or modifying appropriate line breaks, spacings, and indentations; alphabetizing variable names and constant names within their respective sections; and capitalizing reserved words. The final modified Pascal program was either stored in another file, displayed on the screen, or printed. Any combination of these three options could be specified by the user of the Pretty Printer.

The instructor manually prepared identical detailed requirements and gave them to the students at the beginning of

each quarter. The requirements were not prepared using CASE technologies; use of CASE might have provided an advantage to the subjects using CASE and have a negative impact on the non-CASE subjects. All students had available the same computer resources, and the same programming implementation language, the same debugging tools. They were constrained by the ten-week quarter time period.

The task was divided into two major phases, design and coding, with the emphasis on the design phase (first five weeks). Students in the Spring 1989 quarter (control group) did not use CASE technologies while they learned structured methodologies; those in the Autumn 1989 quarter (treatment group) used CASE technologies.

THE SUBJECTS

Participants in the study were junior-level Information Systems majors enrolled in the Software Engineering course in the day program of the College of Business Administration of the University of Cincinnati. All students enrolled in the two sections participated. In order to enroll in the course the students must have completed all their freshmen-level and sophomore-level Information Systems courses: Introduction to Data Processing; Principles of Structured Programming; COBOL I; COBOL II; and Data and File Structures. The structured programming concepts introduced in these previous Information Systems courses are formalized and expanded within the scope of a larger project in the Software Engineering course.

Almost all of the students were familiar with the computer system and the implementation language. Most previous Information systems programming courses used the VMS operating system and two of the prerequisite courses, Introduction to Data Processing and Data and File Structures, required the use of Pascal. The students were not familiar with a team concept of programming nor formal structured software development methodologies. Teams worked independently in both quarters, with no collaboration among teams, during all phases of project.

During the Spring 1989 quarter, the student teams implemented a Pretty Printer using structured methods, including top-down design and structured programming [32, 26]. They did not use any of the integrated CASE technologies. All of the teams decided, independent of class instruction, to learn and use FLOW [27] for their structure charts and data flow diagrams. FLOW is a graphics word processor. All the checking and verifying within the data dictionaries, structure charts, and data flow diagrams was done "manually" (without any computer aided integration). These seven three-person teams were the control group or benchmark for the experiment.

During the following Autumn 1989 quarter, the same task was implemented. Student teams received the same instruction and used the same structured methodologies as in the Spring 1989 quarter but also were required to use the available integrated CASE technologies during the design phase.

TEAM COMPOSITION

In a previous study using student teams, Rombach [29] ranked students on their educational performance (grades), experience (industry), and relative programming talent. In the present study, educational performance and programming talents were combined as the students' ability and were used in conjunction with work experience in determining team composition. Each team had a "more-experienced" member, a "less-experienced" member, and an "inexperienced" member. "More-experienced" was defined as either several quarters of co-op work experience or more than a year of part-time work and familiarity with several operating systems. "Less-experienced" students had one or two quarters of co-op experience or less than one year of part-time work and familiarity with one or two operating systems. The students classified as "inexperienced" had little or no practical work experience; most of their knowledge about the field had been acquired from their courses. Demographic data and information about the level of experience were collected using a pretest questionnaire. The level of experience was evaluated by the instructor, a graduate student, and a Senior Information Systems major (grader). There was no significant difference in either the age or grades between the control group and treatment group. The two groups were similar in course background and work experience. Data regarding students' ability was also gathered from several of the students' previous instructors and the grades (Basili 1981) earned in their prerequisite IS courses. As long as the

teams maintained this mix of experience and ability, some consideration was given to students' preferences in regard to team members.

In the Spring 1989 quarter there were seven three-person teams, and in the Autumn quarter there were three three-person teams and one four-person team. The four-person team had one "more- experienced" member, one "less-experienced" member and two inexperienced" members. This team was formed originally as a three-person team, but two students from another team dropped the course and the remaining student had to be placed on a team.

