Software Engineering Education: A Survey of Current Courses

Laurie Honour Werth
University of Texas at Austin
Austin, Texas 78712

Abstract

This study summarizes the software engineering course offerings of nearly one hundred universities responding to a recent survey. Information to determine the current state of software engineeering education is tabulated and analyzed in order to support curriculum development and to determine deficiencies and future needs.

Characteristics of the institutions are used to compare relative numbers, types and academic levels of course offerings. A history of course startups provides insight into the growth and direction of new classes. Instructor background is examined to determine current and future faculty support needs. Similarly, the use and availability of textbooks, automated tools, and other teaching materials are investigated for course support requirements. Information on course format and project organization provides additional understanding of the structure of current software engineering courses.

Summary

Sixty-nine percent of the ninety-five schools responding to this survey offer one or more software engineering courses. One third of the remaining schools integrate software engineering into other courses. Three quarters of the courses are general in nature, with at least one half of them offered at the undergraduate level. While one third of the respondents offer more than one course, only ten percent are able to offer courses specialized to one phase or aspect of software development.

Eighty percent of the larger institutions offer software engineering, but only half of the smaller schools have courses. Similarly, institutions offering the PhD are far more likely to offer software engineering than other schools. At least one third of the undergraduate courses are required.

Only twenty percent indicate neither industry experience nor formal training in software engineering as instructor background. Multiple text and reprint titles demonstrate a need for additional teaching materials. Speaker videotapes, case studies, and automated tools would be other helpful improvements.

I. INTRODUCTION

The field of Software Engineering has grown rapidly since the term was first introduced in the late sixties. Several attempts have been made to assess the current state of software engineering education efforts [1,2]. The current survey builds on these earlier results to determine the current state of university level software engineering course offerings, to find out what progress is being made, and to help in planning for future needs. Because of the diverse nature of software engineering and its rapid growth, it is vital for instructors, curriculum developers, textbook writers, and others to keep abreast of educational developments. The speed of the replies, together with the strong interest reflected in the comments from the schools involved in the study, demonstrates the importance which the responding departments place on learning more about software engineering education efforts.

II. THE SURVEY

Software Engineering has become an accepted part of many university computer science department offerings, but little is known about the status of these newly developed courses. To gain insight into the nature of these courses, a survey was mailed recently to about

200 universities which were considered to be likely candidates to be offering software engineering classes.

The survey questions covered institutional characteristics, types of computer programs offered by the respondents, software engineering courses offerings, and efforts to integrate software engineering into other computer science courses. Software engineering course information included course title, date first offered, number of students, course level and prerequisites, instructor background, textbook and other teaching materials used, project activities, organization, and phases completed, and use of automated tools. The length and complexity of the survey undoubtedly reduced the number of responses, but this depth of coverage was considered essential to gain an adequate understanding of the organization of the courses and how they fit into the overall offerings. Ninety-five responses have been received to date. Not all items were completed by each respondent, so some partial results are reported.

The Institutions

Information about the kinds of programs offered by the responding institutions is summarized in Table I. Sixty-nine percent of the schools offer one or more software engineering courses. One third of the schools which do not offer separate courses, integrate software engineering topics into other courses.

Nearly half of the schools offering software engineering courses have enrollments of more than 10,000 fulltime students. Only twenty percent of these large schools do not offer such a course, while approximately half of the smaller schools do not offer software engineering. Similarly, while the respondents were fairly equally divided between being primarily undergraduate, masters degree and doctorate granting institutions, eighty-five percent of the Ph.D. granting institutions, two thirds of the masters granting institutions and one half of the primarily undergraduate institutions offer software engineering.

The Courses

Information on current software engineering course offerings is summarized in Tables IIa and IIb, with courses listed in order by the year in which they were first offered. Courses are separated into general courses, which cover most or all of the phases of the software life cycle, and specialized courses, which concentrate on one phase such as testing or project mangagement. The number of students is calculated by multiplying the number of offerings per year times the average number of students per year (not including classes which are part of a sequence). Within years, courses are separated into undergraduate and graduate level courses, if necessary.

