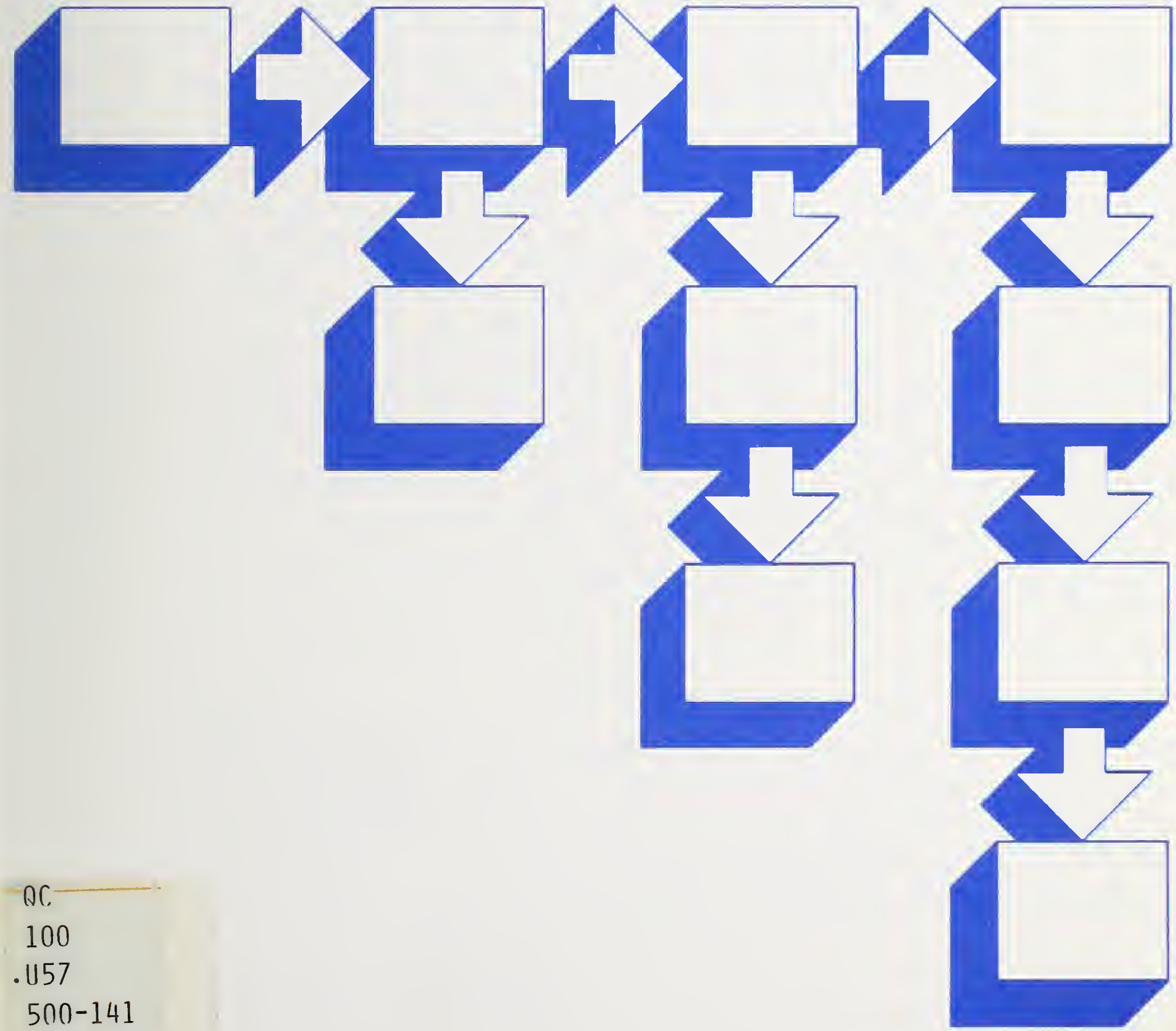
Computer Science and Technology

NBS Special Publication 500-141

Annotated Bibliography on Software Maintenance

Wilma M. Osborne and Ron Raigrodski



The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, the Institute for Computer Sciences and Technology, and the Institute for Materials Science and Engineering.

The National Measurement Laboratory

Provides the national system of physical and chemical measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; provides advisory and research services to other Government agencies; conducts physical and chemical research; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

- Basic Standards[2]
- Radiation Research
- Chemical Physics
- Analytical Chemistry

The National Engineering Laboratory

Provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

- Applied Mathematics
- Electronics and Electrical Engineering[2]
- Manufacturing Engineering
- Building Technology
- Fire Research
- Chemical Engineering[2]

The Institute for Computer Sciences and Technology

Conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

- Programming Science and Technology
- Computer Systems Engineering

The Institute for Materials Science and Engineering

Conducts research and provides measurements, data, standards, reference materials, quantitative understanding and other technical information fundamental to the processing, structure, properties and performance of materials; addresses the scientific basis for new advanced materials technologies; plans research around cross-country scientific themes such as nondestructive evaluation and phase diagram development; oversees Bureau-wide technical programs in nuclear reactor radiation research and nondestructive evaluation; and broadly disseminates generic technical information resulting from its programs. The Institute consists of the following Divisions:

- Ceramics
- Fracture and Deformation[3]
- Polymers
- Metallurgy
- Reactor Radiation

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Gaithersburg, MD 20899.
[2]Some divisions within the center are located at Boulder, CO 80303.
[3]Located at Boulder, CO, with some elements at Gaithersburg, MD.

NBS Special Publication 500-141

Annotated Bibliography on Software Maintenance

Wilma M. Osborne and Ron Raigrodski

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

September 1986

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Library of Congress Catalog Card Number: 86-600579
National Bureau of Standards Special Publication 500-141
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-141, 138 pages (Sept. 1986)
CODEN: XNBSAV

U.S. Government Printing Office
Washington: 1986

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

ANNOTATED BIBLIOGRAPHY ON SOFTWARE MAINTENANCE

Institute for Computer Sciences and Technology
National Bureau Of Standards
Gaithersburg, MD 20899

ACKNOWLEDGEMENTS

This report is the result of the untiring efforts of a number of individuals. Special thanks to Monica Holman, who spent many hours reviewing computer books, journals, and news articles, and to Bonnie Smith, Debbie Jackson, and Sharon Federline for their assistance in the data entry of this material. The dedicated efforts of these individuals helped to make this report possible.

ABSTRACT

This annotated bibliography contains summaries of two hundred and eighty-five software maintenance articles or papers from computer science journals, books, proceedings, Federal publications, computer newspapers, and other technical reports. It covers a fifteen year period between 1972 and 1986, and presents an overview of the various aspects of software maintenance including problems and issues faced in most software maintenance environments. It identifies techniques, procedures, methodologies, and tools that have been effectively employed throughout the software system lifecycle to improve the quality of that system.

Key words: documentation, metrics, productivity, programmers, software configuration management(SCM), software errors, software lifecycle, software maintenance cost, software packages, software quality, techniques, testing, tools, users.

INTRODUCTION

This annotated bibliography contains summaries of two hundred and eighty-five software maintenance articles or papers from computer science journals, books, proceedings, Federal publications, computer newspapers, and other technical reports. It covers a fifteen year period between 1972 and 1986, and presents an overview of the various aspects of software maintenance including management and technical issues faced in most software maintenance environments. It identifies techniques, procedures, methodologies, and tools that have been effectively employed throughout the software system lifecycle to improve the quality of that system.

The abstracts or summaries describe factors to be considered whether planning to make a software change or ensuring that the change has been properly and correctly incorporated. Many of the summaries suggest actions and procedures that can help software maintenance organizations satisfy the growing demands of maintaining existing systems. Others suggest steps that can be taken to improve the quality and maintainability of the code during development. Still others describe activities found to be effective in reducing the cost and difficulty of software maintenance.

This bibliography provides a comprehensive collection of information on how many organizations handle the software maintenance process and the associated problems. It contains information based on case studies and actual applications from software maintenance organizations in Government and industry, as well as material from the software engineering research community. It is intended as a basic reference for Federal ADP managers and software maintainers with responsibility for lifecycle software quality and maintainability, as well as for researchers, practitioners, and vendors of software maintenance tools.

A consistent theme throughout this bibliography is that there is an increasing need for effective management control over the software engineering process throughout the software system lifecycle.

The following cross reference is provided to aid in searching for articles on a specific topic. References which address several issues have been listed under multiple categories as a convenience to the reader.

CROSS REFERENCE

DOCUMENTATION

[AGRE82a]
[ANDE81]
[BASI82]
[BERN84]
[BOSS82]
[BRIC84]
[BROO82B]
[CLAR80]
[DUNN84a]
[DUNN84b]
[EBER80]
[FIPS105]
[FORN83a]
[FORN83b]
[GILL83a]
[GUND73]
[HANN85]
[HOWA84b]
[LAFF77]
[LETO86]
[LIU76]
[MERR78]

DOCUMENTATION

[MILS81]
[MUNS81]
[MURP82]
[PARI81]
[PARI82]
[PENN80]
[PEER81]
[RICH84]
[SCHN82a]
[SCHN82b]
[SCHN83]
[SHEP81]
[SING80]
[SNEE84]
[SOLO84]
[STAN77]
[TANN80]
[TAUT84]
[TINN84]
[TEIC77]
[WILS81]

METRICS

[AGRE82b]
[ARNO83]
[ARTH83]
[BASI80]
[BASI84]
[BELL84]
[BOEH78]
[CLAP81]
[CURT79]
[DEAN82]
[HARR82]
[YAU80b]

METRICS

[HARR84]
[MCCCA77]
[MCCA83]
[NARR84]
[NBS99]
[PERL81]
[RAMA84]
[ROMB84]
[SILV83]
[WALT78]
[YAU80a]

PRODUCTIVITY

[ARTH83]
[BASS84]
[BELL84]
[COUG83]
[COUG84]
[FERR81]
[FINK77]
[FORN83b]
[FRAN83]
[GROS82]
[HOWA83a]
[HOWA84a]
[JONE78]
[LANE84]
[LEAV83]

PRODUCTIVITY

[LIEN78b]
[LIEN80b]
[LIEN83]
[MANC83]
[MART82]
[MILL83]
[MURR84]
[NBS74]
[OSBO86]
[PATT81]
[PRES81]
[RYAN82]
[TINN83]
[ZELL83]

SOFTWARE CONFIGURATION MANAGEMENT (SCM)

[BANN83]
[BERS79]
[BERS80]
[BRYA84]
[CANN72]
[DERO78]
[NBS129]

[DUNN82]
[DUNN84a]
[FEIN84a]
[GLAS82]
[LAFF77]
[MAUL83]
[ZAK83]

SOFTWARE ERRORS

[BASI84]
[BROO82a]
[CODI77]
[DONA80a]
[DUNN82]
[ENDR75]
[FERR81]
[GLAS81a]
[GREM84]
[GUND73]
[KEID74]
[KULL83]
[LIPO79]

SOFTWARE ERRORS

[LITT80]
[MALM77]
[MAMR77]
[MAUL83]
[MYER78]
[MYER79]
[PARI80]
[SCHN79]
[SHEP79]
[SHOO83]
[SWAN79]
[WEIS81]

SOFTWARE LIFECYCLE

[BERS79]
[BOEH78]
[BOEH81b]

SOFTWARE LIFECYCLE

[LIEN78a]
[LIPO79]
[MCCL81]

[COOP78]
[DERO78]
[DEUT82]
[DONA80b]
[EDWA84]
[FIPS105]
[FIPS106]
[FRAN83]
[GLAS81b]
[GREE81]
[HIGG81]
[LANE84]
[LEHM77]
[LERN83]

[MILL77]
[MUNS82]
[MUSH84]
[NAVE79]
[OSBO84]
[PODO77]
[PUTN80]
[RAMA84]
[RAYN83]
[SHNE86]
[WALK81]
[WHIT77]
[WIEN84]
[YAU82]

SOFTWARE MAINTENANCE COST

[ARNO86]
[BASI80]
[BELL83]
[BOEH81b]
[CHAP84]
[DERO78]
[DONE84]
[ELSH82]
[FRAN83]
[GAO81]
[GAO83]
[GILL83a]
[GILL83b]
[GUIM83]
[HARR83]

SOFTWARE MAINTENANCE COST

[INMO84]
[KAPU83]
[LAFF78]
[MCKE84]
[MUNS82]
[OVER74]
[PENN80]
[PRES82]
[RAMA84]
[REUT81]
[SHAR77]
[SHOO83]
[SING80]
[WHIT81]

SOFTWARE PACKAGES

[GILL83b]
[GROC84]
[HOWA84b]
[MAJO84]
[MART82]
[NBS114]

SOFTWARE PACKAGES

[PARI82]
[PARI84]
[SPOK84]
[WASS78]
[WASS82]

SOFTWARE QUALITY (QA)

[AFTE80]
[ALFO77]
[ANDE78]
[ARNO83]
[BOEH78]

SOFTWARE QUALITY (QA)

[MCCA83]
[MILL83]
[NBS74]
[NBS129]
[OSBO84]

[CANN78b]
[CENT82]
[COOP79]
[DUNN82]
[FCSC83]
[GLAS82]
[HALL78]
[JONE78]
[LAFF77]
[LIEN80b]
[LIEN81]
[MCCA77]

<u>TECHNIQUES</u>

[ANDE81]
[ARAN86]
[BASI75]
[BASI82]
[BEEL83]
[BRIT86]
[CHAP79]
[DONA80a]
[DONA80b]
[EBER80]
[GILB79a]
[KERN78]

<u>TESTING</u>

[AFTE80]
[BERL83]
[BOSS82]
[CANN78a]
[DEAN82]
[DONA80a]
[DUNN82]
[EBER80]
[ENDR75]
[FISC77]
[GLAS81a]
[HALL78]
[HUAN75]
[LIPO79]
[LOVE77]

<u>TOOLS</u>

[ANDE78]

[OSBO86]
[PEER81]
[PEER84]
[PRES81]
[RAMA84]
[RYAN82]
[SHNE80]
[SHOO83]
[TAUT83]
[TINN84]
[WALT78]

<u>TECHNIQUES</u>

[LETO86]
[MARS83a]
[OSBO86]
[PEER81]
[PEER84]
[PIZZ84]
[ROSS75]
[SARS77]
[SHNE86]
[YAU78]
[YAU79]
[YAU80b]

<u>TESTING</u>

[MCCA76]
[MYER78]
[MYER79]
[NBS98]
[NBS98]
[PARI80]
[PIZZ84]
[PRES82]
[RAMA84]
[SCHN79]
[SHOO83]
[SILV83]
[SNEE84]
[WALT78]
[WASS78]

<u>TOOLS</u>

[LIEN83]

[BERN84]
[BRAN77]
[BRIC81]
[BRIC84]
[CANN72]
[CLAP81]
[COWE79]
[DEAN82]
[DONA80a]
[DONA80b]
[DUNN81]
[DUNN84a]
[ER84]
[FCSC83]
[FIPS106]
[FORN83a]
[FORN83b]
[FORN83c]
[GAO80]
[GILB79b]
[GLAS79]
[GLAS81b]
[SHNE86]
[STAN77]