THE CLASSROOM PROCESS

Students were given the specifications for the Pretty Printer during the first week of class. Those in the treatment group (those that were required to use Excelerator) were told that they should start learning how to use that software. Those in the control group (without Excelerator) were told that they could use any software that was available; there were no restrictions. However, neither Excelerator nor other integrated CASE products were available. Class lectures for the first four weeks of the quarter covered structure charts, data flow diagrams, data dictionaries, module specifications, and interface specifications. Time for questions was allotted at the beginning of each class period; there were two 75-minute class meetings per week.

During the Autumn quarter, any questions on Excelerator were answered in class. However, except for distributing passwords and project designations, the only class time spent on Excelerator was that time used to answer students' questions. Designs were collected, graded, and returned during the fifth week. Included in the design package were the structure charts, data flow diagrams, data dictionaries, module specifications, interface specifications, and a brief description of the team's design with an explanation of a Pretty Printer. The coding phase then began. Class lectures concentrated on modularity, cohesion, coupling, fan-in, fan-out, and external procedures in Pascal on the VAX. There was some discussion of external Pascal procedures on the VMS operating system and VMS command files.

The project was handed in during the tenth week of the quarter. Required in this phase were the Pascal code, a user's manual, a programmer's manual, and any changes made to the original design with an explanation as to why the changes were necessary.

A history effect on the autumn quarter from the spring quarter probably did not occur. Any communication across quarters probably did not have a significant impact on the autumn projects. Also, students tend not to plagiarize on large scale projects. Also, students tend not to plagiarize on large scale projects. It has been observed by the faculty that, while students may give students in a following quarter a small, several hundred line program, they are not willing to do the same with a full-quarter project. The instructor had 10 years experience teaching Information Systems courses and one year experience teaching Software Engineering. Any improvement in instructor performance across quarters was likely to be marginal.

DATA COLLECTION

The development process was evaluated using the data collected during the design phase and information collected automatically during the implementation phase. The product was evaluated using a number of software metrics. Data involving the design phase was collected using weekly progress reports, minutes of team meetings, and personal logs. The student logs contained a record of all the activities associated with the Software Engineering course and the amount of time spent on each activity. This information was recorded daily or whenever the students worked on the project.

Students were told that their logs should be complete so that if someone would replace them on the project, their logs could serve as an introduction and a clarification of the work already in progress. The log would explain both how and why things were being done.

In order to prevent the logs being written at the end of the quarter and consequently presenting an inaccurate picture of the individual processes, the logs were date-stamped weekly. The students' logs for both quarters were examined and the reported individual times and group times spent for each phase of the project were tallied.

Data from the coding phase were automatically collected. Several programs had been created that collect this data without the students' interaction. The students knew that

data about their project was being collected, but they did not know how or why. Measures were taken to insure confidentiality; after the data was collected and the grades for the quarter assigned, the data was not associated with identifiable individuals. Each logon, compilation, link, run, and logoff was recorded. The total amount of time on the system and the counts for the compilations, links, and runs were used to determine whether use of CASE technology reduced the number of iterations necessary to implement the system. This data was collected for each individual student and combined to form the group totals. Due to individual debugging and coding styles, some individual differences were introduced during this phase of the project. A senior in Information Systems evaluated the projects for completeness. He was given all eleven projects to evaluate, but he did not know which projects were developed using CASE and which were developed without CASE. The seven projects developed without CASE were randomly merged with the four that used CASE. The instructor kept a key to the projects.

The original twenty-four specifications were used to determine the completeness of each project. On each of the different specifications, a 0 - 1 - 2 scale was used to record the completeness. A score of '0' indicated that no attempt was made to accomplish that specification. A score of '1' indicated that some attempt was made, but it was not a totally successful attempt. A score of '2' indicated a totally successful attempt at a particular specification. Once the scales were returned to the instructor, they were connected with their appropriate group.