Table I. The Institutions
(in percentages)

Fulltime Enrollment		Highest Degree Offered		Name of College		Name of Department	
< 2,000	7	Undergrad	30	Engineering	23	Computer Sci	60
2,000- 5,000	21	Masters	26	Arts & Sci	23	Math & CS	5
5,000-10,000	20	Doctoral	39	Science	26	Mathematics	17
>10,000	49	Unspecified	5	Liberal Arts	20	Engineer & CS	6
Unspecified	4			Math & CS	4	Other	5
				Unspecified	5	Unspecified	7

Highest CS Degree Offered		Computer Engineering Programs Offered		Information Systems Programs Offered	
Bachelor	33	At Institution	38	At Institution	48
Master	37	In Department	17	In Department	15
PhD	28	Unspecified	1	Unspecified	5
Unspecified	2				

Sixty-six general software engineering courses and three systems analysis courses are described in Table IIa. At least ten of these represent full year sequences. Incomplete information was also collected on another twenty-eight courses. Two thirds of the courses that specified level are undergraduate courses and one third are graduate courses. Twenty-three courses in specialized software engineering topics are offered as shown in Table IIb. One fourth of these are undergraduate, three fourths are graduate, and two do not have a level specified.

Two thirds of the schools offer only one course, and it is generally called Software Engineering. The next most popular names include Software Design and Development/ Implementation, Software Development Methods/ Techniques, or Large Scale Software Development. Project Management, Topics in Software Engineering, Software Engineering Economics, or Testing and Quality Assurance are the specialized courses most often offered. Ten percent of the schools which had software engineering courses, offered all the specialized courses listed in Table IIb.

Thirty-five percent of the respondents offer more than one software engineering course, so multiple offerings do not necessarily imply specialized courses. Only twenty percent of those offering more than one course report a full year sequence. Another twenty percent combine one or more graduate courses with an undergraduate course. The rest represent other combinations.

The Prerequisites

Data structures is the prerequisite mentioned by about three quarters of the schools which included this information. Other schools require data structures together with some combination of programming languages, operating systems, and/or file systems. A few schools limit enrollments to seniors or those who have met the bachelor's degree requirements. Most of the specialized courses required an undergraduate software engineering course.

Table IIa. Software Engineering Course History
General SE and Systems Analysis

Year First Offered	Num. of Courses	Students Per Year	Number Required	Course Level
1960	1	60	0	U
1972	2	180	1	U
1973	1	70	1	U
1975	2	105	-	4M
1977	1	70	1	4M
1978	1	120	1	U
	2	15+	0	4M,MP
1979	1	70	1	U
	1	30	1	M
<1980	5	240	2	U
	1	10	-	M
1980	5	305	1	U
	3	105	1	4M
1981	3	125	0	4,2U
	4	75	2	4M,MP
1982	5	250+	2	U
1983	7	100	2,Y/N	U
	2	130	1,Y/N	4M
1984	3	290	2	U
	2	130	1	M
1985	2	23	1	U
	1	8	-	M
1986	3	44	1	U
	3	105+	0	4M
1987	2	60+	1	U
	1	15	-	M
no year	2	-	-	4
given	3	90	1	M

44% Undergrad 24% Graduate 31% Unspecified
3% Sys Analy. 66% General 30% Unspecified

Table IIb. Software Engineering Course History
Specialized Courses

1980	2	170	-	U
1981	1	10	0	MP
1982	1	20	-	MP
1983	4	67	-	2M,2MP
1984	1	20	0	U
	3	60	1	2M,MP
1985	3	23	-	M,2MP
1986	2	42	-, Y/N	M
1987	1	-	0	4
no year	5	-	-	1M,4MP

21% Undergraduate 70% Graduate 9% Unspecified

Y/N = Required for some +/- = Unspecified
U = Undergraduate (Upper Division)
4 = Seniors 4M = Seniors + Masters
M = Masters MP = Masters + PhD

Course Materials

Information on the textbook(s) and teaching materials was requested as this is often cited as a problem area [2]. The wide range of texts in use is compiled in Appendix I. The two texts most often mentioned were Fairley [12] and Pressman [28]. Other texts freqently cited include Brooks [5], DeMarco [9], Sommerville [33], and Zelkowitz, Shaw and Gannon [41].