<u>USERS/PROGRAMMERS</u>

[ARNO86]
[BALL85]
[BELL83]
[BENS78]
[BRON81]
[CANN81]
[CHAP78]
[CLAR80]
[COUG83]
[COUG84]
[FEIN84b]
[FERR81]
[FINK77]
[FORN83b]
[GROS82]
[HENK81]
[HOUT83]
[HOWA83a]
[HOWA83b]
[HULI84]
[KAPU83]
[KULL83]
[LETO86]
[LIEN79]

[LIPO79]
[LYON81]
[MART83]
[MCCO84]
[MILS81]
[MYER79]
[NBS74]
[NBS98]
[NBS106]
[NBS129]
[OSBO84]
[OSBO86]
[PARI80]
[PARI82]
[PARI83a]
[PARI83b]
[PEER84]
[PERL81]
[RAYN83]
[RICH84]
[ROSS75]
[SHAR77]
[SNEE84]
[ZIRK78]

<u>USERS/PROGRAMMERS</u>

[MCGR73]
[MIAR83]
[MOON75]
[NBS130]
[ODGI72]
[OVER73]
[OVER74]
[PARI80]
[PARI82]
[PAR183a]
[PERL81]
[PETE84]
[PODO77]
[PUNT75]
[RAY83]
[RAYN83]
[REUT81]
[SCHN83]
[SHEP77]
[SHEP79]
[SING80]
[SOLO84]
[TANN80]
[THAY80]

[LIEN83]	[TIPS80]
[MANC83]	[WEIN71]
[MANT81]	[WEIS82]
[MART84]	[ZELK78]
[MAUL83]	[ZUCK83]
[MCCA83]	[ZVEG82]

SOME ADDITIONAL MAINTENANCE TOPICS

[BOEH81a]	[MART79]
[BRAN75]	[MURR83]
[BRAV76]	[NARR84]
[CASH80]	[PARN72]
[CONN80]	[PARN79]
[DALY77]	[PETE77]
[DUNS80]	[RICH83]
[GILB77]	[SCHN82c]
[GILH82]	[SHIF78]
[KAPL77]	[SPIE76]
[KHAN75]	[STAN77]
[LEHM76]	[SWAN84]
[LIEN80a]	[THAY81]
[MARS83b]	[WINO79]

****

[AFTE80] Software OT&E Guidelines, Volume III, Software: Software Maintainability Evaluator's Handbook, Air Force Test and Evaluation Center (AFTEC), RADC, Griffiss AFB, NY, vol. III, April 1980.

Software OT&E Guidelines is a set of handbooks prepared by the Computer/Support Systems Division of the Test and Evaluation Directorate of the Air Force Test and Evaluation Center (AFTEC) for use in the operational test and evaluation of software. Volumes in the set include: the Software Test Manager's Handbook (AFTECP 800-1); the Handbook for the Deputy for Software Evaluation (AFTECP 800-2); the Software Maintainability Evaluator's Handbook (AFTECP 800-3); the Software Operator-Machine Interface Evaluator's Handbook (AFTECP 800-4); and the Software Support Facility Evaluation Tools User's Handbook (AFTECP 800-5). The Software Maintainability Evaluator's Handbook is a guide for the software evaluator participating in AFTEC`s software maintainability evaluation process. It provides a methodology to enable evaluation team members to independently evaluate software maintainability. The handbook also includes examples, explanations, definitions, and a set of characteristics used to rate the software.

Key words: operational, quality factors, software evaluation, software maintenance, software testing.

****

[AGRE82a] W.W. Agresti, "Managing Program Maintenance", Journal of Systems Management, vol. 33, pp. 34-37, February 1982.

This article defines 'software maintenance'; identifies three categories of maintenance (perfective, corrective, and adaptive); describes software maintenance problems; and presents ideas for improving software maintenance management. The high turnover rate among programmers and the use of entry-level versus senior level programmers are also discussed. The author recommends that:

o software maintenance should be managed directly;

o more creative and innovative approaches for managing and organizing software maintenance should be utilized;

o software changes should be scheduled and documented; and

o the user should perform some of the software maintenance.

Key words: documentation, management, software maintenance, software management, turnover.

****

[AGRE82b] W.W. Agresti, "Measuring Program Maintainability", _Journal of Systems Management_, vol. 33, pp. 26-29, March 1982.

This paper discusses metrics and methods for measuring a program's maintainability. It also describes how metrics can be used to estimate the resources needed to maintain a program. According to the author, the use of metrics helps place the responsibility of building maintainable software on the developers, and helps focus attention on those aspects of software design that facilitate software maintenance. A questionnaire for calculating the maintainability metric of a program is provided.

Key words: maintainability, metrics, software maintenance, software maintenance management.

****

[ALFO77] M.W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements", _IEEE Transactions on Software Engineering_, IEEE Computer Society Press, vol. SE-3, no. 1, pp. 60-69, January 1977.

This paper presents a methodology for the generation of software requirements for large, real-time, unmanned weapons systems. It describes what needs to be done, how to evaluate the intermediate products, and how to use automated aids to improve product quality. An example of the methodology, the resultant products and the benefits provided by using this methodology and other experimental applications results are presented.

Key words: methodology, real-time, software engineering, Software Requirements Engineering Methodology, Software Requirements Engineering Program.

****

[ANDE78] P.G. Anderson, "Redundancy Techniques for Software Quality", _Proceedings of the Annual Reliability and Maintainability Symposium_, IEEE Computer Society Press, pp. 86-93, January 1978.

This article describes and compares the use of redundancy in both hardware and software systems. The author demonstrates the use of redundancy for improving the quality of software, specifically in the areas of program and data structures, programming language design, and programming team utilization. The author recommends that software engineers adopt the tools and techniques of hardware engineers to produce high quality software designs.

Key words: maintainability, program structures, language, design, programming team, reliability, software maintenance, software quality, tools.

****

[ANDE81] R.E. Anderson, "Modular documentation: a software development tool", FIPS 1981 National Computer Conference Proceedings, pp. 401-405, May 4-7, 1981.

This paper presents a method for documenting the design and implementation of a large software system. The method is presented in terms of a family of documents based on the decomposition of any system into specific levels of abstraction for the purpose of software development. This approach facilitates the use of structured design techniques, provides tangible objects for organizing manpower resources on the basis of the system's structure, aids in the generation of more meaningful milestones, and results in a fully, documented system when the implementation phase is complete.

Key words: documentation, management, design, development, implementation, maintenance, design techniques.

****

[ARAN86] G. Arango, I. Baxter, P. Freeman, and C. Pidgeon, "TMM: Software Maintenance by Transformation", IEEE Software, vol. 3, no. 3, pp. 27-39, May 1986.

Porting an undocumented program without any source changes demonstrates the value of a transformational theory of maintenance. The theory is based on the reuse of knowledge. As needs change, software must be amended, or maintained, to adapt to the new environment. Often, such adaptation involves porting programs from one machine to another. If there is no information about original design decisions or abstractions, the software becomes obsolete, and the enormous resources invested in its construction are lost.

This paper proposes a method that will allow practitioners to recover abstractions and design decisions that were made during implementation.

Key words: design, Draco, paradigm, programming languages, software maintenance, techniques.

****

[ARNO83] R.S. Arnold, "On the Generation and Use of Quantitative Criteria for Assessing Software Maintenance Quality", University of Maryland Department of Computer Science, PhD Dissertation, February 16, 1983.

Whenever software metrics are used in assessing the quality of software maintenance their values must be interpreted relative to the metrics' intended use. Further, these interpretations should lead to concrete suggestions for improving software maintenance quality. This dissertation presents a method known as Criteria Application (CA) for performing these interpretations and improvements.

The principle feature of this method is the development of criteria for modeling software/software maintenance quality concepts. A criterion is a testable condition, concerning software metrics values, with an explicit judgment attached to the condition's outcome. The criteria are used in (1) establishing goals for software maintenance, (2) quantitatively determining whether the goals are met, and (3) guiding remedial work if the goals are not met. To illustrate its usefulness, CA is applied to some actual software maintenance data, from the maintenance of scientific application software. The author also shows how CA can be applied beyond software software maintenance to software development.

Key words: criteria application, software development, software maintenance, software metrics, software quality,

****

[ARNO86] R.S. Arnold, <u>Tutorial on Software Restructuring</u>, IEEE Computer Society Press, February 1986.

This Tutorial addresses restoring software structure whether the lack of structure results from software maintenance or from software development. With continued change, programs tend to become less "structured." This is manifested by out-of-date documentation, by code that does not conform to standards, by increased time for programmers to understand code, by increased ripple effect of changes, and so on. These can - and usually do - imply higher software maintenance costs.

Software restructuring is an important option for putting high software maintenance costs under control. The idea is to modify software - or programmer's perceptions of software structure - so one can understand it and control it anew.

There are many reasons why software managers and programmers should be aware of software restructuring. This Tutorial addresses:

1. Implementing standards for software structure
2. Regaining understanding of software by installing software with known, easily traceable structure
3. Extending system lifetime by retaining a system's flexibility through good structure
4. Preparing software for conversion.

Key words: change, conversion, programmers, restructuring, software maintenance cost, software managers, standards, techniques.

****

[ARTH83] L.J. Arthur, <u>Programmer Productivity</u>, John Wiley and Sons, 1983.

This book discusses software metrics for measuring quality, achieving reliability and maintainability in programs, and acquiring software to improve productivity. The author emphasizes the need to develop an environment and standards to increase productivity. This new environment is referred to as a 'software factory'. A number of methods that can be used to identify software maintenance-prone software are presented.

Key words: environment, maintainability, productivity, software factory, software maintenance, software measurement, metrics, standards.

****

[BALL85] R.K. Ball, "Managing the Transition From Development to support", <u>Data Management</u>, pp. 34-37, March 1985.

Problems often arise in DP organizations when a new system is transferred from the development to the software support group. The pressures on systems software maintenance managers to declare the system operational, and the high expectations of the users often compound the problems. A software maintenance manager has a number of options for handling this situation:

1. Do nothing - depend on development to produce a good product.
2. Use the 'escort approach' - a member of the development team should accompany the system into support.
3. Use the systems acceptance approach.

The author discusses each of these options and recommends option 2 which has built in safeguards that hold the development team accountable to the future 'users' of the system. The objectives of this approach are to: ensure that systems design considers future support requirements; verify the completeness and quality of the system when passed on to support; ensure that the support staff are adequately trained and equipped to support the system; and to manage the transition. Procedures for accomplishing these objectives are presented.

Key words: development, management, software maintenance, acceptance approach.

****

[BANN83] K. Banni, M. Suzuki, and T. Terano, "A Total Approach to a Solution for the Maintenance Problems Through Configuration Management - Maintenance Support Facility MSF", COMPSAC Proceedings, IEEE Computer Society Press, pp. 404-411, November 7-11, 1983.

This paper describes the solution of a software maintenance problem in an electric power company. The authors developed a system called DNUS which supports software maintenance tasks through system configuration management. The paper provides a detailed explanation of the components and behavior of DNUS, and summarizes the effects of DNUS on the software maintenance effort.

Key words: configuration management, DNUS, management, software maintenance.

****

[BASI75] V.R. Basili and A.J. Turner, "Iterative Enhancement: A Practical Technique for Software Development", IEEE Transactions on Software Engineering, IEEE Computer Society, vol. SE-1, no. 4, pp. 390-396, December 1975.

This paper recommends the "iterative enhancement" technique as a practical means of using a top-down, stepwise refinement approach to software development. Initially, a properly chosen (skeletal) subproject is implemented. This is followed by the gradual enhancement of successive implementations in order to build the full implementation. The development and quantitative analysis of a production compiler for the language SIMPL-T is used to demonstrate that the application of iterative enhancement to software development is practical, efficient, and encourages the generation of an easily modifiable product, and facilitates reliability.

Key words: enhancement, SIMPL-T, software development, software evaluation measures, top-down design.

****

[BASI80] V.R. Basili, Tutorial: Models and Metrics for Software Management and Engineering, IEEE Computer Society Press, September 1980.

This tutorial presents a new, quantitative approach to software management and software engineering that has taken shape over the past few years. This quantitative methodology is needed to understand, evaluate, control, and estimate software development and maintenance and to make the necessary cost, time, and quality trade-offs. The tutorial will focus on attributes that can be managed quantitatively, including product-oriented attributes such as size, complexity, reliability, and change, and process-oriented attributes such as cost, schedules, and resources.

Key words: change, complexity, cost, measures, metrics, models, reliability, resources, schedules.

****

[BASI82] V.R. Basili and H.D. Mills, "Understanding and Documenting Programs", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-8, no. 3, pp. 270-283, May 1982.

This paper reports on an experiment that tests the difficulty of understanding an unfamiliar program. The goal was to stimulate a practicing programmer in a software maintenance environment using the techniques of program design adapted to program understanding and documentation; that is, given a program, a specification and proof of correctness were developed for the program. The approach points out the value of proof of correctness ideas in guiding the discovery process. Toward this end, a variety of techniques were used: direct cognition for smaller parts, discovering and verifying loop invariants for larger program parts, and functions determined by additional analysis for larger program parts. An indeterminate bounded variable was introduced into the program documentation to summarize the effect of several program variables and simplify the proof of correctness.

Key words: program analysis, program correctness, documentation, proof techniques, software maintenance.

****

[BASI84] V.R. Basili and B.T. Perricone, "Software errors and Complexity: an Empirical Investigation", Communications of the ACM, ACM Press, vol. 27, no. 1, pp. 42-52, January 1984.

An analysis of the distributions and relationships derived from the change data collected during development of a medium-scale software project produces some surprising insights into the factors influencing software development. Among these are the trade-offs between modifying an existing module as opposed to creating a new one, and the relationship between module size and error proneness.

Key words: complexity metrics, error analysis, measurement, reliability, software errors.

****

[BASS84] P. Bassett, "Software Manufacturing Techniques and Maintenance", _AFIPS 1984 National Computer Conference Proceedings_, AFIPS, vol. 53, pp. 357-365, May 1984.

A solution to the reusable code problem can also provide a solid technical basis from which to understand and deal with the production, quality, and software maintenance issues of the software industry. A software manufacturing methodology called Computer-Aided Programming (CAP) has been developed. CAP is based on a functional programming concept called a frame. Frames, which were originally developed as a means of resolving software maintenance problems associated with reusable code, are presented here as an alternative design methodology. Statistics from a case study of a project that employed CAP indicate that: (1) production-quality commercial software can be manufactured at rates exceeding 2000 lines of debugged COBOL per man-day (including systems design time), and (2) less than 10% of this code needs to be hand-written or maintained.

Key words: computer-aided programming, frames, productivity, reusable, software maintenance.

****

[BEEL83] J. Beeler, "Manager Cuts Maintenance Time by 70%", _Computerworld_, pp. 7-8, January 31, 1983.

This article discusses how one manager reduced the heavy program software maintenance load in his organization through structured design and programming techniques. The systems design and programming process were divided into several steps. After each step, the work of each employee was submitted to a structured walkthrough. The benefits of using the structured methodology are discussed.