RESULTS

Discriminant analysis (SPSSX) used eight time variables and eight size variables for each team. The time variables were: the total time reported by the teams; the reported time spent on design; the reported time spent on coding and debugging; the number of team compiles; the number of team links; the number of team runs; the number of team logons; and the total team time spent on the VMS system. The size variables were: the number of lines of code; the number of comments; the total number of functions and procedures; Halstead's length; Halstead's estimated length; Halstead's implementation level; Halstead's volume; and Halstead's vocabulary [33]. T-tests tested for significant differences in the means (control group versus treatment group) for each variable. Table 1 and Table 2 show the levels of significance (P values) for each of the variables for the time and size categories respectively.

The P values obtained from the t-tests indicate there is a significant difference for some of the variables in the time category. The number of runs and links are significant at a 0.05 level; the amount of time spent logged on to the VMS operating system is significant at the 0.05 level; and the number of compiles is significant at the 0.10 level. The other four variables are very close to the 0.10 level of significance. In view of the low power of the t-test for such a small sample size, these values should also be considered to have large practical significance.

TABLE 1

LEVEL OF SIGNIFICANCE (P VALUES) AND MEANS
TIME VARIABLES (ALL TEAMS)

NAME	P VALUES	MEANS CONTROL GROUP	MEANS TREATMENT GROUP
TOTDES	0.1473	81.71	50.00
TOTCOD	0.1552	122.71	23.50
TOTTME	0.1091	204.43	73.50
GCOMPL	0.0589	3912.71	1652.50
GLINKS	0.0190	1057.86	559.75
GRUNS	0.0174	1023.14	536.25
GTIME	0.0326	210.86	164.00
GLOGON	0.1164	220.29	165.00

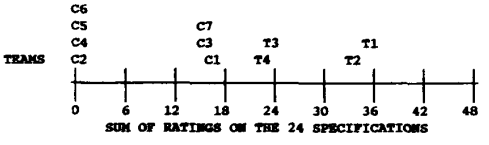
TABLE 2

LEVEL OF SIGNIFICANCE (P VALUES) AND MEANS
SIZE VARIABLES (ALL TEAMS)

NAME	P VALUES	MEANS CONTROL GROUP	MEANS TREATMENT GROUP
LOC	0.3296	1751.00	1437.50
CMMNTS	0.7259	188.15	221.00
MODULES	0.2421	34.00	25.75
LENGTHN	0.4608	4655.71	3694.75
ESTN	0.7508	1567.71	1398.25
IMPLEVEL	0.1878	0.006	0.008
VOLUME	0.5016	36553.71	28439.25
VOCAB	0.7767	215.43	198.75

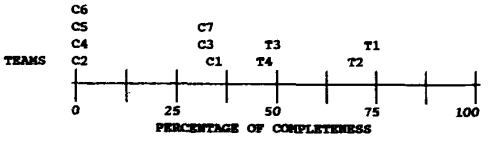
None of the size variables are significant. This should not be too surprising since all the projects were designed and coded to meet the same requirements.

A totally complete system, one that completely met all the requirements for all the Pretty Printer functions would have a rating of 48. Figure 1 indicates the level of completeness for the eleven projects. Figure 2 shows the percentage of completeness. The four projects, all from the control group, that received a '0' rating for all 24 specifications had run time errors caused by either a stack dump error or an access violation error; none of the projects had compile errors. The three remaining projects from the control group were each approximately 33% complete. The four treatment group projects ranged from 44% to 75% complete. Using the mean ratings for both groups, all teams, the p value was 0.002. Omitting those teams that had run time errors, the p value was 0.045. Both values were significant at the 0.05 level.



SUM OF RATINGS ON THE 24 SPECIFICATIONS
('T' indicates the treatment group (CASE) and 'C'indicates the control group (non-CASE). The numbers following the 'T' or 'C' indicate the team within the group.)

FIGURE 1
LEVEL OF COMPLETENESS



PERCENTAGE OF 24 SPECIFICATIONS COMPLETE WITH A TOTAL OF 48 BEING PERFECTLY COMPLETE
('T' indicates the treatment group (CASE) and 'C'indicates the control group (non-CASE). The numbers following the 'T' or 'C' indicate the team within the group.)