Several of the IEEE tutorials are also mentioned, especially for the advanced, specialized courses. A few instructors include various IEEE and DOD standards descriptions in their lists. Many cite more than one text, but do not state the reason. They may not be satisfied with the coverage in a single text, or the text may vary by instructor, or perhaps some of the additional texts are supplements. Most provide copies of their slides, notes, or other handouts, as class material.

Many instructors indicate the use of reprints, ranging from "a few" to as many as thirty. Several respondents included their reprint titles or complete reading lists. These titles were combined and are included as Appendix II. A general collection of papers seems to be needed, as an IEEE tutorial for example, to reduce the time requirements for faculty and students to acquire an up-to-date set of software engineering reprints. The current tutorials are excellent, but they are often too specialized for the typical general introductory course. Two instructors use the historical papers included in [40].

The Instructors

Finding qualified instructors is another problem area in software engineering education [2]. Interestingly, only four schools indicated the use of adjunct instructors from industry, and they also use regular instructors for some of their courses. Half indicated that faculty had industry experience and almost half marked "formal training" in Software Engineering, with twenty percent indicating both industry experience and formal training. Only about twenty percent indicated neither industry experience nor formal training as instructor background. About twenty percent of the respondents did not mark the instructor information.

The Course Format

Approximately half of the respondents offer the course as a lecture course with the other half combining lecture and laboratory components. Very few schools mention a seminar format, and those are almost always the specialized, graduate courses. Even courses called Software Engineering Projects or Workshop contain a lecture component. Nearly all courses appear to be typical three semester hour or four quarter hour courses, with only a handful adding an extra credit for the lab component.

Only half a dozen mention the use of invited speakers and then usually only one, though as many as three speakers were reported. Speaker source is generally given as industry, though one respondent complained of a lack of local industry from which to draw.

Three responses indicate the use of case studies. While the case study is a common technique in many business and information systems courses, there seems to be a serious lack of such material readily available for software engineering. One mentioned the need for examples of good size projects, preferably documented to DOD standards.

The Projects

All but two of the courses reported include a project. Several of the advanced courses involve only an individual project, but almost all of the remaining courses use teams. Project sizes range from 500 to 5000 lines of code. Projects run the gamut from various industry and university systems to tools such as editors, plagiarism detectors, and prerequisite checkers. Only a very few mention more specific software engineering tools such as test harnesses, diagramming tools, or program quality/style measurement systems, which would seem to be natural projects for this type of class.

Projects account for 20 to 100 percent of the course grade, with most in the range from 40 to 70 percent. Team sizes range from small, two or three members, to large, around 15 members, with the vast majority being three to five members.

About half the projects are organized so that each team works on a different project, especially where teams work on projects in conjunction with local industry. About one third

use teams to duplicate the same project, while the remaining classes have several teams cooperating on a single project. Some vary the organization depending on the instructor or project. One school is considering "involving graduate students as group managers, technical writers from their expository writing program, etc."

Approximately three quarters of the projects are developed by the instructors, while most of the remainder are done in conjunction with local industry. A few projects involve other departments or organizations within the university. Several mention allowing students to develop projects themselves or to select from a list of projects. There were no complaints of difficulty with finding projects reported, unlike the earlier survey [2].

