AN ECLECTIC SOFTWARE ENGINEERING BIBLIOGRAPHY

by

Peter Freeman

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

May, 1979
Technical Report #136

Many of the entries in this bibliography are taken from three sources: Software Design Techniques (Freeman and Wasserman), Software Engineering Education (Wasserman and Freeman), and Software Engineering and Its Structured Methodologies (Freeman and McGowan); grateful acknowledgement is made to my colleagues. A few more recent items have been added. The references range from introductory to advanced research and from educational practice to practical guidelines. The bibliography is in no way complete nor exhaustive, but it does provide an entry to literature covering most aspects of software engineering.

Aron, J. D., The Program Development Process. Reading, Mass.: Addison-Wesley Publishing Company. Part 1: The Individual Programmer, 1974. Part 2: The Programming Team (to appear, 1977).

Aron takes a balanced view of the software development process, focusing concurrently on computer science/programming methodology issues and on management techniques. Part 1 covers problem analysis, program design, coding, and debugging. Part 2 will cover the system development life cycle, system analysis, and project management techniques, including planning, resource management, and documentation. Aron does not go into great depth on these topics, but manages to raise all of the important points and gives numerous references to other work.

Baker, F. T., "Chief Programmer Team Management of Production Programming," IBM Systems Journal vol. 11, no. 1 (1972), pp. 56-73.

This paper describes the team organization first used in development of the New York Times information bank. The team organization involves four separate management techniques: 1) a functional organization in which creative work is separated from clerical work, with a highly skilled chief programmer responsible for all design and code production; 2) a program production library, consisting of an internal library, an external library, machine procedures, and office procedures which serve as rules for managing the library; 3) top-down programming, and; 4) structured programming. While this approach was found to be an effective management technique in that environment, it should be carefully tested and evaluated for applicability to other settings.

Bauer, F. L., (ed.) Advanced Course on Software Engineering. Berlin: Springer-Verlag, 1973.

The collection of papers represents the text material for a short course on software engineering given in early 1972. As such, it is one of the few books which is even marginally suitable for a software engineering course. The quality of the material is somewhat erratic, but there are some excellent sections, particularly those on portability and adaptability and on performance prediction. Unfortunately, there is no material oneither specification problems or verification, two central areas of software engineering.

Bentley, J. and Shaw, M., "An Alphard Specification of a Correct and Efficient Transformation on Data

Structures," Proceedings Conference on Specifications for Reliable Software, IEEE Computer Society, 1979, p. 222 ff.

Bentley and Shaw illustrate in a detailed and understandable way how data abstraction and program transformation techniques may some day be a very powerful tool for software engineering.

Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," Datamation, May, 1973. (Reprinted in Freeman and Wasserman, 1977)

This classic paper was one of the first public documentations of the need for improved software development practices.

Boehm, B. W., "Software Engineering," IEEE Trans. on Computers, December, 1976.

This paper provides a comprehensive survey of the state-of-the-art of software engineering as of 1976 and provides many references.

Brooks, F. P., Jr., The Mythical Man-Month. Reading, Mass.: Addison-Wesley Publishing Company, 1975.

Brooks relates the lessons of his experience as manager of the OS/360 development effort with clarity and wit. Brooks' Law, "Adding people to a late software project makes it later," refers to the non-interchangeability of people and time, thereby motivating the book's title. Brooks also focuses on team organization for software development and the value of documentation. The reader must be careful not to miss the important points hidden among photographs of Babe Ruth, dinosaurs, and Mickey Mouse or the valuable lessons interwined with pithy quotations from Goethe, Swift and Ovid.

Caine, S. H., and Gordon, E. K., "PDL--A Tool for Software Design," Proc. AFIPS 1975 National Computer Conference, pp. 271-276.

This paper describes a Program Design Language (PDL), a tool for the production of well-structured designs. PDL, which may be thought of as "structured English," is used to show the detailed structure and logic of program modules at the design stage. The working design document which is produced then serves as the basis for programming the given system. The authors have found that PDL is a useful tool in

the design of both large and small programs and that it provides a useful measure of the quality and degree of completeness of a problem solution.

Dahl, O-J, Dijkstra, E. W., and Hoare, C. A. R., Structured Programming, London: Academic Press, 1972.