FIGURE 2
PERCENTAGE OF COMPLETENESS

CONCLUSION

There was a significant difference in the time required to code the system. The treatment group, CASE, used less time than the control group, non-CASE. The size of the products were not significantly different; however, the level of completeness of the systems was significantly different. The CASE group developed systems that were more complete than systems developed by the non-CASE group.

If productivity is defined as producing the same product in a lesser amount of time, it can be concluded that the productivity during the coding phase was increased by the use of CASE technologies during the design phase. Since the CASE group also produced a more complete project, we can conclude that productivity was also increased because a 'better' product was developed in less time. The CASE-developed systems were better able to meet the specifications in less time than the non-CASE developed systems. The hypothesis that there is no difference in the productivity of the system analyst/programmer who uses CASE technologies and the productivity of the system analyst/programmer who does not use CASE technologies is proven false; there is a difference in the productivity. Therefore, our original belief is supported.

SUMMARY/IMPLICATIONS

Information Technology managers should be encouraged in their quest for increased programmer productivity. Most of the students majoring in Information Systems will be the applications systems analysts of tomorrow; therefore, the results may generalize to the entire population of professional system analysts.

It should be noted that the students in the Autumn quarter were novices in CASE and in software engineering; novices are not always as successful as seasoned users [3]. Although the subjects were novices with CASE technologies, they had a significant improvement in productivity. However, the sample size was small and the project was not a true "programming in the large" project.

Only one CASE tool, Excelerator, was used and the authors would like to repeat the experiment using different CASE products. The results from only one task are reported and the authors

are currently collecting data for another project. Future research and analysis is planned to include the data from additional projects and tools.

BIBLIOGRAPHY

[1] Acly, Ed (1988) "Looking Beyond CASE." IEEE SOFTWARE, March 1988, 39-43.

[2] Attewell, Paul and Rule, James (1984) "Computing and Organizations: What We Know and What We don't Know," COMMUNICATIONS OF THE ACM, Vol 27, No. 32, December 1984, 1184- 1192.

[3] Basili, Victor R. and Reiter, Robert W. (1981) "A Controlled Experiment Quantitatively Comparing Software Development Approaches." IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE- 7, No.3, May 1981, 299-320.

[4] Boehm, Barry W., (1981) SOFTWARE ENGINEERING ECONOMICS, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[5] Cameron, Robert D. (1988) "An Abstract Pretty Printer." IEEE SOFTWARE, November 1988, 61-67.

[6] Card, David (1988) "Major Obstacles Hinder Successful Measurement." IEEE SOFTWARE, November 1988, 82,86.

[7] Carey, Jane M. and McLeod, Raymond (1988) "Use of System Development Methodology and Tools." JOURNAL OF SYSTEMS MANAGEMENT, March 1988, Vol. 39, No. 3, 30-35.

[8] Chikofsky, Elliot J. (1988) "Software Technology People Can Really Use." IEEE SOFTWARE, March 1988, 8-10.

[9] Chikofsky, Elliot J. (1989) "Making CASE Pay Off." CIO, February 1989, Vol. 2, No. 5, 12-16.

[10] de la Torre, Jose. (1988) "Quality-assured Software in 4GL/CASE." BUSINESS SOFTWARE REVIEW, March 1988, Vol. 7, No. 3, 30-33.

[11] Fersko-Weiss, Henry (1990) "CASE Tools for Designing Your Applications," PC MAGAZINE, Vol. 9, No. 2, January 30, 1990, 213-251.

[12] Frenkel, Karen A. (1985) "Toward Automating the Software- Development Cycle. "COMMUNICATIONS OF THE ACM, Vol. 28, No. 6, June 1985, 578-589.

[13] Glass, R. L. (1982) "Modern Programming Practices: A Report from Industry", Prentice-Hall, Englewood Cliffs, New Jersey, 1982 as cited in Abdel-Hamid, Tarek K. (1988) "Understanding the "90% Syndrome" in Software Project Management: A Simulation-Based CaseStudy", THE JOURNAL OF SYSTEMS AND SOFTWARE, August 1988, 319- 330.