Most of the projects for the general courses include the requirements, design, coding and testing phases of the software life cycle. Two of the courses did not include implementing the project, citing a lack of time on the quarter system. Approximately half of the projects include some project management aspect(s), but less than ten percent include maintenance activities. This is not too surprising, given the brisk pace of the courses. The programs which integrate software engineering topics indicate more of an emphasis on software maintenance.

Several respondents comment on the importance of the team project. They feel that the combination of working in teams, and/or working on realistic projects (if not actual projects for local industry) is extremely important and provides considerable motivation for their students. This enthusiasm extends to the instructor as well, and seems to help reduce the burden of the extra work required for supervising projects (at least for the first few times the class is taught).

Other Activities

Quite a few of the respondents emphasize the effort devoted to documentation. Several mention the inclusion of users manuals, technical documentation, daily journals and/or program unit development folders. Others stress the emphasis on general writing and speaking skills. Two describe small group exercises designed to emphasize the difficulty of communication between teams. One mentions the importance of requiring the student teams to use and evaluate each other's software tools for quality of user documentation and ease of use. This provides a more authentic "acceptance testing" with strong motivation for repairing errors discovered by their peers.

Several respondents brought up the use of reviews or walkthroughs, because they feel that they are particularly beneficial for the students. Other project activities added by respondents include system integration, installation, and training. One incorporates some discussion of Software Psychology topics, especially computer-human interface concerns. Several mention using reviews of papers from the reading list as an activity.

The Tools

In response to the question on the use of tools, most indicate no use, though some mention editors, debuggers, compilers, hierarchical tree diagrams, Warnier diagrams, PDL, and the like. One respondent each uses of the following: Verdix Ada compiler and design documentation tools; Tektronix's SA system and WICOMO (the Wang Institute's implementation COCOMO); RCS; and USE.IT. No other application of automated, commercially available tools is indicated. A few report implementing software engineering tools as class projects. Another avenue would be to use and expand upon the UNIX environment which is often available, but this was only mentioned once.

Integrating Software Engineering

Efforts to integrate software engineering into other computer science courses were not widely reported by the survey respondents. Those mostly likely to complete that part of the survey were schools which did not offer separate software engineering courses.

The courses most often mentioned as integrating software engineering topics are CS1 and CS2 from the ACM curriculum [3], a two semester introductory programming sequence, the second semester of which introduces basic concepts of data structures, program verification, and algorithm analysis. A recent report from the CS2 curriculum committee [4] strongly recommends the inclusion of software engineering topics under the categories of specification, design, coding, and program

correctness, but does not specify further details. Of schools that responded to the question, about one half reply that they integrate software engineering into CS1 and/or CS2. They typically indicate about five to ten class hours spent on the various phases of the project life cycle, with another five hours spent on general subjects such as the software development cycle, walkthroughs, teams, documentation and software maintenance.

Some of the new introductory textbooks have added a short chapter on software engineering and a few introduce the use of hierarchy (structure) charts, but none mention other methodology. Testing and/or program verification usually receive some general discussion in these texts, but there is little material on specific techniques or formal terminology. There are many software engineering topics and techniques which could be presented at this level, but this is not yet happening. The general lack of software engineering expertise and the demands of teaching the advanced software engineering courses, will likely slow efforts in integrating software engineering into other courses. Given that many first year courses are taught by part-time instructors and/or teaching assistants, it could be even longer before software engineering enters the curriculum at this level.

Other courses, listed more than once as integrating software engineering topics, are data structures and programming languages courses. The project life cycle and software tools are the topics most often selected, though some say that they cover specification (specification languages), design (abstract data types), coding, testing and documentation in these courses.

III. CONCLUSIONS

One of the more noticeable features of the survey is the pride taken in the software engineering courses. The most frequent comment is one noting the positive feedback from students and/or local industry concerning the course.