Key words: programming techniques, software maintenance, structured design, structured walkthrough.

****

[BELL83] D. Bellin, "After the Purchase: Software Maintenance and Outside Support", Software Review, vol. 2, no. 2, pp. 69-72, June 1983.

This article provides information for users of microcomputer systems about the problems of software maintenance. Recommendations on how to handle problems such as training, program enhancement, and modification, communication with other computers, and system expansion are addressed. Emphasis is placed on the need to plan thoroughly and be aware of the needs and costs for ongoing software support. The article provides useful information for organizations planning to purchase microcomputers.

Key words: microcomputer, planning, enhancement, software maintenance, software support.

****

[BELL84] F.J. Bell, "Technology Transfer in the Maintenance Environment", AFIPS 1984 National Computer Conference Proceedings, AFIPS, vol. 53, pp. 229-234, May 1984.

This paper reports on the software maintenance productivity project at The Equitable Life Assurance Society of the United States. In 1982, this firm established a software maintenance productivity project (MPP). Three potential areas were identified for technology transfer: the software maintenance function, the software maintenance environment, and software maintenance metrics. The programs instituted include a software maintenance management handbook, a software maintenance managers' round table, and greater vendor, user, management, and programmer involvement during the development of the integrated software maintenance environment. As a result of this emphasis, software maintenance has become an established and recognized area of specialization at Equitable.

Key words: environments, management, metrics, productivity, programming, software maintenance.

****

[BENS78] M. Benson, "Responsibility vs. Authority, Lying to Management: A Legitimate Solution?", Computerworld, pp. 30-31, September 11, 1978.

This paper describes a confrontation between a very talented software maintenance programmer and an incompetent DP manager. The programmer had been working without much direction or management for five years. He had revived and maintained programs that were literally on the 'scrap heap'. The new manager now wanted reports on the programmer's activities in two categories, changes and fixes, but not on cleanup since the manager felt that was a waste of time. The programmer strongly disagreed. The author's question is whether the programmer should listen to management or continue to include his cleanup activities as changes or fixes. This is a very controversial issue which attempts to establish responsibility for the quality of the software. This article provides an interesting study in the management of talented software maintenance programmers.

Key words: management, personnel, programming, software maintenance management.

****

[BERL83] T.H. Berliner and T.J. DeGarmo, "Test Drive Your Software", Computer Decisions, pp. 160-166, September 1983.

According to the authors, proper software testing provides a meaningful, quantitative basis for ensuring that desired features have been included, and that they will perform as required. The procedures for software testing, planning, test designing, and preparation of test data, are detailed.

Key words: planning, software maintenance, software testing, test designing.

****

[BERN84] G.M. Berns, "New Life for Fortran", Datamation, pp. 166-174, September 1, 1984.

The Maintainability Analysis Tool (MAT) is a diagnostic, documentation program that analyzes FORTRAN source program modules. (i.e., subroutines or functions). It runs on a DEC VAX-11 computer system under the VAX/VMS operating system but can also process modules written for any computer in the FORTRAN 66 or 77 dialects. MAT can analyze existing programs, as well as those currently under development. Problems that can be detected by MAT include: module internal discrepancies, module clutter or unused code, module interface discrepancies, and discrepancies in program structures. In addition, MAT assists programmers by documenting the module's interface and the program's structure. The author suggests that MAT can improve the reliability and maintainability of old and new Fortran programs through the use of an index technique.

Key words: documentation, FORTRAN, maintainability Analysis Tool, programming, reliability, software maintenance, tools.

****

[BERS79] E.H. Bersoff, V.D. Henderson, and S.G. Siegel, "Software Configuration Management: A Tutorial", Computer, pp. 6-14, January 1979.

Software configuration management (SCM) is the discipline of identifying the configuration (or arrangement) of a system at discrete points in time for the purpose of systematically controlling changes to this configuration and maintaining the integrity and traceability of the configuration throughout the system lifecycle. The primary objective of SCM is the effective management of a software system's lifecycle and its evolving configuration. This paper identifies the need for a SCM methodology and describes SCM concepts and guidelines. The four component elements of configuration management are defined as follows:

- o Identification - the items in the system configuration are distinguished (identified).
- o Control - managing changes to the system.
- o Status accounting - provides a history of changes made to the system; describes how the system evolved.
- o Auditing - answers the question: 'Does the system I am building satisfy the stated needs?'.

Key words: software configuration management (SCM), software lifecycle, software maintenance.

****

[BERS80] E.H. Bersoff, V.D. Henderson, and S.G. Siegel, Software Configuration Management-An Investment in Product Integrity, Prentice-Hall, 1980.

A detailed discussion of software configuration management (SCM) is presented. The authors:

- o explain the basic elements of the SCM discipline;
- o show how the application of these elements to the software development cycle facilitates the transformation of software into a visible, manageable entity;
- o demonstrate how this transformation is a key ingredient to achieving software and system product integrity.

Key words: product integrity, software configuration management (SCM), software development, software maintenance.

****

[BOEH78] B.W. Boehm, J.R. Brown, H. Kasper, M. Lipow, G.J. MacLeod, and M.J. Merrit, Characteristics of Software Quality, North-Holland, 1978.

The study described in this book establishes a conceptual framework for achieving software quality. The key points are:

- o Explicit attention to characteristics of software quality can lead to significant savings in software lifecycle costs.
- o The current software state-of-the-art imposes specific limitations on the ability to automatically and quantitatively evaluate the quality of software.
- o A definitive hierarchy of well-defined, well-differentiated characteristics of software quality is developed. Its higher-level structure reflects the actual uses to which software quality evaluation would be put; its lower-level characteristics are closely correlated with actual software metric evaluations which can be performed.
- o A large number of software quality evaluation metrics have been defined, classified, and evaluated with respect to their potential benefits, quantifiability, and ease of automation.
- o Particular software lifecycle activities have been identified which have significant leverage on software quality.

Key words: software lifecycle, software maintenance, software metrics, software quality.

****

[BOEH81a] B.W. Boehm, "An Experiment in Small-Scale Application Software Engineering", IEEE Transactions on Software Engineering, vol. SE-7, no. 5, pp. 482-493, September 1981.

This paper reports on two small interactive application software products that were developed to the same specification, one using Fortran and one using Pascal. Several hypotheses were tested, and extensive experimental data collected. The major conclusions were as follows:

- o Large project software engineering procedures can be cost effectively tailored to small projects
- o The choice of programming language is not the dominant factor in small application software product development
- o Programming is not the dominant activity in small software product development

- o Most of the coding on a small application software product is devoted to "housekeeping".

Experimental data supporting these conclusions, their context and implications is presented.

Key words: programming, languages, software engineering, software management.

****

[BOEH81b] B.W. Boehm, Software Engineering Economics, Prentice-Hall, 1981.

This book presents economic concepts and techniques in software engineering. It provides a framework of software engineering goals, and discusses a quantitative model of the software lifecycle, the fundamentals of software engineering economics as they apply to software projects, the detailed techniques for software lifecycle cost estimation which support the software engineering economic analysis techniques.

Key words: cost estimation, software engineering economics, software lifecycle, software maintenance.

****

[BOSS82] R.W. Boss, "Software Documentation", Software Review, vol. 1, no. 2, pp. 165-167, October 1982.

The detailed record of the development of a computer program, including its modification and testing histories (i.e., software documentation) is a necessary peripheral in the development, evaluation, purchase and ongoing use of custom or prewritten software. The author defines the various components of documentation and describes its functions and usefulness for both the vendor and the user.

Key words: documentation, software design, software development, software packages, software maintenance, testing.

****

[BRAN75] C.L. Brantley and Y.R. Osajima, "Continuing Development of Centrally Developed and Maintained Software Systems", COMPCON Proceedings, IEEE Computer Society Press, pp. 285-288, February 25-27, 1975.

A field evaluation of DIR/ECT, a man/machine white pages telephone book production system is presented. It describes the long term requirements for balancing an overall program of software maintenance and continuing development of large scale software systems. The latter is emphasized.

Key words: DIR/ECT, redesign, requirements, software development, software maintenance.

****

[BRAN77] W.E. Branning, J.P. Schaenzer, D.M. Willson, and W.A. Erickson, "Modern Programming Practices Study Report", Rome Air Development Center, RADC, Griffiss AFB, N.Y., April 1977.

This report on the technical study of programming practices shows that:

- o Top-down program development reduced cost by 15%,
- o Regardless of the method for development used labor was generally distributed as follows:
  - . 10% for analysis
  - . 30% for design
  - . 35% for coding and debugging
  - . 25% for testing
- o Program testing was automated using real-time on line simulation, scenario control, data recording and reduction.
- o Through system design, operating system software provided real-time, on-line recovery of system functions from hardware failures.
- o Software tools found effective on the surveyed programs were described and evaluated.

Key words: AN/UYK-7 computer, modern programming practices, real-time, command and control systems, software tools, top-down program development.

****

[BRAV76] P.H. Braverman, "Managing Change", <u>Datamation</u>, pp. 111-113, October 1976.

Guidelines for managing changes made to projects are provided. The author defines change as: 'any event, action, or edict which may affect the scope of a project, the schedule of a project or the resources planned for the project.' A description of the internal and external types of changes is provided. Project change planning and control procedures are discussed. The need to quantify and document project assumptions, and management and user expectations is stressed. According to the author, a system for managing change is a "must" for project management.

Key words: changes, development, management, project management, software maintenance.

****

[BRIC81] L. Brice, "Existing Computer Applications Maintain or Redesign: How to Decide?", Computer Measurement Group International Conference Proceedings, IEEE Computer Society Press, pp. 20-28, December 1-4, 1981.

Maintenance of large applications programs is an aspect of performance management that has been largely ignored by those studies that attempt to bring structure to the software production environment. Maintenance refers to fixing "bugs", modifying current design features, adding enhancements, and porting applications to other computer systems. It is often difficult to decide whether to maintain or redesign. One reason for the difficulty is that good models and methods do not exist for differentiating between those programs that should be maintained and those that should be redesigned.

This paper presents a case study of a large application software maintenance effort which was monitored. The information obtained provided significant insight into factors to be considered when considering whether to redesign or continue to maintain. The study found that much of the data that is necesary to establish accurate predictions on redesign vs maintain is difficult to obtain. Suggestions for collecting and measuring these and other types of data are presented. Some of the tools that can be used for the collection and measurement of performance data are also highlighted.

Key words: management, measurement, programs, redesign, software evaluation, software maintenance, tools.

****

[BRIC84] L. Brice and J. Connell, "System Information Database: An Automated Maintenance Aid", AFIPS 1984 National Computer Conference Proceedings, vol. 53, pp. 209-216, May 1984.

This paper discusses a case study of how one data processing organization applied student labor and a relational database management system in a prototype to automate much of their applications systems documentation function. Documenting application systems has long been considered a necessary evil. Necessary because documentation provides a map to present systems, serves as a software maintenance aid, and is required by the auditors; evil because it is an activity generally dreaded by those who develop the systems. Since normal behavior regarding unpleasant chores is avoidance, application systems documentation is sometimes absent and often incomplete. Benefits of automated documentation are presented.

Key words: automation, database management system, documentation environments, software maintenance, tools.

****

[BRIT86] R.N. Britcher, J.J. Craig, "Using Modern Design Practices to Upgrade Aging Software Systems", IEEE Software, vol. 3, no. 3, pp. 16-24, May 1986.

Modifying 100,000 lines of 20-year-old code, IBM-FSD successfully applied its software engineering principles to modernization of the FAA's air traffic control system. One of the major challenges facing software system managers in the 1980's is how to upgrade large, complex, embedded systems, written a decade or more ago in unstructured languages according to designs that make modification difficult. This paper describes a plan to modernize its entire National Airspace System, including towers, approach controls, and "en route" centers.

Key words: design, software maintenance, software, techniques.

****

[BRON81] G.M. Bronstein and R. Okamoto, "I'm OK, You're OK, Maintenance is OK", Computerworld, pp. 21-24, January 12, 1981.

Industry attitudes towards software maintenance programmers are examined in this article. Many of the managerial problems connected with software maintenance programming are due to preconceived notions and attitudes towards software maintenance programming. New perceptions of software maintenance programming (i.e. software maintenance programming is a rewarding and challenging career requiring considerable skill and experience) must replace the erroneous, traditional perceptions that were previously used in the organization. Recommendations are provided for addressing software maintenance personnel problems. The author suggests that one approach for improving the software maintenance situation is to develop an environment within the organization where it is 'OK' to be a software maintenance programmer and where they are both rewarded and recognized for their accomplishments.

Key words: environments, programmers, attitudes, programming, software maintenance.

****

[BROO82a] F.B. Brooks Jr., The Mythical Man-Month, Addison-Wesley publishing Company, 1982.

The author presents numerous insights into software engineering management, drawn from practical experiences gained while implementing OS/360. The problems of large and small programming projects are addressed. The author states that large programming projects suffer management problems different in kind from small ones, due to the division of labor. The critical need is seen as the preservation of the conceptual integrity of the product. The difficulties of achieving this unity and methods for doing so are explored. System debugging, component debugging, and methods of design that reduce the number of bugs in a program are addressed.

Key words: debugging, design, OS/360, management, personnel, software engineering, software maintenance, system debugging.

****

[BROO82b] R. Brooks, "A Theoretical Analysis of the Role of Documentation in the Comprehension of Computer Programs", _Proceedings of the Human Factors in Computer Systems_, ACM Press, pp. 125-129, March 15-17, 1982.

The use of documentation is critical to the success of the software maintenance process. Documentation facilitates the understanding of a computer program listing, an important prerequisite to updating or enhancing a program. This article discusses a theory useful for understanding programs and provides inferences and predictions derived from this theory. These include the following:

- o The role of documentation devices, such as flow charts or program design languages, will be particularly effective if the information they provide is useful in constructing and confirming a hypothesis.
- o It may be as important to adequately document the problem as it is to document the program.
- o Documentation must be matched to particular programming environments. For example, different programming languages may need different kinds of documentation for different tasks.
- o Multiple forms of documentation for the same program will be beneficial when the different forms convey different kinds of information."

Key words: beacons, documentation, flow-charts, program design language, (PDL), programming environments, software maintenance.

****

[BRYA84] W. Bryan and S. Siegel, "Making Software Visible, Operational, and Maintainable in a Small Project Environment", _IEEE Transactions on Software Engineering_, IEEE Computer Society Press, vol. SE-10, no. 1, pp. 59-67, January 1984.

This paper discusses suggested software management practices resulting from the experiences of a small (approximately 100 employees) software engineering company that develops and maintains computer systems supporting real-time, interactive, commercial, industrial, and military applications. Practical suggestions are presented for effectively managing software development in project environments budgeted at no more than several million dollars per year. The suggestions are based on the findings of a product assurance group that is independent from the development group. Within this check-and-balance management/development/product assurance structure, a design review process is described that effects an orderly transition from customer needs statement to software code. Activities of a change control body (CCB) and supporting functions geared to maintaining delivered software also are described.

Key words: configuration control board, configuration management, design review, product assurance, project management, software development, software maintenance, testing.