This book consists of three monographs: "Notes on Structured Programming," by Dijkstra, "Notes on Data Structuring," by Hoare, and "Hierarchical Program Structures," by Dahl and Hoare. Dijsktra's paper was perhaps the single most influential document in the so-called "programming revolution," and typewritten copies were privately circulated for some time before publication here. Dijkstra used the word "programming" in the broadest possible sense, encompassing the entire problem-solving process, and presents several meticulous examples of systematic program construction. It should be noted that Dijkstra's use of the Hoare's paper is the outline of a theory of data structuring. It contains some elements of his work on axiomatizations of computer programming, but is most interesting for the view it gives of the design of PASCAL. (Warning: This book may change your life.)

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

This book describes the Yourdon approach to structured analysis.

DeRemer, F., and Kron, H., "Programming-in-the-Large versus Programming- in-the-Small," IEEE Transactions on Software Engineering, vol. SE-2, no. 2 (June, 1976) pp. 80-86.

DeRemer and Kron distinguish the problem of writing small programs from that of writing large programs. While present programming languages are largely suitable for writing modules (small programs), a distinct language is needed for structuring a large collection of modules. DeRemer and Kron outline the objectives of such a Module Interconnection Language (MIL) and show the application of the MIL to a theorem-proving program.

Dijkstra, E. W., "The Humble Programmer," CACM, vol. 15, no. 10 (October 1972), pp. 859-866.

This lecture, given upon Dijkstra's receipt of the ACM Turing Award, combines a highly selective view of historical developments with a perspective on revolutionary developments in programming. Dijsktra regards it as

necessary that our approach to software development be drastically changed in order to achieve software reliability. He emphasizes the intellectual complexity of the programming task, the need for tools and techniques to manage that complexity, and the need to recognize the intrinsic limitations of the human mind in this regard.

Elspas, B., et al., "An Assessment of Techniques for Proving Program Correctness," ACM Computing Surveys, vol. 4, no. 2 (June 1972), pp. 97-147.

This paper surveys the entire range of approaches to proving the correctness of programs. It begins with a discussion of abstract programs (program schemas). The paper discusses both informal and formal approaches to verification, beginning with the Floyd-Naur techniques of inductive assertions, proceeding through a discussion of theorem-proving techniques, and then covering formal approaches to the verification of programs, focusing on work by Manna. Although there are a large number of more recent papers on this subject, this paper represents an excellent starting point and provides a good framework for understanding current work. A more recent survey with a good set of references is London, R., "A View of Program Verification," Proc. 1975 Int'l. Conference on Reliable Software, pp. 534-545.

Fagan, M., "Design and Code Inspections to Reduce Errors in Program Development." IBM Systems Journal, Vol. 15, No. 3, pp. 182-211, 1976.

This is a good overview of the results obtained by IBM using design walkthroughs and code inspections. It includes detailed suggestions of how to conduct such reviews.

Fitzsimmons, A. and Love, T., "A Review and Evaluation of Software Science," ACM Computing Surveys, Vol. 10, pp. 3-18, March, 1978.

This paper provides a very readable introduction to the topic of software science (also known as software physics).

Freeman, P., "Software Design Representation: A Case Study" and "Software Design Representation: Analysis and Improvements," Software Practice and Experience, Vol. 8 (501-528), 1978.

The design representations used on an operating system design project are described and then analyzed to indicate areas needing improvement.

Freeman, P. and McGowan, C., Software Engineering and Its Structured Methodologies. Chicago: Advanced Systems Inc. Video-Assisted Instruction Course #1480, 1978.

This videotape introduction to software engineering is intended for managers and others needing a broad introduction to the concepts and techniques of software engineering. It includes printed study guides with some material expanding on the video introductions.

Freeman, P. and Wasserman, A. I., (eds.). Software Design Techniques, Second Edition, Long Beach: IEEE Computer Society, 1977.

This volume contains 23 reprints plus over 100 pages of original material focusing on software design techniques but also covering requirements definition and other topics.

Gane, C. and Sarson, T., Structured Systems Analysis. New York: Yourdon Press, 1978.

Describes Structured Systems Analysis - a collection of techniques intended to strengthen traditional systems analysis.

Gerhart, S. L., and Yelowitz, L., "Observations of Fallibility in Applications of Modern Programming Methodologies," IEEE Trans. Sfw. Eng., SE-2,3 (September, 1976).

The authors have noted a number of errors in published algorithms that are used in various ways in expositions of modern programming practices. Analysis of these errors leads them to suggestions for avoiding such problems.