[14] Hanna, Mary Alice (1990) "Move Is On To Tie Vision To Information Systems," SOFTWARE MAGAZINE, Vol. 10, No. 1, January 1990, 39-45.

[15] Hartog, Curt and Herbert, Martin (1986) "1985 Opinion Survey of MIS

Managers: Key Issues," MIS QUARTERLY, December 1986, p. 351-361.

[16] Humphrey, Watts S. (1989) MANAGING THE SOFTWARE PROCESS. Addison-Wesley Publishing Company, Reading, Massachusetts.

[17] Martin, Charles F. (1988a) "Getting CASE in Place." BUSINESS SOFTWARE REVIEW, Vol. 7, No. 5, April 1988, 20-25.

[18] Martin, Charles F. (1988b) "Second-Generation CASE Tools: A Challenge to Vendors." IEEE SOFTWARE, March 1988, 46-49.

[19] McClure, Carma (1989) CASE IS SOFTWARE AUTOMATION. Prentice-Hall, New Jersey.

[20] Myers, Glenford J. (1978) "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections", COMMUNICATIONS OF THE ACM, Vol 21, No. 9, September 1978, 760-768 as cited in Abdel- Hamid, Tarek K. (1988) "Understanding the "90% Syndrome" in Software Project Management: A Simulation-Based Case Study", THE JOURNAL OF SYSTEMS AND SOFTWARE, August 1988, 319-330.

[21] Nejmeh, Brian A. (1988) "Designs on Case." UNIX REVIEW, Vol. 6, No. 11, November 1988, 45-50.

[22] Norman, Ronald, J. and Nunamaker, Jay F. (1988) "An Empirical Study of Information Systems Professionals' Productivity Perceptions of CASE Technology." PROCEEDINGS OF THE NINTH INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS, Minneapolis, Minnesota, November 30-December 3, 1988, eds. DeGross, Janice I. and Olson, Margrethe H., 111-118.

[23] Norman, Ronald, J. and Nunamaker, Jay F. (1989a) "CASE Productivity Perceptions of Software Engineering Professionals." COMMUNICATIONS OF THE ACM, Vol. 32, no. 9, September 1989, 1102-1108.

[24] Norman, Ronald J. and Nunamaker, Jay F. (1989b) "Integrated Development Environment: Technological and Behavioral Productivity Perceptions," THE 22ND HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, Vol. II, ed. Shriver, Bruce D., January 3-6, 1989, 996-1003.

[25] Oppen, Derek C. (1980) "Prettyprinting." ACM TRANSACTIONS ON PROGRAMMING LANGUAGES AND SYSTEMS, Vol. 2, No. 4, October 1980, 465-483.

[26] Page-Jones, Meilir (1988) THE PRACTICAL GUIDE TO STRUCTURED SYSTEMS DESIGN. Yourdon Press, Englewood Cliffs, New Jersey.

[27] Patton (1986). FLOW CHARTING II+ [Computer Program]. (Version 2.40B). San Jose, California: Patton & Patton Software Corp.

[28] Rochester, Jack B. (1989) "Building More Flexible Systems." I/S ANALYZER, Vol. 27, No. 10, October 1989, 1-12.

[29] Rombach, H. Dieter (1987) "A Controlled Experiment on the Impact of Software Structure on Maintainability." IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE-13, No. 3, March 1987, 344-354.

[30] Rubin, Lisa F. (1983) "Syntax-Directed Pretty Printing - A First Step Towards a Syntax-Directed Editor." IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE-9, No. 2, March 1983, 119-127.

[31] Shemer, Itzhak (1987) "Systems Analysis: A Systematic Analysis of a Conceptual Model," COMMUNICATIONS OF THE ACM, Vol. 30, No. 6, June 1987, 506-512.

[32] Yourdon, Edward N. and Constantine, Larry L. (1979) STRUCTURED DESIGN. Prentice-Hall, New Jersey.

[33] Halstead, Maurice H. (1977) ELEMENTS OF SOFTWARE SCIENCE. Elsevier, New York.