****

[CANN72] R.G. Canning (editor), "That Maintenance Iceberg", EDP Analyzer, pp. 1-14, October 1972.

Case studies of how two companies handle the software maintenance function are presented. A plan is described for making application systems more maintainable. The author defines 'more maintainable' systems as those systems that have been developed according to policies and procedures that will both reduce the need for software maintenance as well as make any necessary software maintenance easier to perform. The six aspects of this plan are: designing for change by establishing standards, designing changes with the aid of tools, making use of configuration policies, control and audit, organizing for software maintenance, and converting to more maintainable systems.

Key words: configuration management, development, maintainability, management, software maintenance, standards, tools.

****

[CANN78a] R.G. Canning (editor), "Progress in Software Engineering: Part 1", EDP Analyzer, pp. 1-13, February 1978.

An analysis of the concepts, features, and scope of software engineering, including technological and managerial aspects is provided. Improved design methodologies and the use of tools to reduce the time spent on software maintenance are presented. The following steps are recommended:

- o Begin to develop and classify a list of errors found in requirement statements.
- o Get user involvement.
- o Select an approach for handling complexity (i.e., functional decomposition or information flow

analysis).
- o Use an inspection process.
- o Define expected performance.

Key words: design methodologies, management, quality assurance, requirements, software engineering, testing, tools.

****

[CANN78b] R.G. Canning (editor), "Progress in Software Engineering: Part 2", EDP Analyzer, pp. 1-13, March 1978.

This report discusses some tools and techniques for managing the creation and maintenance of software. An overview of the methods used to evaluate the quality and performance of software is given. Descriptions of some of the methods used to evaluate the quality and performance of the software development staff, the products, and the ability of users, are contained in this article. The author discusses software engineering management which provides a methodology for improving the management of software development and modification. Three areas of software engineering management: staff behavior patterns, project behavior patterns, and system behavior patterns are examined.

Key words: adaptive maintenance, configuration management, corrective management, personnel, program evolution, software engineering, software quality, software maintenance.

****

[CANN81] R.G. Canning (editor), "Easing the Software Maintenance Burden", EDP Analyzer, pp. 1-14, August 1981.

Case studies of three organizations involved in software maintenance are described. Techniques for lowering software maintenance costs are discussed. Each organization uses different methods for improving the maintainability of their systems. The author discusses how the methods of performing software maintenance changed during the 1970s. Techniques such as, fine tuning the organization, fine tuning software maintenance procedures, programming for maintainability, and off-loading software maintenance work onto others (i.e., users or package vendors) are recommended for improving software maintenance.

Key words: adaptive software maintenance, corrective maintenance, enhancements, maintainability, software maintenance organization, users.

****

[CASH80] P.M. Cashman and A.W. Holt, "A Communication-Oriented Approach to Structuring the Software Maintenance Environment", _Software Engineering Notes_, pp. 4-17, January 1980.

This article describes the communication oriented approach to handle software maintenance. This approach emphasizes the need to support the software maintenance process rather than emphasizing the capabilities, tools, or static structures. This approach helps to improve the software maintenance environment so that it reflects the communication problems between different agents (either people or programs) involved in the process of software maintenance. It provides an environment in which the software maintenance process can be easily monitored by managers.

MONSTR (MONitor for Software Trouble Reporting), a software maintenance system designed using the communication-oriented approach, is also described. MONSTR is a protocol-driven system that is supplied with a description of the communication paths (protocols) which may be used by project members during the process of software maintenance.

Key words: communication, MONSTR, software maintenance, trouble reports.

****

[CENT82] J.W. Center, "A Quality Assurance Program for Software Maintenance", _AFIPS 1982 National Computer Conference Proceedings_, pp. 399-408, June 7-10, 1982.

An implementation of a quality assurance (QA) program applied to software maintenance projects is described. The relationship between the QA program and project management is identified. The paper includes a brief discussion of waivers and deviations to standards and control documents. The QA checks are delegated to three levels: rationale, scope, and authority. A list of sample criteria used for each QA check or inspection is provided.

Key words: maintenance projects, management, project management, quality assurance, software maintenance.

****

[CHAP78] N. Chapin, "Semi-Code in Design and Maintenance", _Computers and People_, pp. 17-27, June 1978.

This paper discusses semi-code, which is a narrative, prose-like technique that describes the design of a system in narrative form instead of using graphs or flowcharts. It fits well with the stepwise, incremental, or iterative refinement techniques that are growing in popularity with designing and maintaining programs and systems. Semi-code is referred to by many names including structured english, pseudo-code, program design language, system development language and structured text.

Some of the major characteristics of semi-code are: (1) it narrates the description of the program or system in terms of a sequence of functions; (2) it is consistent with the disciplines of structured programming and structured design; (3) it imposes no restrictions on vocabulary at the start; and (4) it organizes the description into four major parts: an identifying name, a list of the input data, a list of the output data, and a set of statements for the function.

Key words: design, documentation, program design language(PDL), programmers, pseudo-code, semi-code, software maintenance.

****

[CHAP79] N. Chapin, "Some Structured Analysis Techniques", Database, pp. 16-23, Winter 1979.

Techniques for using structured analysis which is patterned after the discipline of structured programming are discussed. These techniques also focus on formal communications using graphic techniques which include the: Data Flow Diagram (DFD) or bubble chart, Compound System Chart (CSC), and actigram/datagram.

- o The DFD or bubble chart represents the logical flow of data,
- o The CSC represents the logical or physical flow of data at the users option, and
- o The actigram/datagram models a system, emphasizing the constraints.

The four areas to which these techniques are applied are: organization, data and data flow, processes, and use. In addition, two versions of the DFD, one advocated by DeMarco and Rose and the other advocated by Gane and Sarson, are included in the comparison.

Key words: actigram/datagram, bubble chart, compound system chart(CSC), data flow diagram(DFD), graphics, structured analysis techniques.

****

[CHAP84] N. Chapin, "Software Maintenance With Fourth Generation Languages", ACM SIGSOFT Software Engineering Notes, vol. 9, no. 1, pp. 41-42, January 1984.

The impact of fourth-generation languages on software maintenance is discussed. The advantages and disadvantages of both third-generation and fourth-generation languages are discussed. The author suggests that the use of fourth-generation languages in the development stage could result in increases in both the cost and difficulty of software maintenance.

Key words: fourth generation languages, software maintenance costs, program languages, software maintenance, third generation languages.

****

[CLAP81] J. Clapp, "Designing Software for Maintainability", Computer Design, pp. 197-207, September 1981.

Factors that contribute to the cost of maintaining software, and differences between software maintenance and software development are presented. The article references an IBM report which addresses the effects of software maintenance on a large system (IBM's OS/360) and the three laws of "Program Evolution Dynamics". The author identifies factors to be considered when planning, designing, documenting, and managing software changes. Four trends in computer technology that could affect the future of software maintenance are discussed. These are: increased availability of tools, quantitative measures of maintainability, monitoring the software maintenance process, and the use of higher level software.

Key words: design, evolution, maintainability, metrics, software maintenance costs, tools.

****

[CLAR80] D.M. Clark, "Maintenance Programming", Computerworld, pp. 27-30, July 28, 1980.

This article discusses a methodology for making changes to an existing program. The methodology emphasizes change preparation, since it takes more time to determine what to change in a program than to make the actual change. The following guidance is provided:

- Determine approximately how long the task should take, how important it is.
- Read and study the change instructions from the user; contact the user for clarifications. This is an important step because most user-related problems are not technical problems, but communication problems.
- Acquire an overview of the program and determine the program's purpose, framework, and output.
- Ignore that part of the program that will not be affected by the change.

- o Document your changes as well as any information which you have learned about the program's logic. Placing the documentation and a reference to the changed instructions in the code itself is recommended.

Key words: documentation, software maintenance, users.

****

[CODI77] C. Coding, "What Diagnostics Don't Tell You", Computer Decisions, vol. 9, no. 6, pp. 68-69, June 1977.

The use of proper compiler diagnostics is an important factor in reducing the cost and effort in software maintenance. This article discusses the problems and solutions in the area of diagnostics. The author suggests that since a compiler can translate an error code into a meaningful description much faster than a human, error codes should be translated into English. Even though memory space and compile time are needed to adequately diagnose errors, a machine is much more effective than a user would be. The author also discusses diagnostics that should be avoided such as "SYNTAX ERROR" and "ILLEGAL CHARACTER". Recommendations for improved diagnostics are as follows:

- o Pointer characters - a single character beneath the point of detection is useful. Also, each diagnostic should appear after the source statement in question.

- o Forgiving compilers - Under obvious conditions, the compiler should flag the error, make the appropriate, obvious assumption and continue. It is important, however, that the compiler indicate what assumptions are made and that they be logical choices.

Key words: compiler, diagnostics, errors, software maintenance.

****

[CONN80] A.P. Conn, "Maintenance: A Key Element in Computer Requirements Definition", COMPSAC Proceedings, IEEE Computer Society Press, pp. 401-406, November 1980.

This article analyzes software maintenance of large-scale, state-of-the-art systems. The requirements specification process is also examined. Of key concern is the suitability of the requirements for contractual maintenance agreements.

Key words: development, software maintenance, requirements definition, requirements specification.

****

[COOP78] J.D. Cooper, "Corporate Level Software Management", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-4, no. 4, pp. 319-326, July 1978.

Software management and standardardization are discussed from the corporate viewpoint. Standardization is presented as the most effective management device available at the corporate level for enhancing the overall software posture. Research initiatives, as well as corporate management actions available for favorably influencing the quality of software over its lifecycle are described.

Key words: lifecycle costs, lifecycle management plan, software maintenance, contracts, standardization.

****

[COOP79] J.D. Cooper and M.J. Fisher, Software Quality Management, Petrocelli publishers, 1979.

This book is comprised of papers presented at the 1979 Conference on Software Quality and Management. Issues such as maintainability and reliability are extensively discussed. One of the major premises of this text is that many software problems are the result of applying traditional hardware reliability, and maintainability techniques and procedures to the software development process.

Key words: maintainability, management, reliability, software maintenance.

****

[COUG83] J.D. Couger and M.A. Colter, Motivation of the Maintenance Programmer, CYSYS, Inc., 1983.

The results of an in-depth research effort on approaches to motivate programmers and analysts assigned to software maintenance activities is described. In phase I of the research, over 500 persons in 10 organizations completed the Couger-Zawacki (C-Z) diagnostic survey questionnaire for computer personnel. In phase II, on-site interviews were conducted with 104 persons (61 analysts and programmers and 43 supervisors/managers of system departments). Data were analyzed utilizing rigorous statistical methodology. The results indicate that productivity can be improved by motivation enhancement procedures.

Key words: maintenance programmers, motivation, programmers, productivity, software maintenance.

****

[COUG84] J.D. Couger and M.A. Colter, "The Effects of Maintenance Assignments on Goal Congruence for Programmers and Analysts", CIS Conference, ACM Press, June 1984.

The effect of varying amounts of software maintenance work on perceptions of role conflict, role clarity and reward clarity was analyzed in ten organizations representative of widely varying computing environments. The organizations had one characteristic in common - productivity of software maintenance personnel was satisfactory, consistent with that of personnel was satisfactory, consistent with that of personnel assigned to new development work. The research revealed significant differences in perceptions of congruency between high software maintenance and low software maintenance employees. High software maintenance employees appear to attain much higher levels of role clarity and reward clarity and much lower role conflict.

Key words: environments, goal congruence, software maintenance personnel, personnel, productivity, programmers, role clarity, role conflict, software maintenance.

****

[COWE79] W.R. Cowell and W.C. Miller, "The Toolpack Prospectus", Argonne National Laboratory, Applied Mathematics Division, Tech. Memo Number 341, WRC/wWCM90906, pp. 3-14, September 1979.

A group of computer scientists from universities, government laboratories, and private industry explored the idea of producing a systematized collection of software tools to facilitate the development and maintenance of Fortran programs. The collection would be a mechanism for making existing capabilities, resulting from research in software engineering, more readily available to Fortran users, in particular those developing numerical software. This prospectus was intended as a summary of the background, goals, and implementation strategy of the Toolpack project.

Key words: Fortran programs, program development, software maintenance, software engineering, software maintenance, software tools, Toolpack.

****

[CURT79] B. Curtis, S.B. Sheppard, P. Milliman, M.A. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", <u>IEEE Transactions on Software Engineering</u>, IEEE Computer Society Press, vol. SE-5, no. 2, pp. 96-104, March 1979.

Three software complexity measures (Halstead's E, McCabe's v(G), and the length as measured by number of statements) were compared to programmer performance on two software maintenance tasks. In an experiment on understanding, length and v(G) correlated with the percent of statements correctly recalled. In an experiment on modification, most significant correlations were obtained with metrics computed on modified rather than unmodified code. All three metrics correlated with both the accuracy of the modification and the time to completion. Relationships in both experiments occurred primarily in unstructured rather than structured code, and in code with no comments. The metrics were also most predictive of performance for less experienced programmers. Thus, these metrics appear to assess psychological complexity primarily where programming practices do not provide assistance in understanding the code.

Key words: commenting, complexity metrics, documentation, Halstead's E, human factors in software engineering, McCabe's v(G), mnemonic variable names, modern programming practices, modifications, software science, structured programming.

****

[DALY77] E.B. Daly, "Management of Software Development", IEEE Transactions on Software Engineering, IEEE Computer Society Press, pp. 229-242, May 1977.

This paper describes four major aspects of software management: development statistics, development process, development objectives, and software maintenance. The control of both large and small software projects is included in the analysis.

Key words: program methodologies, software design software maintenance, software development estimates, software management.

****

[DEAN82] J.S. Dean and B.P. McCune, "Advanced Tools for Software Maintenance", Rome Air Development Center, Griffiss AFB, Rome, N.Y., RADC, December 1982.

This is the final report on a project entitled "Software Maintenance Techniques". The purpose of this project was to study and design advanced software maintenance tools and techniques for the future ADA programming environment. Current software maintenance practices for Air Force C3I software were studied. Three out of the four major problems identified were attributed to the difficulty of comprehending software. Nine tools have been proposed to help solve these and other problems, including a tool to help coordinate the programming process (the "Programming Manager"), a tool to aid in the collection and use of documentation ("the Documentation Assistant), and an editor that is knowledgeable about what it is editing ("the Intelligent Editor"). The nine tools are based on the computer science technologies of artificial intelligence (particularly knowledge based and expert systems), automatic programming, intelligent user interfaces, formal verification, software engineering, programming environments, software metrics, and computer-assisted instruction.

Key words: ADA, ADA Programming Support Environment (APSE), artificial intelligence, documentation, environments, knowledge-based systems, metrics, program management, program modification, program verification, software engineering, software maintenance, software testing, software tools.

****

[DERO78] B.C. DeRoze and T.H. Nyman, "The Software Life Cycle - A Management and Technological Challenge in the Department of Defense", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-4, no. 4, pp. 309-318, July 1978.

This article stresses the need to manage software as a critical component of defense systems over their lifecycle is becoming widely recognized. Software costs are continuing to multiply in step with advancing weapons systems sophistication, and opportunities for cost avoidance now are leveraged against large dollar investments. The establishment of software lifecycle management policy and practices, and the vigorous development and application of new software technology are discussed in considerable depth.