Goodenough, J. B., and Gerhart, S. L., "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, vol. SE-1, no. 2 (June 1975), pp. 156-173.

The authors point out the importance of testing in software development and take some steps toward developing a theory of testing. Their goal is to define a means for selecting test data such that successful program operation for that test data will yield a high degree of confidence and that the program will operate successfully for all data. With certain constraints upon their definitions of test reliability and validity, they show that properly structured tests can be used to demonstrate the absence of errors from a program.

Hice, G. F., Turner, W. S., and Cashwell, L. F., System Development Methodology. Amsterdam and New York: North-Holland/American Elsevier Publishing Company, 1974.

This book contains a wealth of suggestions for project management. The authors have developed a coordinated and well-documented approach to the development of man/machine information systems. Their methodology consists of seven activities: definition study, preliminary design, detail design, program and human job development, testing, data conversion and system implementation, and operations and maintenance. A chapter is devoted to each activity and each includes detailed lists of methods and products for that stage: altogether, a useful book for the serious system developer.

Horowitz, E., (ed.) Practical-Strategies for Developing Large Software Systems. Reading, Mass.: Addison-Wesley Publishing Company, 1975.

This book is an outgrowth of a one-week seminar held in 1974 in which a number of topics relevant to the creation of large software systems were addressed. The book consists of the edited and expanded versions of presentations made at the seminar. Included among the authors are Barry Boehm, Jules Schwartz, Leon Stucki, and Ray Wolverton. The papers are generally good and present a number of interesting ideas relating to design as well as other aspects of system development.

Hugo, I., "A Survey of Structured Programming Practice," Proceedings 1977 National Computer Conference. AFIPS Press, Montvale, N.J., pp. 741-747.

The results of a survey by InfoTech of 300 companies worldwide to determine the extent and type of the involvement in structured programming.

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

The Jackson Design method is presented here covering data and problem structure, structure clashes, program inversions, backtracking and error handling. This is a teachable technique that works on a wide variety of business data processing problems. The presentation is terse but complete.

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

Important software engineering topics including design, programming, verification and validation, security and privacy, management, and legal aspects of software development are treated from the viewpoint of practicing professionals. In addition to the editors, contributing authors include Enos, Van Tilberg, Deutsch, Hascall, and Reddien.

Kernighan, B. W., and Plauger, P. J., The Elements of Programming Style New York: McGraw-Hill Book Company, 1974.

Kernighan, B. W., and Plauger, P. J., Software Tools, Reading, Mass.: Addison-Wesley Publishing Company, 1976.

Kernighan and Plauger set out, in these two volumes, to improve life for the average programmer. In the Elements of Programming Style, patterned after Strunk and White's The Elements of Style (Macmillan, 1972) they present a set of rules for improving the correctness and comprehensibility of computer programs. These rules are illustrated by examples of poorly constructed, often incorrect programs, which are rewritten to improve their quality. Software Tools shows how to build good programs which can be used to build other programs. The book contains thousands of lines of actual program code and usable tools for a variety of text (or program) processing tasks. The authors' consistency of programming style and clarity of explanations make it possible to understand even the largest of their programs.

Knuth, D. E., The Art of Computer Programming. Reading, Mass.: Addison-Wesley Publishing Company.
Vol. 1 - Fundamental Algorithms, 2nd ed., 1973
Vol. 2 - Seminumerical Algorithms, 1969
Vol. 3 - Sorting and Searching, 1973
Vol. 4 - Forthcoming

These books belong in the collection of every serious computer scientist. From solid foundation for the discipline, and to provide proper mathematical approaches to the subject. For software engineers, the major contribution lies in the development of methodology for the analysis of algorithms. Throughout, Knuth manages to be mathematically rigorous and historically precise without losing sight of the practical problems which have motivated much of the work in computer science. I anxiously await the future volumes.

Knuth, D. E., "Structured Programming with goto Statements," ACM Computing Surveys, vol. 6, no. 4 (December, 1974), pp. 251-301.

Although the literature is seemingly filled with endless discussion of the merits of writing programs which contain unrestrained transfers of control, no paper summarizes the issues more clearly than this one. Knuth presents many examples showing alternate language constructs which reduce the need for the goto, along with examples in which elimination of the goto leads to unwieldy or inefficient programs. Knuth observes that the ongoing debate over the goto is merely a symptom of the larger issue of language support for abstraction and that there is a considerable need for additional reseach in programing language design.