The problems described by Petricig and Freeman [2] seem to have improved to a degree. The first obstacle they reported was a shortage of books and teaching materials, together with the difficulty of finding realistic software projects. Several new textbooks have appeared and more are under development. However, from the multiple titles listed as texts, one might conclude that some instructors do not find any one text adequate. Collections of reprints, case studies and examples are virtually nonexistent and would also be helpful. Because invited speakers are rarely used, low cost videotapes of knowledgeable speakers is a possible solution to encourage more class input from practicing software engineers. Projects do not seem to be in short supply, although one might expect more instructors to develop tools for the use of the software engineering classes themselves, setting a good example for current students and providing needed software for future classes.

Petricig and Freeman [2] described the problem of acquiring qualified instructors as being largely overcome by the use of part-time instructors from industry. The schools represented here generally use regular faculty, most of whom have either industry experience or formal training in software engineering. The earlier report described the problem of the large amount of instructor time required as largely unrecognized by their department, and this complaint is echoed in the current survey. Similarly, the lack of acceptance by other faculty is still prevalent. Still, one third of the undergraduate courses are required for graduation, which indicates considerable importance has been placed on the course by those departments. The courses offered at the graduate level are less likely to be required, but this is to be expected.

Several respondents comment on the difficulty of trying to cover software engineering theory, together with applying the ideas on a realistic project in a single semester. At the same time, they do not have the resources to offer multiple courses, much less a complete program in software engineering. Only ten percent of the schools can include entire specialized courses on requirements and/or design, maintenance, documentation, testing or project management, and none offer software engineering degrees at this point. There are at least three schools offering a Master of Software Engineering degree, and several more will begin programs shortly.

Recently, considerable effort has been expended on determining what a software

engineering curriculum [5], though most of the work has been directed at developing a masters degree program and expanding the current masters programs into doctoral programs. Both the new Software Engineering Institute established by DOD at Carnegie-Mellon and the Rocky Mountain Institute of Software Engineering established in Boulder, Colorado include improved software engineering education in their goals. The curriculum development, training institutes, and teaching materials under development are all vitally needed to develop advanced courses and to strengthen existing undergraduate courses such as those described here.

In summary, we see that software engineering courses are alive and well and their numbers are increasing rapidly. Software engineering is exciting, if somewhat demanding to teach, and somehow, experienced instructors are being found and pressed into service. While additional instructor training, teaching materials, and classroom-oriented automated tools are still needed, the project oriented software engineering course appears to be a viable and successful technique for imparting improved software design and development concepts.

REFERENCES

[1] A.A.J. Hoffman, "A Survey of Software Engineering Courses," SIGCSE Bulletin, vol. 10, no. 3, 1978.

[2] M. Petricig and P. Freeman, "Software Engineering Education: A Survey," SIGCSE Bulletin, Vol. 16, No. 4, 1984.

[3] R.H. Austing, B.H. Barnes, D.T. Bonnette, G. Engel and G. Stokes (Eds.), "ACM Curriculum Committee on Computer Science Curriculum '78 Recommendations for the Undergraduate Program in Computer Science," Communications of the ACM, Vol. 22. No. 3, Mar. 1978.

[4] E.B. Koffman, D. Stemple, and C.E. Wardle, "Recommended Curriculum for CS2, 1984: A Report of the ACM Curriculum Committee Task Force for CS2," Communications of the ACM, Vol. 28. No. 8, Aug. 1985.

[5] Proceedings of the Software Engineering Education Workshop, Sponsored by the Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PN, Feb, 1986

Appendix I
Software Engineering Textbooks

[1] L.J. Arthur, *Measuring Programmer Productivity and Software Quality*, New York: John Wiley and Sons, 1985.

[2] B. Beizer, *Software Testing and Quality Assurance*, New York: Van Nostrand Reinhold, 1984.

[3] B.W. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall, 1981.

[4] G. Booch, *Software Engineering with Ada*, Benjamin-Cummings Publishing Co., 1983.