Key words: computer resources, configuration management, DoD, software, software lifecycle, software management, software risk analysis, standardization.

****

[DEUT82] M. Deutsch, Software Verification and Validation: Realistic Project Approaches, Prentice-Hall, 1982.

Verification and validation approaches that have been used successfully on large-scale software projects are described. The author discusses methodologies that can be applied to complex software development and that take account of cost, schedule, and management realities in the actual production environment. Verification is defined as an activity which assures that the results of successive steps in the software development cycle correctly embrace the intentions of the previous step. Validation is defined as an activity that ensures the software end product is functional and which contains the feature prescribed by its requirements specification. The book is divided into five parts: Part 1 deals with verification and validation at summary level. Part 2 is a tutorial on testing techniques. Part 3 describes the role of automated tools in verification and validation. Part 4 explains the complete set of verification and validation activities performed over the lifecycle. Part 5 identifies future directions in the verification and validation field.

Key words: automated verification, management, methodologies, software lifecycle, testing methodologies, validation, verification.

****

[DONA80a] J.D. Donahoo and D. Swearinger, "A Review of Software Maintenance Technology", Rome Air Development Center, Grifiss, AFB, Rome, N.Y., RADC-TR-80-13, February 1980.

This report is intended to be a comprehensive statement about software maintenance techniques and tools in use today. It focuses on software maintenance technology as it is described and defined in open literature and technical reports. Material was selected based on its relevance to the subject of software maintenance and date it was published. Generally, only papers and articles published since 1974 and reports and books published since 1975 were selected.

Key words: software failures, software maintenance techniques, software maintenance tools, software testing, verification and validation.

****

[DONA80b] J.D. Donahoo and D. Swearinger, "Software Maintenance Technology", COMPSAC Proceedings, IEEE Computer Society Press, pp. 394-400, November 1980.

Effective computer software maintenance is becoming increasingly important to data processing center managers who must commit more and more of their resources to the support of operational software, to programmers and analysts who find themselves responsible for increasing volumes of program code and to users who have come to expect improved performance and expanded capabilities from their existing systems. This paper presents information compiled from published literature dealing with software maintenance technology; its development and application. The tools and techniques described in this paper have been selected from those presented in the report A Review of Software Maintenance Technology (RADC-TR-80-13). They are considered representative of the state-of-the-art in software software maintenance technology.

Key words: software lifecycle management, software maintenance, software software maintenance techniques, software maintenance tools, software modification.

****

[DONE84] B.S. Donefer, "Software Maintenance: An Investment, Not an Expense", Data Management, pp. 34-35, March 1984.

This article provides a different perspective on software software maintenance. It suggests that instead of being considered an expense, software software maintenance should be viewed as a means of protecting the software, a valuable asset. It also states that DP professionals must convince top management of the importance of software maintenance and the need to allocate resources for software maintenance.

Key words: software maintenance resources, management, software maintenance.

****

[DUNN81] R. Dunn and R. Ullman, Quality Assurance For Computer Software, McGraw-Hill, 1982.

An in-depth analysis on how to organize and run a quality assurance program to improve software reliability and performance is presented. Software quality assurance is defined as the mapping of the managerial precepts and design disciplines of quality assurance onto the applicable management and technological space of software engineering. The authors discuss how to solve software engineering problems using structured software development, configuration management, an orderly development cycle, purposeful and planned testing techniques, and independent verification and validation. The author suggests that quality assurance and productivity can be improved through defect collection, analysis, and feedback. Other topics discussed include:

-What needs to be controlled
-When controls should be introduced.
-How to search for defects
-Library control tools
-Staffing and training requirements
-How to sell top management on the need for a quality assurance program.

Key words: configuration management, library control tools, management, software, software defects, software engineering, software verification and validation, staffing, testing, training.

****

[DUNN84a] R.H. Dunn, "Configuration Control", Software Defect Removal, McGraw-Hill, pp. 285-303, 1984.

Some of the key software maintenance issues addressed in this book are:

1. Maintenance and modification operations require exact information about the program being altered.

2. Baselines are controlled collections of information concerning a program - initially just documentation, but later including code. All development work constitutes a departure from the previously established baseline.

3. Effective control is possible only with auditable library control tools.

4. The principal instruments used to control software are release notices for new material or new versions, change requests, which indicate the scope of effort to make a change and its impact on baseline material, and change notices which are used to disseminate the new status of a baseline.

Key words: baselines, change requests, configuration management, configuration control, documentation, library control tools, management, software, software maintenance, versions.

****

[DUNN84b] R.H. Dunn, "Maintenance and Modification", Software Defect Removal, McGraw-Hill, pp. 304-321, 1984.

Some of the key software maintenance issues addressed in this book are:

1. Properly performed software maintenance and modification activities represent iterations of the programming development process.

2. The difficult conditions of software maintenance programming result in the commission of an untoward number of errors.

3. With regard to the properties of programs that abet the work of software maintenance programmers, simplicity and clarity are the

most important.

4. The initiation of the removal of a defect from operational software starts with a program trouble report. Since this is the source document for a software maintenance project, printed forms should be used to guide the preparation and ensure the inclusion of all relevant information.

5. Software warranties offer advantages to both seller and buyer. However, it is necessary to carefully spell out the seller's obligations if the buyer modifies the product.

6. The rate at which programs become obsolete has to do not only with changes in programming practices and the operational environment but also with the state of preservation of the program's structure.

Key words: documentation, software maintenance, software maintenance programmers, program trouble reports, programming, programming environments, software maintenance, software warranties.

****

[DUNS80] H.E. Dunsmore and J.D. Gannon, "Analysis of the Effects of Programming Factors on Programming Effort", The _Journal of Systems and Software_, vol. 1, no. 2, pp. 141-153, February 1980.

The paper describes a series of experiments which were conducted to investigate program construction, comprehension, and modification. Ease of construction seemed related to average nesting depth, percentage of global variables used for data communication, average variables referenced, and average live variables per statement. Data communication and live variables were shown to be related to ease of modification.

Key words: data, communication, programming factors, languages.

****

[EBER80] R. Ebert, J. Lugger, and L. Goecke, _Practice in Software Adaptation and Maintenance_, North-Holland, 1980.

A collection of articles from the proceedings of the SAM-Workshop are contained in this book. The main topics and issues covered include:

1. conversion case studies;

2. software maintenance tools;

3. analysis and documentation techniques to describe internal software structures;

4. test techniques; and

5. user oriented interface and documentation.

Key words: conversion, documentation, reliability, software adaptation, software maintenance, testing techniques, user-interface.

****

[EDWA84] C. Edwards, "Information Systems Maintenance: An Integrated Perspective",<u>MIS Quarterly</u>, vol. 8, no. 4, pp. 237-256, December 1984.

This article suggests that information systems software maintenance is a complex task, requiring not only the maintenance of the applications software, but also all the other elements in an operational system. Literature relating to the maintenance of each element reveals substantial underdevelopment and fragmentation in many areas. Alternative methods of managing the software maintenance operation are examined and the implications of these methods in terms of designing the procedures, staffing the software maintenance function, and the need for communication are discussed.

Key words: information systems management, software maintenance, lifecycle, lifecycle management.

****

[ELSH82] J.L. Elshoff and M. Marcotty, "Improving Computer Program Readability to Aid Modification", <u>Communications of the ACM</u>, vol. 25, no. 8, pp. 512-521, August 1982.

Most of the problems encountered during change implementation can be attributed to difficulty in understanding the program to be modified. This paper provides guidance that that will make it easier: to write effective specifications , make accurate cost estimates modifications , design simpler software changes, and implement error-free modifications.

Key words: modification, modification cost, readability, software maintenance.

****

[ENDR75] A. Endres, "An Analysis of Errors and Their Causes in System Programs", <u>IEEE Transactions on Software</u>

Engineering, IEEE Computer Society Press, vol. SE-1, no. 2, pp. 140-149, June 1975.

Program errors detected during internal testing of the operating system DOS/VS are addressed in this paper. This testing operation serves as a basis for an investigation of error distributions in the operating system programs. Since errors are classified according to various attributes, conclusions can be drawn concerning the possible causes of these errors. The information can then be used to help determine the most effective methods for the detection and prevention of errors.

Key words: errors, testing, methodology, error classification, software reliability.

****

[ER84] M.C. Er, "Managing Source Code", Journal of Systems Management, pp. 12-14, October 1984.

One of the main problems encountered in development and maintenance of large projects is managing software changes. It is very difficult to keep track of all the changes that take place over a long period of time (i.e., who made the change, why it was changed, etc.). This article describes the features that should be included in a successful software control system, called the Source Code Control System (SCCS). SCCS provides two basic commands, GET and DELTA. GET is used to retrieve an old version of a program from a file, and edit and test the program. DELTA is used to store the new version back to the original file. SCCS saves disk and tape space because instead of storing the source code of two different versions entirely in a file, it compares the differences between these two versions and then stores only the differences to the original file. SCCS assigns a version number to each version of a program created by DELTA; recovers any previous version of a program; prompts the user for reasons for the update; facilitates protection from unauthorized users; and prevents more than one person from working on the same version of the program.

Key words: development, management, software changes, software maintenance, Source Code Control System, tools.

****

[FCSC83] Federal Conversion Support Center, "Guidelines for Planning and Implementing a Software Improvement Program (SIP)", General Services Administration (GSA), OSD FCSC-83/004, May 1983.

This document presents managers with guidance for establishing, planning, and implementing a SIP. It stresses the importance of timely and thorough planning, and recommends an emphasize the innovative, top-down, incremental approach. The emphasis is on "what needs to be done" rather than "how to accomplish" a SIP.

Key words: management, planning, quality assurance, software engineering, software improvement program, tools, training.

****

[FEIN84a] D.A. Feinberg, "A Simple Route to Program Configuration Control", <u>Data Management</u>, pp. 24-27, January 1984.

This article discusses program configuration control, an automated technique to manage the software maintenance function. The technique is based on four components: A baseline source file (the original software product), a change journal, an editor preprocessor and a `time machine' program (this program traces historical information). These components are explained and demonstrated through an example. Basically, the program configuration control technique works as follows:

1. A source program update is created
2. the editor uses this update to revise the master file
3. the preprocessor detects information on changes and appends it to the product's change journal.

Key words: software changes, configuration management, software maintenance.

****

[FEIN84b] D.A. Feinberg, "Credit Where It's Due", <u>Datamation</u>, pp. 256-258, June 15, 1984.

In this article, the author describes the difficulties facing software maintenance programmers: a negative perception of the software maintenance function; and the bias of DP organizations in favor of those programmers working on new projects. Two solutions are suggested: (1) replace the word 'software maintenance' with a name that provides a better description for the latter part of the software lifecycle; and (2) 'projectize' all but the simple error-correcting portions of the software maintenance. The author states that implementing these solutions will allow DP organizations to retain and recruit qualified personnel more easily.

Key words: attitudes, software maintenance programmers, personnel, software maintenance.

****

[FERR81] A. Ferrentino, "Making Software Development Estimates "Good", <u>Datamation</u>, pp. 179-182, September 1981.

Problems inherent in estimating factors connected with development projects are addressed. One such problem is the use of lines completed as opposed to the number of functions implemented. Another problem is the lack of sound methods for accurately predicting the time and manpower needed to develop a software system. Other factors mentioned are: cost of error removal; use of average productivity figures in estimating resources (high programmer variability make this a unreliable measure), and the realization that organizational requirements, which impact the software systems, are continually changing.

Key words: error removal, estimating factors, management, productivity, programmers, project management, software systems.

****

[FINK77] R.C. Fink, "Major Issues Involving the Development of an Effective Management Control System", COMPSAC Proceedings, IEEE Computer Society Press, pp. 533-538, November 1977.

Eight issues are presented in this paper which can help revitalize an ineffective Management Control System. The goal is to achieve better decisions by managers, ultimately leading to higher productivity of the computer programmers and analysts involved in software maintenance.

Key words: management, productivity, programmers, software maintenance.

****

[FIPS105] Guideline For Software Documentation Management, NBS Federal Information Processing Standards Publication 105, June 6, 1984.

This guideline can assist managers in establishing policies and procedures for preparation, distribution, control, and maintenance of documentation and computer programs. It identifies related software engineering standards and provides checklists of documentation that can be used by persons other than the originators of the programs. This document provides explicit advice on managing the planning, development, and production of software documentation.

Key words: documentation, guidelines, lifecycle, standards.

****

[FIPS106] Guideline on Software Maintenance, NBS Federal Information Processing Standards Publication 106, June 15, 1984.

There is a need for a strong, disciplined, clearly-defined approach to software maintenance. This report emphasizes the importance of considering of software maintenance throughout the lifecycle of a software system and stresses the need to plan, develop, use, and maintain software with future software maintenance in mind. General and functional definitions of software maintenance are provided and software change activities are identified. The report presents guidance for controlling and improving the software maintenance process and includes suggested criteria for deciding whether continued maintenance of a software system is justified. According to this report an organization's software maintenance efforts can be improved through the institution and enforcement of software maintenance policies, standards, procedures, and techniques.

Key words: adaptive maintenance, corrective maintenance, Federal Information Processing Standards publications, management, perfective maintenance, software engineering, lifecycle, tools.

****

[FISC77] K.F. Fischer, "A Test Case Selection Method for the Validation of Software Maintenance Modifications", <u>COMPSAC Proceedings</u>, IEEE Computer Society Press, pp. 421-426, November 8-11, 1977.

Validation of software maintenance modifications is commonly referred to as retest, and has yet to be adequately resolved. The problem is how to efficiently select previously run test cases to be rerun on the software to assure no degradation of reliability. This paper presents alternative retest philosophies and identifies a common operations research technique for solution. Detailed examples show how 0-1 integer programming can identify a minimum number of previously executed tests necessary to fully retest every affected program element at least once. Use of this model to determine proper selection of test cases can reduce the cost of software maintenance and increase confidence in the reliability of the code.

Key words: modifications, programming, reliability, retesting, software maintenance, software management, testing, validation.

****

[FORN83a] R.H. Forney, R.G. Race, and R.P. Nashleanas, "Data Processing: A Management Paradox", <u>Journal of Systems Management</u>, vol. 34, pp. 21-23, May 1983.

In this article, the causes for many software maintenance problems are identified. According to these authors: (1) management generally believes it is difficult, if not impossible to plan for software maintenance; (2) a rise in DP salaries is causing the cost of software maintenance to increase; (3) the person who wrote the original program is usually not the person who maintains it; (4) there is a lack of system documentation (i.e., when modifications occur, the documentation is rarely changed or updated); (5) present maintainability standards are not being applied to existing programs; and (6) an insufficient amount of resources are being allocated for software maintenance. Their recommendations are: change management's attitude towards software maintenance; make management more aware of the resources needed, and more aware of computer technology (i.e., software tools) to aid software maintenance programmers.

Key words: documentation, maintainability, management, resources, software maintenance, tools.