Liskov, B., and Zilles, S. N., "An Introduction to Formal Specifications of Data Abstractions," in Current Trends in Programming Methodology, vol. 1, ed. R. Yeh. Englewood Cliffs, N.J.: Prentice-Hall, 1977.

This paper examines the general problem of specifying the semantics of abstract data types. The authors discuss graphical methods, axiomatic specifications, and algebraic specifications. The notion of a data abstraction provides a framework by which the operations upon an object may be formally specified. The problems of formal specification for data abstractions is just part of the much larger problem of being able to develop formal specifications for an entire program. However, many of the techniques discussed here would seem to be applicable to the larger problem as well.

Liskov, B., and Zilles, S. N., "Programming with Abstract Data Types," ACM SIGPLAN Notices, vol. 9, no. 4, (April, 1974), pp. 50-59.

When one refines the solution to a problem, it is necessary to refine the data structure along with the procedure. One thinks in terms of "abstract data structures," which are eventually mapped into a physical data structure. An abstract data type is a form of programming language extension whereby one may introduce a new data type into the language, along with the representation of that type and the set of allowable operations on objects of that type. The representation and the set of operations are encapsulated in such a way that the representation of the type is hidden from external modules, which are constrained to use only the given set of operations. The use of abstract types in the programming language CLU is shown.

Lucas, H. C., Jr., Toward Creative Systems Design New York: Columbia University Press, 1974.

Lucas identifies three major problems in the design of information systems: technical, organizational behavior, and project management. In this short treatise, he concentrates on the organizational problems (entire organization, groups, and individuals) and puts forth a design approach intended to use creatively the inevitable behavior changes that introduction of an information system will bring about. This book provides a good balance to the purely technical aspects of design normally dealt with in the course on design and gives an introduction to the managerial and behavioral aspects.

McGowan, C., and Kelly, J. R., Top-Down Structured Programming Techniques New York: Petrocelli-Charter, 1975.

A best selling book in this area. The title describes its contents.

Merwin, R. E., "Software Management: We Must Find a Way," IEEE Trans. Sfw. Eng., SE-4,4 (July, 1978).

This guest editorial introduces a special section on software management that contains several useful papers.

Myers, G. J., Software Reliability: Principles and Practice New York: John Wiley and Sons, 1976.

Myers gives an overview of the topics which affect software reliability, giving primary attention to issues of program design and testing, but also touching upon several related topics, including programming languages, verification, and management techniques. Although the book does not contain any exercise, it is the first book which is truly suitable as a text for a software engineering course.

Myers, G. J., Reliable Software Through Composite Design. New York: Petrocelli/Charter, 1974.

"Structured design" is an approach to program development which incorporates a collection of considerations and techniques intended to simplify the coding and debugging process. The author recommends the use of a documentation aid such as HIPO and the use of structured programming techniques for coding. The book stresses the need for simplicity in system design, particularly as it applies to the coupling of modules. (Much of this work is based on work by L. L. Constantine--see, for example, Stevens, W. P., Myers, G. J., and Constantine, L. L., "Structured Design," IBM Systems Journal vol. 13, no. 2 (1974), pp. 115-130.)

Naur, P., and Randell, B., (ed.) Software Engineering Brussels: NATO Science Committee, 1969. Buxton, J. N., and Randell, B., (ed.) Software Engineering Techniques. Brussells: NATO Science Committee, 1970. (Both volumes out of print. Reprinted as Naur, P., Randell, B., and Buxton, J. N., (ed.) Software Engineeering: Concepts and Techniques. New York: Petrocelli/Charter, 1976.)

These volumes are the proceedings from the NATO conferences for which the term "software engineering" was coined. Although their historical value alone makes them essential reading, the volumes also contain a number of excellent short papers and remarks. Dijkstra's earliest remarks on structured programming are found here, along with his concept of an "abstract data structure" and his now-controversial statement, "Testing shows the presence, not the absence of bugs." Despite the intervening years, these volumes remain very timely (which may show the difficulty of making substantial progress in this discipline).

Orr, K., Structured Systems Development. New York: Yourdon, Inc. 1977.

This book describes Ken Orr's variation on the Warnier approach. It uses the data structure of the output to derive a structured program that computes this output. While very readable, the book is more of an introduction than a complete guide.

Parnas, D. L., " A Technique for Software Module Specifications with Examples," CACM, vol. 15, no. 5, (May 1972), pp. 330-336.

Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," CACM vol. 15, no. 12 (December 1972), pp. 1053-1058.

The two papers explain the principles of modularity by which one can improve the flexibility and comprehensibility of system design. Use of this technique permits one module to be written with little knowledge of the code of other modules and permits modules to be replaced independently. Proper use of modularity permits the inputs, outputs, and the function of each module to be visible, while hiding the essential details of implementation. Both of these papers have been heavily cited in more recent work. A more recent paper, by Parnas and D. P. Siewiorek, CACM, vol. 18, no. 7 (July 1975). discusses the concept of transparency in system design.

Simon, H. A., The Science of the Artifical. Cambridge, The

MIT Press, 1969.

This book is a written version of the Karl Taylor Compton lectures delivered by Simon in 1968, plus a reprint of one of his earlier papers. It provides very thought-provoking reading on the nature of those activities -- such as management science and computer science--that deal with man-made phenomena. A central theme of the lectures is the role of design in these disciplines. His discussion of the necessary skills for design is of particular interest.

Warnier, J., Logical Construction of Programs. New York: Van Nostrand Reinhold, 1974.

This is an English translation (and apparently not a very good one) of Warnier's L.C.P. approach. It uses the data structure of the output to derive a structured program that computes this output. While very readable, the book is more of an introduction than a complete guide.

Wasserman, A. I., and Freeman, P., (eds.). Software Engineering Education: A Status Report. New York and Berlin: Springer- Verlag, 1976.

A one-day workshop was held at UCI in 1976 to help establish communication between industry leaders concerned with technical education and university faculty trying to address those needs. This volume contains position papers prepared for the workshop and edited discussion from it.

Wasserman, A. I., and Stinson, S. K., "A Specification Method for Interactive Information Systems," Proc. Conf. on Specifications for Reliable Software, IEEE Computer Society, 1979.

This paper describes several views of a specification and a method of specifying the system structure for an interactive information system.

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

Weinberg studied programming not from this technical standpoint, but rather from its standpoint as a human activity. This book discusses such topics as the personality factors which make for good programmers, the social structure of programming projects, and the group interactions among programmers. Weinberg advanced the concept of "egoless programming," in which programmers work as a group to produce good programs, and routinely work to correct and improve one another's programs without ego

attachment to one's "own" code.

Wirth, N., "Program Development by Stepwise Refinement," CACM, vol. 14, no. 4 (April 1971), pp. 221-227.

This short paper illustrates the approach of developing programs by breaking tasks into subtasks by refining program and data specifications in parallel. Much of the paper is devoted to the stepwise generation of partial solutions to the 8-queens problem. Wirth's intent in the paper is to show the need for examining design decisions at each stage of refinement and to illustrate the difficulty of careful programming.

Wolverton, R. W., "The Cost of Developing Large-Scale Software." IEEE Transactions on Computers vol. C-23, no. 6 (June, 1974), pp. 615-636.

This paper discusses the problems of forecasting software costs. Wolverton discusses the entire software development cycle as practiced in organizations which produce software under contract. He discusses a number of techniques for cost estimation, allocating labor costs and computing costs over both time and development steps. Finally, a number of questions are raised concerning the effects of programmer productivity and system complexity on costs. While the paper emphasizes the TRW experience in producing large scale software systems for the military, many of these notions are applicable in other settings as well.

Yourdon, E., Techniques of Program Structure and Design. Englewood Cliffs: Prentice-Hall, 1975.

This book is more oriented to the lower levels of design and programming then many of the references in this list. Nonetheless, it provides lucid discussions of top-down program design, modular programming, structured programming and programming style.

Yourdon, E., and Constantine, L. L., Structured Design: 2nd Edition. New York: Yourdon, Inc., 1978.

This is the primary reference for this method.

The journals Software: Practice and Experience and IEEE Transactions on Software Engineering are good sources for articles, as are conference proceedings such as the 1975 International Conference on Reliable Software and the

International Conference on Software Engineering (1974, 1976, 1978, 1979), and Specifications for Reliable Software Conference (1979). Most proceedings are available from one of the following sources:

IEEE Computer Society
5855 Naples Plaza
Long Beach, CA 90803

Assoc. for Computing Machinery
1133 Avenue of the Americas
New York, NY 10036

Zelkowitz, M., "Perspectives on Software Engineering," ACM Computing Serveys vol. 10, no. 2, June 1978.

Professor Zelkowitz gives a rather complete survey of the state of the art in software engineering.