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

[6] C. Browning, *Guide to Effective Software Technical Writing*, Englewood Cliffs, NJ: Prentice Hall, 1984.

[7] P. Bruce and S.M. Pederson, *The Software Development Project: Planning and Management*, New York: John Wiley and Sons, 1982.

[8] S.D. Conte, *Software Engineering Metrics and Models*, Benjamin-Cummings Publishing Co.

[9] T. DeMarco, *Structured Analysis and System Specification*, New York: Yourdon Press, 1978.

[10] T. DeMarco, *Concise Notes on Software Engineering*, New York: Yourdon Press, 1979.

[11] M.W. Evans, P. Piazza, and J.B. Dolleas, *Principles of Productive Software Management*, New York: John Wiley and Sons, 1983.

[12] R. Fairley, *Software Engineering Concepts*, New York: McGraw Hill, 1985.

[13] R.E. Filman and D.P. Friedman, *Coordinated Computing--Tools and Techniques for Distributed Software*, New York: McGraw Hill, 1984.

[14] C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, New York: Improved System Technologies Inc, 1977.

[15] R.L. Glass, *Real-Time Software*, Englewood Cliffs, NJ: Prentice Hall, 1984.

[16] P. Heckel, *The Elements of Friendly Software Design*, Warner, 1982.

[17] M. Jackson, *System Development*, Englewood Cliffs, NJ: Prentice Hall, 1983.

[18] R.W. Jenson and C.C. Tonies, *Software Engineering*, Englewood Cliffs, NJ: Prentice Hall, 1979.

[19] B. Kernighan and P.J. Plager, *Software Tools*, Addison-Wesley, 1976.

[20] C.P. Lecht, *The Management of Computer Programming Projects*, American Management Association.

[21] Liskov and Gutlag, *Abstraction and Specification in Program Development*.

[22] P.W. Metzger, *Managing a Programming Project*, Englewood Cliffs, NJ: Prentice Hall, 1981.

[23] G. Meyers, *Software Reliability: Principles and Practices*, New York: John Wiley and Sons, 1976.

[24] G. Meyers, *Composite/Structured Design*, New York: Van Nostrand, 1978.

[25] G. Meyers, *The Art of Software Testing*, New York: John Wiley and Sons, 1979.

[26] K. Orr, *Structured Systems Development*, New York: Yourdon Press, 1977.

[27] M. Page-Jones, *A Practical Guide to Structural Systems Design*, New York: Yourdon Press, 1980.

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

[29] P.C. Semprevivo, *Systems Analysis*, 2nd Edition, SRA, 1982.

[30] B. Shneiderman, *Software Psychology*, Winthrop Publishers, 1980.

[31] M.L. Shooman, *Software Engineering, Design, Reliability, and Management*, New York: McGraw Hill, 1983.

[32] Skees, *Writing Handbook for Computer Professionals*, Lifetime Learn, 1982.

[33] I. Sommerville, *Software Engineering*, Addison Wesley, 1982.

[34] R. Turner, *Software Engineering Methodology*, Reston, 1984.

[35] U.S. Government, *Defense System Software Development*, Washington, D.C.: Department of Defense, DOD-STD-2167, June 4, 1985.

[36] C.R. Vick and C.V. Ramamoorthy, *Handbook of Software Engineering*, New York: Van Nostrand Reinhold, 1984.

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

[38] R. Wiener and R. Sincovec, *Software Engineering with Modula-2 and Ada*, New York: John Wiley and Sons, 1984.

[39] E. Yourdon and L. Constantine, *Structured Design*, New York: Yourdon Press, 1975.

[40] E.N. Yourdon, *Classics in Software Engineering*, New York: Yourdon Press, 1979.

[41] M.V. Zelkowitz, A.C. Shaw, J.D. Gannon, *Principles of Software Engineering and Design*, Englewood Cliffs, NJ: Prentice Hall, 1979.