****

[FORN83b] R.H. Forney, R.G. Race, and R.P. Nashleanas, "A Strategic Approach: The use of Tools", Journal of Systems Management, vol. 34, pp. 29-31, June 1983.

The authors advocate the use of tools in software maintenance and describe tool attributes. The authors discuss two approaches for performing software maintenance. The traditional approach handles each request separately; the strategic approach groups together similar problems in order to reduce the programmer's familiarization time (i.e., faster understandability of the program) and make the solution easier to implement. Software tools, which automate highly repetitive tasks can help accomplish this goal. Some tools found to be effective for software maintenance are: static analyzer, formatter, documentor, code splitter and dynamic analyzer.

Key words: code splitter, documentor, dynamic analyzer, formatter, productivity, programmers, software maintenance, tools.

****

[FORN83c] R.H. Forney, R.G. Race and R.P. Nashleanas, "Reconditioning the Software Asset", Journal of Systems Management, vol. 34, pp. 20-21, July 1983.

This article addresses structuring engines which transform poorly structured programs into programs with a well-defined structure. This tool has been found to be effective as an aid in planning for software maintenance. Several specific and broad goals for software maintenance are also discussed.

Key words: management, software maintenance, structured retrofitting, structuring engines, tools.

****

[FRAN83] W.L. Frank, Critical Issues in Software, John Wiley and Sons, 1983.

This book describes software from both an economic and historical point of view. It addresses the: cost factors relating to software; lifecycle process for developing, operating, and maintaining software; history and current state of software development; economic issues of productivity for software development and maintenance; impact of microcomputers on the software industry; and the importance of managing the software process.

Key words: management, microcomputers, productivity, software cost, software development, software lifecycle, software maintenance.

****

[GAO80] "Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs", General Accounting Office (GAO), FGMSD-80-38, April 29, 1980.

GAO conducted a survey to determine if the latest tools and techniques were being used to develop and maintain Federal computer software. They found that computer specialists at many agencies were unaware of the newer, better methods; others were reluctant to adapt them. GAO recommends various ways to promote the use of better tools and techniques to improve management control and control spending.

Key words: development, management, software maintenance, software technology, software tools.

****

[GAO81] "Government-Wide Guidelines and Management to Improve ADP Systems Development", General Accounting Office (GAO), AFMD-81-20, February 20, 1981.

In more than 57 reports in the past 10 years, GAO has identified management weaknesses in the design and development of large, complex Federal data processing systems. These weaknesses led to the waste of over $300 million and systems which

-were not cost effective,
-did not meet agency needs,
-took too long to develop, or
-simply did not work.

The report suggests that guidelines for a structured management approach to ADP systems development be developed, that their use be required throughout the Government, and that a cost reimbursable center to assist agency top management in planning, designing, acquiring, and evaluating large, complex ADP systems development projects be established.

Key words: data processing systems, development, design, management, planning, structured management.

****

[GAO83] "The Air Force Can Improve Its Maintenance Informations Systems", General Accounting Office (GAO), GAO/GCD-83-20, January 25, 1983.

The Air Force spends millions on aircraft software maintenance information. This report suggests that a timely development decision on standard use of the software maintenance information system throughout the Air Force-wide is necessary to eliminate duplicate system development efforts and to prevent the acquisition of unnecessary computer equipment. Other recommendations are made to help reduce many of the software maintenance and information management problems.

Key words: data, information management, information systems, management, system development.

****

[GILB77] T. Gilb, "The Measurement of Software Reliability and Maintainability", <u>Computers and People</u>, pp. 16-21, September 1977.

This article presents a number of technical and management approaches to the production and maintenance of reliable software. The four major areas covered are:

1. Quantification of reliability and maintainability: the reliability and maintainability goals of a project must be stated in measurable terms.
2. Automation of the reliability and maintainability tasks: the evaluation of source program structure is currently capable of being done automatically.
3. Management of reliability and maintainability: managers should set standards which are checkable by machine so that they can be effectively and economically controlled.
4. Redundancy-based reliability technology: automation of the production set of test cases.

Key words: automation, management, programming, redundancy, software maintainability, software reliability, program analyzer, standards.

****

[GILB79a] T. Gilb, "A Comment on the Definition of Maintainability", <u>Software Engineering Notes</u>, IEEE Computer Society Press, vol. 4, no. 3, pp. 32-33, July 1979.

The theme of this article is that maintainability is directly measurable at many stages of system development. The author list ten measurable factors to define maintainability. To aid managers in predicting the maintainability of the company's programs, they should use techniques that will measure the effect of design decisions during development.

Key words: maintainability, management, measuring, software maintenance, development.

****

[GILB79b] T. Gilb, "Gilb's Methodology: Need for Training in Program Maintenance", <u>Computer Weekly</u>, p.6, March 15, 1979.

This article reveals the necessity for training in the area of software maintenance. The author believes that the rapid turnover in software maintenance personnel is caused by the fact that the workers are neither well-trained nor motivated to maintain software. He suggests that colleges or other training institutions should establish special programs to address software maintenance issues as well as tools and techniques required to successfully maintain a system. According to the author, it is essential that a percentage of the increasingly large number of persons entering the field of computer science be recruited and trained to address the demanding problems of software maintenance.

Key words: colleges, software maintenance, training, tools.

****

[GILH82] I.A. Gilhooley, "Organizing for Effective Program Change Control", <u>The Internal Auditor</u>, vol. 39, pp. 59-63, February 1982.

Many organizations do not have formal systems for controlling changes to application programs. The author suggests an approach which centers around the creation of a change review board. All changes should be: authorized by an appropriate level of management, fully tested, and migrated into the production mode in a controlled fashion, and include a complete management trail from initiation to implementation. The change review board should include people from the following departments: systems and programming, computer operations, primary users, and audit. The change review board has six major functions: review change requests, assign priorities and resources, review system analysis and design, coordinate company resources, ensure completion of a change request, and control the implementation of changes. A detailed discussion of this approach is presented.

Key words: change review board, change requests, management, change control, software maintenance.

****

[GILL83a] P. Gillin, "Experts Detail Methods to Control Maintenance Costs", Computerworld, p. 8, February 21, 1983.

The following advice is offered by a number of experts on reducing software maintenance costs:

-coordinate modification and documentation.

-adopt the `release concept' where patches and enhancements are collected over a period of time, implemented at one time, and then offered as a new release.

-make management aware of the need to encourage greater respect for maintenance within the DP department. Other suggestions for controlling software maintenance costs are presented.

Key words: documentation, enhancements, software maintenance costs, release concept, software maintenance.

****

[GILL83b] P. Gillin, "Users Fighting to Contain Costs", Computerworld, p. 8, February 21, 1983.

Changes undertaken by four companies to reduce software maintenance costs are discussed in this article. Three of these companies decided to purchase application packages. The case studies presented support the decisions by each company to purchase off-the-shelf packages.

Key words: maintainability, maintenance costs, software packages, software maintenance.

****

[GLAS79] R.L. Glass, Software Reliability Guidebook, Prentice-Hall, 1979.

This book discusses technical and management techniques for achieving software reliability, and the interrelationship between reliability and software maintenance. The author describes a variety of tools and methodologies to assist in making software more reliable. Special emphasis is placed on the problems of large projects.

Key words: management, methodologies, reliability, software maintenance, software reliability, tools.

****

[GLAS81a] R.L. Glass, "Persistent Software Errors", <u>IEEE Transactions on Software Engineering</u>, IEEE Computer Society Press, vol. SE-7, no. 2, pp. 162-168, March 1981.

Persistent software errors, including those which are not discovered until the software becomes operational, are predominantly errors of logic. Peer design and code review, desk checking, and rigorous testing have been found to be the most helpful in reducing this type of error. Glass concludes that new and better methodologies are needed.

Key words: complexity, software error, problem report, testing.

****

[GLAS81b] R.L. Glass and R.A. Noiseux, <u>Software Maintenance Guidebook</u>, Prentice-Hall, 1981.

This book discusses the people, technical, and managerial aspects of software maintenance. Maintenance is discussed in relation to the software lifecycle. The authors examine several personality traits which can affect the work of a maintainer and describe the tools and techniques found to be effective. This text provides a comprehensive management perspective on software maintenance is provided.

Key words: maintenance programmers, management, personality traits, software lifecycle, software maintenance, tools.

****

[GLAS82] R.L. Glass, <u>Modern Programming Practices: A Report From Industry</u>, Prentice-Hall, 1982.

The author presents an industry view of industrial software practices. This book relates the experiences of six companies who were paid by the Air Force to report their techniques for developing and maintaining software. These reports also discussed techniques for programming quality, performance, and productivity. With regards to software maintenance, companies discussed it very little in their reports. Techniques for software maintenance were not mentioned. However, configuration management was almost an universal concern. Change control was one of the most frequently mentioned modern programming practices. Unfortunately, software maintenance was considered by companies to be necessary because it was "not done right the first time".

Key words: change control, configuration management, modern programming practices, productivity, programming, quality, software.

****

[GLAS84] R.L. Glass, "Report on the Software Maintenance Workshop", System Development, vol. 3, no. 12, pp. 1-3, February 1984.

This article describes the proceedings at the 1983 software software maintenance shop. First, there were keynote presentations evaluating the problems in software maintenance and presenting solutions to these problems. Second, there were technical sessions which included presentations by Japanese and Swedish speakers. Finally there were two sessions dealing with methods of developing software to make it more maintainable. This article provides a good update about present issues, solutions, and problems in software maintenance.

Key words: maintainability, maintenance.

****

[GREE81] J.F.Green and B.F. Selby, "Dynamic Planning and Control of Software Maintenance: A Fiscal Approach", Naval Postgraduate School, Master's Thesis, 1981.

Until recently, much of the budget planning for software systems has been primarily targeted at costs incurred during the development phase. However, with increasing software system life span and complexity, software maintenance costs have become a more prevalent concern. As a result of necessary corrections for design errors and evolutionary software maintenance, post-delivery investment in software systems now requires a greater proportional share of the lifecycle costs. In this research, various methodologies and system factors relating to software cost accounting are reviewed with the intent of developing a cost control model for arriving at a well-structured view for the management of the software maintenance phase of the software lifecycle. The model proposed embodies a planning concept for establishing a software maintenance strategy and a control concept for analyzing manloading requirements during the software maintenance phase.

Key words: software evolution, software lifecycle, software macroestimation, software maintenance, software microestimation.

****

[GREM84] L.L. Gremillion, "Determinants of Program Repair Maintenance Requirements", Communications of the ACM, vol. 27, no. 8, pp. 826-832, August 1984.

Considerable resources are devoted to the maintenance of programs including that required to correct errors not discovered until after the programs are delivered to the user. A number of factors are believed to affect the occurrence of these errors, e.g., the complexity of the programs, the intensity with which programs are used, and the programming style. Several hundred programs making up a manufacturing support system are analyzed to study the relationships between the number of delivered errors and measures of the programs' size and complexity (particularly as measured by software science metrics), frequency of use, and age. Not surprisingly, program size is found to be the best predictor of repair software maintenance requirements. Repair software maintenance is more highly correlated with the number of lines of source code in the program than it is to the software science metrics, which is surprising in light of previously reported results. Actual error rate is found to be much higher that that which would be predicted from program characteristics.

Key words: program complexity, program errors, software maintenance.

****

[GROC84] J.M. Grochow, "When and How to Modify Packages", Computerworld, pp. 7-13, March 12, 1984.

This paper analyzes the option of purchasing and modifying software packages as opposed to in-house development. Different types of changes to a package are discussed. These changes include: fixing reports and input screens, adding a new feature, changing package performance in a particular area, and interfacing a package with other operational systems. The author suggests alternative approaches for modifying software packages. Some factors that should be considered while evaluating software packages are: warranty, customer support, ongoing vendor software maintenance, and vendor assistance on program modifications. He describes modifying packages as a cost-effective solution to system development.

Key words: customer support, packages, software maintenance, vendor software maintenance, warranty.

****

[GROS82] J.F. Gross, "Writing Straightforward, Maintainable Programs", Computer Programming Management, Van Nostrand Reinhold, pp. 101-112, 1982.

This chapter covers several topics in software maintenance. The author discusses three situations that can force a program to be modified. Various causes for the development of hard-to-maintain programs are examined. These causes are as follows: philosophy of perfection, educational bias towards one-shot programs, tradition, laziness, the company's work environment, and a programmer's need to be creative and to personalize his work. The use of a different philosophy to solve these problems is also discussed. The basis of this philosophy is to write programs with software maintenance in mind. In addition, the use of foresight, a team effort in programming, and instituting accurate reports and schedules are included in this philosophy. The use of various standards and techniques to improve program maintainability are recommended. The author concludes that improvements in design and coding will save money and increase productivity.

Key words: productivity, program maintainability, software maintenance, programmers, programming, standards, work environment.

****

[GUIM83] T. Guimaraes, "Managing Application Program Maintenance Expenditures", Communications of the ACM, vol. 26, no. 10, pp. 739-746, October 1983.

Program software maintenance represents a major portion of the total expenditures on application programs. Despite the attention this subject has received in the MIS literature, new guidelines to action in this area remain of great interest to practitioners. A large number of variables thought to be determinants of application program software maintenance expenditure have been studied through the inspection of application portfolios and personal interviews with top computer executives and systems development personnel. Based on the results, recommendations are made on how to reduce application software maintenance expenditures.

Key words: database management systems, personnel, software documentation, software maintenance costs, program redevelopment, program size, software packages.

****

[GUND73] R.E. Gunderman, "A Glimpse Into Program Maintenance", Datamation, vol. 19, no. 6, pp. 99-101, June 1973.

This article gives a brief overview of the problems and solutions involved with program software maintenance. The author states that software maintenance is not an activity that should be given to beginner programmers. He observed a significant amount of programming time can be consumed in software maintenance and large programs are virtually never completely debugged. The author suggests a software maintenance-support facility as an aid for software maintenance programming. This facility would provide: (1)

a software maintenance specialist who could isolate program bugs and give feedback to designers for enhancing the maintainability of programs (2) classification of bug types and their correlation with program sequences, routines, and/or instructions (3) program definition through an established set of simple verbs (4) software maintenance history and documentation of any pertinent events and actions taken during a software maintenance task (5) a software documenter (6) software maintenance data and statistics from software maintenance activities.

Key words: documentation, debugging, software maintenance programming, software maintenance-support facility.

****

[HALL78] T.G. Hallin and R.C. Hansen, "Toward a Better Method of Software Testing", COMPSAC Proceedings, IEEE Computer Society Press, pp. 153-157, November 13-16, 1978.

While software testing is necessary for the production of quality products, it is generally one of the least efficient steps in the software development process. Testing typically consumes over 50 percent of the development effort. Software testing is thought of as an art and not a science, and artistic approaches to testing propagate the thinking. The complex test systems, complex program implementations, seemingly unrelated requirements and design information, and the complex system interactions all come to haunt and confuse the poor software developer when it comes time to certify the product is correct. The authors have attempted to step back from all of these program intricacies and define what it means to test software programs. A cause and effect testing methodology is described. Promising results are provided from two development efforts that have used this approach for software testing.

Key words: quality, software development, software testing.

****

[HANN85] J. Hannan, "Consultant: Upkeep is Service to Users", Government Computer News, p. 41 and p. 54, March 29, 1985.

This article discusses reducing software maintenance cost and the application backlog. Most of the material in this paper is taken from an article in Auerbach's System Development Management publication. The ADP manager establish short-term and long-term objectives. Since software maintenance is a service to users the short-term objective is to execute the change request as quickly as possible. Guidelines for establishing priorities and long-term objectives in the software maintenance phase are presented. Staffing, organizing, and directing the software maintenance function are also discussed. Improvent in the software maintenance team's performance can be achieved by attention to:

1. Service requests - all requests should be logged to ensure that none of them are lost.
2. Maintenance steps - software maintenance and development requests should be handled the same way.
3. Setting Priorities - the software maintenance manager, not users, should set priorities.
4. Implement standards for software maintenance, design, documentation, and programming.
5. Maintenance Planning - The software maintenance manager must establish software maintenance goals and the means to achieve them.
6. Training - software maintenance programmers should be trained in all applicable ADP tools and techniques.

Key words: documentation, planning, management, organizing, programming, software maintenance, staffing, standards, training.

****

[HARR82] W. Harrison, K. Magel, R. Kluczny, and A. DeKock, "Applying Software Complexity Metrics to Program Maintenance", Computer, pp. 65-79, September 1982.

Predicting software complexity can save millions in software maintenance costs, but while current measures can be used to some degree, most are not sufficiently sensitive or comprehensive.

Key words: data flow, data structures, metrics, control structures, software maintenance, software complexity.

****

[HARR83] W. Harrison, K. Magel, and R. Kluczny, "Research in Software Maintenance", Journal of Systems Management, vol. 34, pp. 10-14, July 1983.

This article discusses factors causing high software software maintenance expenditures and presents two descriptive models of the software maintenance field. The authors estimate that between 40% and 75% of software and hardware expenses are for software maintenance. An example is given to show that decreasing the cost of software maintenance is the only viable way of increasing the net return of the software. Also, the authors compare software maintenance to other types of software maintenance (i.e., hardware). Two software maintenance process and environment models are described. The first model emphasis is on user perception of th e software maintenance process, while the second focuses on the characteristics in the software maintenance environment. Use of these models help to show the how improved communication between users and programmers increases the effectiveness of software maintenance.

Key words: software maintenance costs, software maintenance environment.

****

[HARR84] W. Harrison, "Software Complexity Metrics", Journal of Systems Management, pp. 28-30, July 1984.

This article reviews the status of software complexity metrics and describes present research in this area. The two types of performance and error data are also described. Software complexity metrics are measurements that assess how difficult it is to understand programs. Typically, such a metric attempts to measure to what degree some characteristic exists within the software. Usually, the characteristic of interest is one which the metrician believes hinders the programmer from understanding the program. These metrics are important in software maintenance because a major factor affecting software maintainability is how easily can the program be understood. The two approaches for evaluating metrics are: Experimentation and field studies. Most experimental studies are done with college students using small programs. Field studies are usually more accurate since they examine professional programmers working in a professional environment. The characteristics which are often measured in assessing program complexity can be divided into three main groups:

1. Control flow - the number of decisions made in the program and their interrelationships.
2. Program size - the size of the program which can be expressed in many ways.
3. Data structuring and flow - the relationship and use of variables within the program.

Key words: control flow, data structuring, metrics, program complexity, software maintenance.

****

[HENK81] T. Henkel, "Opportunity Sighted in Software Maintenance", Computerworld, p. 13, May 11, 1981.

This article discusses the future of software software maintenance programmers. A breakdown of software maintenance programmer activities is provided: coding and emergency repairs (10%), enhancements to existing systems (45%), changing conditions or the way the business is handled (25%), upgrades or recoding to adapt to hardware upgrades (10%), and expanding the existing system to accommodate a higher demand (10%). An assessment of job opportunities (i.e., pay and status) for software maintenance programmers is presented. The author notes that job opportunities are becoming an increasingly important function within the organization. This article disputes the widely held notion that software maintenance programming is a 'dead-end' job.

Key words: maintenance programmer, software maintenance.

****

[HERN83] M.A. Herndon and J.A. McCall, "A Tool For Software Maintenance Management", Softfair Proceedings, IEEE Computer Society Press, pp. 157-163, July 25-28, 1983.

Software maintenance accounts for a large percentage of the life cycle costs of a software system. Little recognition by the software tool community has been given to the unique problems and tool requirements of software maintenance personnel. This paper discusses these problems, the software characteristics required for a maintainable product, and a system of tools which has been developed specifically to support software maintenance organizations.

Key words: maintainability, software maintenance, software maintenance management, software tools.

****

[HIGG81] D.A. Higgins, "Structured Maintenance - New Tools for Old Problems", Computerworld, pp. 31-40, June 15, 1981.

This report discusses methods for improving program maintainability. The author emphasizes the importance of a program that is easy to maintain. An example of a COBOL program including GOTOs is presented in order to prove that removing the GOTOs does not necessarily improve the program. Since excessive GOTOs are only a symptom of poor program design, their removal does nothing to improve the design of a bad program. A method for the data structured design of programs is presented. The four data structures found in every program and their significance for improving program maintainability is provided.

Key words: COBOL, data structured design, data structures, design, GOTOs, program maintainability, software maintenance, software maintenance management, structured software maintenance, structured programs.

****

[HOUT83] C.A. Houtz, "Software Improvement Program (SIP): A Treatment For Software Senility", General Services Administration (GSA), September 1983.

ADP organizations are plagued with high software maintenance costs, long delays in responding to users' changing needs, and continued development and maintenance of antiquated, outmoded, and relatively obsolete software. This software can be thought of as being in an advanced state of software senility, a degenerative condition, which if not corrected, will eventually render the software totally useless. A reversal of this situation requires a Software Improvement Program (SIP), which is a treatment for the ills of software senility, and offers a cure for many of the software problems from which most ADP organizations are suffering. A SIP is an incremental and evolutionary approach to modernizing software to maximize its value, quality, efficiency, and effectiveness, while simultaneously preserving the value of past software investments and enabling the organization to capitalize on today's modern ADP technology, as well as future technological advances in the field. This paper describes the SIP philosophy and presents a strategy for implementing a dynamic, ongoing SIP coupled with a sound Software Engineering Technology, (SET), to attack the causative factors of the ever-growing software crisis.

Key words: software engineering technology (SET), software improvement, software improvement program (SIP), software obsolescence, stepwise refinement.

****

[HOWA83a] P.C. Howard (editor), "Motivating Maintenance Personnel", System Development, pp. 1-2, July 1983.

This article is based on a session held at the 1983 National Computer Conference, entitled motivating software maintenance personnel. The thesis of this report is that the motivating potential needed by software maintenance programmers who have high need for professional growth is often missing. This, coupled with a relatively short learning curve for software maintenance personnel, leads to poor morale and low productivity. The findings of the research effort on how to motivate software maintenance personnel are presented. They include changes initiated in a number of organizations to motivate the data processing staff in such areas as: (1) job training, (2) simplification of the programming process - large applications are divided into smaller pieces, (3) cross-utilized development and software maintenance personnel to eliminate the stigma of software maintenance programming, and (4) established a user-programmer link.

Key words: maintenance programmers, motivation, personnel, productivity, software maintenance, training.

****

[HOWA83b] P.C. Howard (editor), "Reducing System Maintenance", System Development, p. 7, July 1983.

This article describes a program for reducing software maintenance which has three separate, but overlapping stages: (1) measuring the existing software maintenance effort, (2) summarizing and analyzing the software maintenance activity, and (3) resolving the software maintenance problems by order of priority. A discussion of procedures for generating and tracking activities associated with the software maintenance effort is presented. One of the best sources of information is the data available from the personal logs of software maintenance activities. According to this study, extensive analysis of the logs enabled studied. From this study, management can prepare procedures and guidelines to control costly and unnecessary system software maintenance.

Key words: programmers, software maintenance, software maintenance management, system software maintenance, users.

****

[HOWA84a] P.C. Howard, "Issues in Software Maintenance", System Development, vol. 3, no. 12, pp. 3-4, February 1984.

This article discusses five principal issues relating to software software maintenance. They are as follows: (1) a conceptual issue involving the definition of software maintenance (2) measurement of software maintenance activities (3) the scale of effort spent on software software maintenance (about 50%) (4) the organization of and the placement of the software maintenance activity (5) productivity issues in software maintenance. This article is based on surveys by Bennet P. Lientz.

Key words: maintenance definition, software maintenance organization, measurement of software maintenance activities, productivity, software maintenance.

****

[HOWA84b] P.C. Howard (editor), "Hints on Controlling Maintenance Expenditures", System Development, vol. 4, no. 5, pp. 6-7, July 1984.

This article offers suggestions to managers for controlling software maintenance expenditures. These suggestions are based on the findings from a survey of 43 organizations and five in-depth case studies, which examined the nature of the software maintenance activity in commercial user organizations. A list of the findings from this survey are presented in the article. The suggestions for improving management of application program software maintenance expenditures are as follows:

1. Use Data Base Management Systems
2. Do not use Assembly Language
3. Use application software packages
4. Use self-contained Query Language
5. Provide direct user access
6. Preplan equipment or software changes
7. Develop useful documentation.

This article contains useful suggestions for reducing software maintenance costs.

Key words: assembly language, data base management system, documentation, software maintenance costs, management, software maintenance, self-contained query language, software changes, software maintenance, software packages.

****

[HOWA84c] P. C. Howard (editor), "Building in Maintainability", System Development, vol. 4, no. 10, pp. 3-4, December 1984.

This article discusses methods for building in maintainability, both for new and existing systems. The problems which prevent maintainability are as follows: lack of data on a organization's software maintenance activities, the existence of inflexible software, poor management of software maintenance personnel, and the flood of user requests for enhancements. There is a critical need to adopt methods for designing software that will make the software easier to change. Fourth generation tools as a way of escaping the software maintenance problems associated with procedural languages. Some of the positive effects from these tools on the software maintenance effort include: improved readability of generated code, improved documentation, ability to change programs faster and ability of end-users to make their own enhancements. The article concludes with the statement that for conventional applications fourth generations tools offers a tremendous advantage both in the development and software maintenance of software.

Key words: fourth-generation tools, maintainability, software maintenance personnel, management, software maintenance.

****

[HUAN75] J.C. Huang, "An Approach to Program Testing", ACM Computing Surveys, vol. 7, no. 3, pp. 113-128, September 1975.

One of the practical methods commonly used to detect the presence of errors in a computer program is to test it for a set of test cases. The probability of discovering errors through testing can be increased by selecting test cases in such a way that each and every branch in the flowchart will be traversed at least once during the test. This tutorial describes the problems involved and the methods that can be used to satisfy the test requirement.

Key words: directed graph, path analysis, path predicate, program analysis, program instrumentation, program testing, test-case generation.

****

[HULI84] J. Huling, "User Requests Needn't be a Pain in the Neck", Computerworld, p. 64, May 28, 1984.

This article describes a procedure for handling a backlog of user requests. The first step of the procedure is to organize the requests by some criteria (i.e., size, complexity, or program function). The advantages of organizing the requests are to eliminate any duplication of effort and to allow the analyst to assign priorities to the larger logical units. The second step is to estimate the time and resources needed to complete each group of requests. The third step is to contact the person requesting the work to determine if the request is valid and to let the user know the time and cost involved. The fourth step is to assign priorities and prepare a schedule. When a request (or group of requests) is complete, the person who requested the work should be informed. The fifth and most important step is to gain management support.

Key words: management, organizing requests, programmers, scheduling, user needs, user requests.

****

[INMO84] W. Inmon, "On-Line Maintenance: A New Ball Game", Computerworld, p. 45 and p. 66, September 3, 1984.

This article compares and contrasts system software maintenance in the batch and on-line environments. In the batch environment, there are three categories of software maintenance: "fix it" software maintenance for immediate problems, system modifications, and system additions. However, in the on-line environment there is a fourth category of software maintenance: post-implementation design and development. This fourth category is usually very expensive due to the difficulty of making fundamental changes to a system that has already been implemented. Companies which did not consider important design issues (i.e., performance and availability) had high software maintenance costs. This article provides an interesting comparison between the software maintenance done in the batch and on-line environments.

Key words: batch environments, software maintenance costs, on-line environments, software maintenance, system software maintenance.

****

[IITR82] IITRI Staff, "MPP in the Pave Paws Software Maintenance", Rome Air Development Center, Griffiss AFB, N.Y., June 1982.

This report presents the results of a study to establish a baseline software maintenance experience database and to determine relationships which exist on the use of modern programming practices and software engineering tools on the ease of software maintenance of the PAVE PAWS (Phased Array Warning System).

Key words: modern programming practices, operation and maintenance, software data collection, software engineering tools.

****

[JONE78] T.C. Jones, "Measuring Programming Quality and Productivity", IBM Systems Journal, vol 17, no. 1, pp. 39-63, 1978.

An analysis of common units of measure for assessing program quality and programmer productivity reveals that some standard measures are intrinsically paradoxical. Lines of code per programmer-month and cost per defect are in this category. Presented here are attempts to go beyond such paradoxical units as these. An in-depth discussion of the usefulness of separating quality measurements into measures of defect removal efficiency and defect prevention, and the usefulness of separating productivity measurements into work units and cost units is provided.

Key words: defect prevention, defect removal, productivity measurements, program quality, programmer productivity, programming, quality measurements.

****

[KAPL77] R.S. Kaplan, "Issue: An Information System and Software Update Environment", COMPSAC Proceedings, IEEE Computer Society Press, pp. 527-532, November 8-11, 1977.

An Information System and Software Update Environment (ISSUE) is a data base management system that supports software control and administration. ISSUE has been implemented on an IBM 370/168 and an AMDAHL 470/V6, both running under IBM`s TSS (Time Sharing System) operating system at Bell Laboratories. The major components of ISSUE are described and the procedures used to administer a system under ISSUE are documented.

Key words: data base management system, IBM 370/168, Information System and Software Update Environment (ISSUE), software control.

****

[KAPU83] G. Kapur, "Software Maintenance", Computerworld, pp. 13-22, September 26, 1983.

This article analyzes the problems of software maintenance and offers a plan for reducing its high cost. Reasons for the neglect of software maintenance and for the less than effective use of some of the new innovations are provided. Some of the present factors contributing to the problems of software maintenance include: (1) DP management devotes little effort to solving the software maintenance problem; (2) new tools and methodologies are mostly used for development; (3) a lack of high-quality staff in the software maintenance department; (4) a belief that software engineering techniques are not applicable to software software maintenance. A survey of 14 DP organizations reveals some of the causes for the high cost of software maintenance. The author describes a maintenance reduction plan, which addresses software maintenance management audit, software system audit, and software rehabilitation, are explained in detail.

Key words: maintenance costs, software reduction plan, management, software engineering, software maintenance, software rehabilitation, staffing, tools, users.

****

[KEID74] S.P. Keider, "Why Projects Fail", <u>Datamation</u>, pp. 53-55, December 1974.