[42] M. Shaw, Abstraction Techniques in Modern Programming Languages, *IEEE Software 1*, 4, 10-26, October 1984.
[43] B. Shneiderman, Human Factors Experiments in Designing Interactive Systems, *Computer 12*, 12, 9-19, December 1979.
[44] K. Tai, Program Testing Complexity and Test Criteria, *IEEE Transactions on Software Engineering SE-6*, 6, 531-538, November 1980.
[45] D. Teichroew and E.A.H. III, PSL/PSA: A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems, *IEEE Transactions on Software Engineering*, January 1977.
[46] W. Teitlman, A Tour Through Cedar, *IEEE Transactions on Software Engineering SE-11*, 3, 285-302, March 1985.
[47] W.F. Tichy, Design, Implementation, and Evaluation of a Revision Control System, *IEEE Proceedings of the 6th International Conference on Software Engineering, Tokyo, Japan*, 58-67, September 1982.
[48] I. Vessey and R. Weber, Some Factors Affecting Program Repair Maintenance: An Empirical Study, *Communications of the ACM 26*, 2, 128-134, February 1983.
[49] R.C. Waters, The Programmer's Apprentice: A Session with KBEmacs, *IEEE Transactions on Software Engineering SE-11*, 11, 1296-1320, November 1985.
[50] S.N. Woodfild, An Experiment on Unit Increase in Problem Complexity, *IEEE Transactions on Software Engineering SE-5*, 2, 76-79, March 1979.
[51] W.A. Wulf, Trends in the Design and Implementation of Programming Languages, *Computer 13*, 1, January 1980.

Appendix II
Software Engineering Reading List

[1] W.R. Adrion, M.A. Branstad and J.C. Cherniavsky, Validation, Verification, and Testing of Computer Software, *ACM Computing Surveys 14*, 2, 159-192, June 1982.
[2] A. Avizienis, The N-Version Approach to Fault-Tolerant Software, *IEEE Transactions on Software Engineering SE-11*, 12, 1491-1501, December 1985.
[3] F.T. Baker, Chief Programmer Team Management of Production Programming, *IBM Systems Journal 11*, 56-73, 1972.
[4] L.L. Beck and T.E. Perkins, A Survey of Software Engineering Practice: Tools, Methods, and Results, *IEEE Transactions on Software Engineering SE-9*, 5 ,541-561, September 1983.
[5] G.D. Bergland, A Guided Tour of Program Design Methodologies, *Computer 14*, 10, 13-37, October 1981.
[6] K. Bo, Human-Computer Interaction, *Computer 15*, 2, 9-11, November 1982.
[7] B.W. Boehm, Software and Its Impact: A Quantitative Assessment, *Datamation 19*, 5, 48-59, May 1973.
[8] B.W. Boehm, Software Engineering, *IEEE Transactions on Computers C-25*, 12, December 1976.
[9] B.W. Boehm, Software Engineering Economics, *IEEE Trasactions on Software Engineering SE-10*, 1, 4-21, January 1984.
[10] B.W. Boehm, M.H. Penedo, E.D. Stuckle and A.B. Pyster, A Software Development Environment for Improving Productivity, *IEEE Computer 17*, 6, 30-44, June 1984.
[11] F.P. Brooks Jr., The Mythical Man-Month, *Datamation*, 17-24, December 1974.
[12] T. Carey, User Differences in Interface Design, *Computer 15*, 2, 14-20, November 1982.
[13] D. Chapman, A Program Testing Assistant, *Communications of the ACM 25*, 9, 625-634, September 1982.
[14] B.G. Claybrook, A Specification Method for Specifying Data and Procedural Abstractions, *IEEE Transactions on Software Engineering SE-8*, 5, 449-459, September 1982.
[15] B.J. Cox, Message/Object Programming: An Evolutionary Change in Programming Technology, *IEEE Software 1*, 1, 50-61, January 1984.
[16] B. Curtis, Measurement and Experimentation in Software Engineering, *Proceedings of the IEEE 68*, 9, 1144-1157, September 1980.
[17] R.E. Davis, Logic Programming and Prolog: A Tutorial, *IEEE Software 2*, 5, 53-62, September 1985.
[18] E. Dijkstra, Go To Statement Considered Harmful, *Communications of the ACM 11*, 3, 147-148, March 1968.
[19] E. Dijkstra, The Humble Programmer, *Communications of the ACM 15*, 10, 859-866, October 1972.
[20] E. Dijkstra, Structured Programming, *Software Engineering, Concepts and Techniques, Buxton, Naur and Randell (editor)*, Litton Educational Publishing, Inc., 1976.
[21] M.E. Fagan, Design and Code Inspections to Reduce Errors in Program Development, *IBM System Journal*, No. 3, 182-211, 1976.
[22] J.D. Gannon and J.J. Horning, The Impact of Langauge Design on the Production of Reliable Software, *Proceedings of 1975 International Conference on Reliable Software*, Los Angeles, CA, April 21-23, 1975. Reprinted in *ACM SIGPLAN Notices 10*, 6, 10-22, June 1975.
[23] W.E. Howden, The Theory and Practice of Functional Testing, *IEEE Software 2*, 5, 6-17, September 1985.
[24] C. Jard and G.V. Bochmann, An Approach to Testing Specifications, *ACM Software Engineering Notes 8*, 4, 53-59, August, 1983.
[25] J.K. Kearney, et. al., Software Complexity Measurement, tentatively accepted for publication in: *Communications of the ACM*, submitted: January 1985.
[26] B.W. Kernighan, The UNIX System and Software Reusability, *IEEE Transactions on Software Engineering SE-10*, 5, 513-518, September 1984.
[27] M. Mantei, The Effect of Programming Team Structures on Programming Tasks, *Communications of the ACM 24*, 3, 106-113, March 1981.
[28] R.G. Mays, et. al., PDM: A Requirements Methodology for Software System Enhancements, *IBM Systems Journal 24*, 2, 134-149, 1985.
[29] B. Meyer, On Formalism in Specifications, *IEEE Software 2*, 1, 2-26, January 1985.
[30] J.A. Mills, A Pragmatic View of the System Architect, *Communications of the ACM 28*, 7, 708-717, July 1985.
[31] S.N. Mohanty, Software Cost Estimation: Present and Future, *Software--Practice and Experience 11*, 103-121, 1981.
[32] J.D. Musa, Software Reliability Measurement, *The Journal of Systems and Software 1*, 223-241, 1980.
[33] D.L. Parnas, On the Criteria Used in Decomposing Systems into Modules, *Communications of the ACM 15*, 12, 1035-1058, December 1972.
[34] D.L. Parnas, P.C. Clements and D.M. Weiss, The Modular Structure of Complex Systems, *IEEE Transactions on Software Engineering SE-11*, 3, 259-266, March 1985.
[35] D.L. Parnas, Software Aspects of Strategic Defense Systems, *Communications of the ACM 28*, 12, 1326-1335, December 1985.
[36] N.H. Petschenik, Practical Priorities in System Testing, *IEEE Software 2*, 5, 18-23, September 1985.
[37] C.V. Ramamoorthy and F.B. Bastani, Software Reliability--Status and Perspectives, *IEEE Transactions on Software Engineering SE-8*, 4, 354-371, July 1982.
[38] C.V. Ramamoorthy et. al., Software Engineering, *Computer 17*, 10, 191-209, October 1984.
[39] D.J. Reifer and S. Trattner, A Glossary of Software Tools and Techniques, *Computer*, 6-14, July 1977.
[40] G. Roman, A Taxonomy of Current Issues in Requirements Engineering, *Computer 18*, 4, 14-23, April 1985.
[41] D.T. Ross., Applications and Extensions of SADT, *Computer 18*, 4, 25-34, April 1985.