The author divides projects into five distinct phases: pre-initiation period, initiation period, project duration, project termination period, and post-termination period and describes some serious problems that could jeopardize the success of a project. By using examples of errors made during a number of projects, this author provides an extensive empirical information to aid project management.

Key words: project characteristics, project failures, project management.

****

[KERN78] B.W. Kernighan and P.J. Plauger, <u>The Elements of Programming Style</u>, McGraw-Hill, 1978.

This book provides an analysis of good programming style by means of a set of about 100 rules augmented by examples and discussion. The examples are from 'real' programs written in Fortran and PL/1. The authors also discuss the following ideas:

1. Structured Coding Techniques
2. Top-down design and other design issues
3. Modularize - use subroutines
4. Let the data structure the program
5. Don't comment bad code-rewrite it!

Key words: design issues, Fortran, modularize, PL/1, programming, programming style, programs, rewriting, structured coding, top-down design.

****

[KHAN75] Z. Khan, "How to Tackle The Systems Maintenance Diemma", <u>Canadian</u> <u>Datasystems</u>, pp. 30-32, March 1975.

This article analyzes approaches for obtaining more efficient systems software maintenance including: different types of software maintenance, controlling and evaluating software maintenance requests, how to organize the software maintenance function within the DP department, and suggested solutions for reducing the time spent on software maintenance. The author suggests that establishing a formalized procedure for request, review, and approval of software maintenance activities is the best way to control the amount of resources spent on software maintenance.

Key words: maintenance requests, management, organizing the software maintenance function, software maintenance, systems software maintenance.

****

[KULL83] D. Kull, "Recovering From Mistakes", <u>Computer Decisions</u>, pp. 145-151, April 1983.

This article discusses a number of actions that should be implemented when a DP department makes a serious mistake. Rebounding from mistakes depends on contingency planning - anticipating what might go wrong and a course of action to take if something does. Another preventive measure is to monitor user attitudes so that your organization will focus its service on the users' needs. The author suggest giving the users a ten-item questionnaire about every six months to enable the DP department to pinpoint and correct problems quickly. For some projects, which are vulnerable to foul-ups, one or two aides should be assigned to monitor the project. In addition, the author recommends other preventive measures : involve users in important decisions and do not raise user expectations too high, instead, instill in the users a sense of reality. A six step error recovery procedure is presented.

Key words: contingency planning, credibility, errors, management, users, users' attitudes, users' needs.

****

[LAFF77] A.W. Laffan, "Software Maintenance - The Other Side of the Delivery Goal", Proceedings of the 16th ACM/NBS Annual Technical Symposium, pp. 103-106, June 2, 1977.

This paper explores the goals and problems of software maintenance. The twin goals of software maintenance, correction and change, are explained. Problems such as the inability of management to comprehend the many differences between software and hardware software maintenance, the complexity of software, and poor documentation are discussed. The author concludes by stating the following solutions to these problems : acceptance of software software maintenance as a major expense, a quality assurance plan for all phases of software development, using configuration management, using code that is well-documented and maintainable, and planning for software maintenance. An excellent article, the author's points are clear, concise, and easy to comprehend.

Key words: change, configuration management, correction, documentation, maintainability, management, quality assurance, software complexity, software maintenance.

****

[LAFF78] A.W. Laffan, "The Software Maintenance Problem", Eleventh Hawaii International Conference on Systems Sciences Proceedings, DPMA, pp. 119-123, January 1978.

System software maintenance is becoming increasingly critical. Maintainability must become a management objective. This paper discusses the software maintenance problem in terms of its inevitability, its difference from hardware software maintenance, its cost, and its importance.

Key words: maintainability, software maintenance, software maintenance costs, management, software, software maintenance.

****

[LANE84] R.G. Langergan and C.A. Grasso, "Software Engineering with Reusable Designs and Code", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-10, no. 5, pp. 498-501, September 1984.

For over six years, Raytheon's Missile Systems Division, Information Processing Systems Organization has used a successful approach in developing and maintaining business software. The approach centers on the fact that 60 percent of all business application designs and code are redundant and can be standardized and reused. This approach has resulted in significant gains in productivity and reliability and improved end-user relations, while providing better utilization of data processing personnel, primarily in the software maintenance phase of the software lifecycle.

Key words: personnel, productivity, reliability, reusable code, reusable designs, software development, software engineering, software lifecycle, software maintenance, standards.

****

[LEAV83] D. Leavitt, "Maintenance Work is Different", Software News, vol. 3, no. 10, pp. 30 and p. 37, October 1983.

This article distinguishes between the psychological factors of development and software maintenance programming. It is based on a presentation by Prof. J. Daniel Couger of the University of Colorado at the 1983 National Computer Conference. Couger claims that organizations using his methods in software maintenance activities have experienced up to a 40% increase in productivity. Couger listed five variables which can be used to motivate a worker regardless of his personality type: skill variety, task identity, task significance, autonomy, and feedback from the job. This article provides insight on the psychological factors involved with software maintenance.

Key words: automation, development programming, feedback, software maintenance programming, productivity, psychological factors, software maintenance, skill variety, task identity, task significance,

****

[LEHM76] M.M. Lehman and F.M. Parr, "Program Evolution and its Impact on Software Engineering", Second International Conference on Software Engineering, IEEE Computer Society Press, pp. 350-355, October 13-15, 1976.

Large scale, widely used programs such as operating systems are never complete. They undergo a continuing evolutionary cycle of software maintenance, augmentation and restructuring to keep pace with evolving usage and implementation technologies. The paper provides quantitative evidence, from widely different environments, of the existence and nature of this evolutionary process. Interpretations and possible significance of some of the observed phenomena are discussed. Some implications for software engineering and for project planners and managers are noted.

Key words: environments, program evolution, project planners, project management, software engineering, software maintenance.

****

[LEHM77] M.M. Lehman, "Evolution Dynamics - A Phenomenology of Software Maintenance", Software Life Cycle Management Workshop Proceedings, pp. 313-323, August 1977.

The dynamics of evolution of large software systems has been extensively studied in recent years. This paper briefly reviews the phenomena observed and the main conclusions that follow. The reader is referred to the most recent references and to other contributions to the present Life Cycle Management Workshop for the data and derived models on which these conclusions and generalizations are based. Taken together, these results constitute a phenomenology that provides an essential ingredient of developing computer science.

Key words: evolution dynamics, phenomenology, lifecycle management, programming process, programming management, software maintenance.

****

[LERN83] J. Lerner, "Technique Eases Program Changes", Computerworld, p. SR74 and SR76, June 27, 1983.

The article suggests the use of a technique for easing and managing program changes. Configuration management is concerned with tracking changes in programs. It provides traceability and accountability for all changes, from project inception through the end of the product lifecycle. Attributes that should be included in an effective configuration management system are discussed. An example of how an on-line configuration management system fits into the lifecycle of a project is provided.

Key words: configuration management, productivity, program changes, project lifecycle, project management, quality assurance, software software maintenance.

****

[LETO86] S. Letovsky, E. Soloway, "Delocalized Plans and Program Comprhension", IEEE Software, vol. 3, no. 3, pp. 41-49, May 1986.

A maintainer's understanding can go awry when it is based on purely local clues. How can we spell out the intentions behind a piece of code? A maintainer must develop an understanding of a program if he is to carry out maintenance on it; the more complete and correct the understanding, the more likely that the modifications will be correct. When neither the program nor the documentation reveals that specific pieces of code interact with other pieces of code some distance away, the formation of a purely local understanding can lead to an inaccurate understanding of the program as a whole, which in turn can result in incorrect or inefficient program modifications. This article describes the reasons for comprhension failures in such situations, and goes on to suggest program documentation strategies that should aid maintainers in comprehending code implementing delocalized plans.

Key words: documentation, programmers, software maintenance, strategies, techniques.

****

[LIEN78a] B.P. Lientz, E.B. Swanson, and G.E. Tompkins, "Characteristics of Application Software Maintenance", <u>Communications of the ACM</u>, vol. 21, no. 6, pp. 466-471, June 1978.

Maintenance and enhancement of application software consume a major portion of the total lifecycle cost of a system. Rough estimates of the total systems and programming resources consumed range as high as 75-80 percent in each category. However, the area has been given little attention in the literature. To analyze the problems in this area a questionnaire was developed and pretested. It was then submitted to 120 organizations. There were 69 respondents. The results of the analysis indicate that: (1) software maintenance and enhancement do consume much of the total resources of systems and programming groups; (2) software maintenance and enhancement tend to be viewed by management as at least somewhat more important than new application software development; (3) in software maintenance and enhancement, problems of a management orientation tend to be more significant than those of a technical orientation; and (4) user demands for enhancements and extension constitute the most important management problem area.

Key words: productivity aids, software maintenance, management,

****

[LIEN78b] B.P. Lientz and E.B. Swanson, "Discovering Issues in Software Maintenance", <u>Data Management</u>, pp. 15-18, October 1978.

This article analyzes several important issues in software software maintenance. The authors reveal their findings from an experimental survey as well as the format of a more extensive survey that has been recently undertaken. Literature is then reviewed in the following five areas of software maintenance: conceptual issues (what does `software maintenance' mean), scale-of-effort issues, organizational issues, productivity technique issues (i.e. have the proper productivity techniques been adopted and do they enhance system maintainability), and problem area issues.

Key words: maintainability, organizational issues, productivity technique issues, scale-of-effort issues, scheduled software maintenance, software maintenance, software maintenance issues.

****

[LIEN79] B.P. Lientz and E.B. Swanson, "Software Maintenance A User/Management Tug-of-War", <u>Data Management</u>, pp. 26-30, April 1979.

This article describes issues that the authors researched in their survey. This survey was a 19-page questionnaire mailed to a random sample of 2000 DPMA members of which 487 or (24%) responses were received. The managers included in this survey were from the government and other industries. The answers given in the survey provided the authors with the following results : (1) the DP department spends half of its time on software maintenance which is contrary to some claims that the level of software maintenance has remained constant; (2) managers indicated that their departments were somewhat understaffed; (3) over 40% of the software maintenance work consist of providing users with enhancements that primarily include added volumes of data and reports. Managerial problems with regard to software maintenance, organizational controls, and future avenues for research are discussed. The authors conclude that since user enhancements constitute a significant portion of the software maintenance activity, and since management cites user demands for enhancements as the greatest of its software maintenance problems, therefore, increased attention to user related issues is warranted.

Key words: management, organizational controls, software maintenance, staffing, user enhancements, users.

****

[LIEN80a] B.P. Lientz and E.B. Swanson, Software Maintenance Management, Addison-Wesley publishing Company, 1980.

This book reports on the results of the most extensive public study of the software maintenance and enhancement of existing application software to date. Its objective is to contribute to the understanding of the software maintenance process, to identify and asses key issues, and to suggest future directions for management and research. The emphasis is exploratory, rather than definitive.

Key words: enhancements, software maintenance, software software maintenance issues, software maintenance management.

****

[LIEN80b] B.P. Lientz and E.B. Swanson, "Impact of Development Productivity Aids on Application System Maintenance", Database, pp. 114-120, winter-spring 1980.

The use of productivity aids has been widely advocated as a means of improving software quality, reducing software software maintenance effort and, in some cases, reducing total development costs. A variety of tools and techniques has been proposed for different activities. Some questions that arise are: How widely are such tools employed? Is software of better quality produced? Is the software maintenance effort ultimately reduced? During the past three years, the authors have been involved in studies of application software maintenance, the most recent of which provide some tentative answers to the questions raised above.

Key words: productivity aids, problem factors, software maintenance, software quality, tools, total system cost.

****

[LIEN81] B.P. Lientz and E.B. Swanson, Problems in Application Software Maintenance", Communications of the ACM, vol. 24, no. 11, pp. 763-769, November 1981.

This report analyzes the survey results of the problems of application software maintenance in 487 data processing organizations. It identifies six problem factors: user knowledge, programmer effectiveness, product quality, programmer time availability, machine requirements, and system reliability. The results are both surprising and informative.

Key words: application machine requirements, software maintenance problem factors, product quality, programmer effectiveness, programmer time availability, system reliability, user knowledge.

****

[LIEN83] B.P.Lientz, "Issues in Software Maintenance", Computing Surveys, vol. 15, no. 3, pp. 271-278, September 1983.

This article discusses some of the major issues and problem areas in software maintenance and suggests that the key reasons for research of software maintenance are that: (1) software maintenance consumes substantial hardware and software resources, (2) there is limited personnel availability due to a high turnover rate, and (3) present research has not addressed many of the problems associated with software maintenance. A survey of software software maintenance organization which identified five issues in software software maintenance: conceptual issues, measurement issues, the percentage of staff time spent on software maintenance, organizational issues, and productivity issues is described. According to the survey, the major problem areas were with user knowledge (59.5%) and programmer effectiveness (11.9%) Most of the severe problems were nontechnical. This article provides managers and users with comprehensive information on current problems and issues in software maintenance.

Key words: Conceptual issues, managers, measurement issues, personnel, productivity issues, software maintenance, tools, turnover rate, users.

****

[LIPO79] M. Lipow, "Prediction of Software Failures", The Journal of Systems and Software, vol. 1, no. 1, pp. 71-75, December, 1979.

The article addresses the cost of errors during certain stages of the lifecycle. It compares the costs, types, and causes of software errors during the requirements, design, and coding phases. Errors detected in design phase are compared against errors detected in the test phase. Research in this area indicates that (1) since more errors originate during requirements and design phases than during the coding phase, more effort should be spent in requirements and design validation; (2) tools are less effective than practices and procedures for preventing or detecting faults; and (3) if the cost of developing a tool or technique is low enough, it should be used to detect errors in the requirements or design phases.

Key words: coding phase, design phase, error costs, requirements phase, software errors, software failures, software lifecycle, test phase, tools.

****

[LITT80] B. Littlewood, "Theories of Software Reliability: How Good Are They and How can They Be Improved?", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-6, no. 5, pp. 489-500, September 1980.

An examination of the assumptions used in early bug-counting models of software reliability shows them to be deficient. Suggestions are made to improve modeling assumptions and examples are given of mathematical implementations. Model verification via real-life data is discussed and minimum requirements are presented. Example of how these requirements may be satisfied in practice are provided. The author suggests that current theories are only the first step along what threatens to be a long road.

Key words: debugging software, program error, reliability growth, software bug count, software failure, software failure rate, software lifecycle cost, software reliability measurement.

****

[LIU76] C.C. Liu, "A Look at Software Maintenance", Datamation, vol. 22, no. 11, pp. 51-55, November 1976.

This article describes some of the major problems in the software maintenance field as well as recommendations for rectifying these problems. The author states the contradiction of high software maintenance costs with no recognition of its importance. A list of the software maintenance function are included. The author summaries the software maintenance process as:

1. The capacity, function, and logic of the existing program or system must be understood throughly.
2. New logic to reflect the new request or additional feature must be developed.
3. The new logic must be incorporated into the

existing logic.

The author recommends that organizations:

a. start the documentation activity the moment the project is initiated;

b. document not only 'what has been done', but 'why it has been done';

c. consider management policies, practice and plans as part of the documentation; and

d. when documenting, consider the document users.

Key words: documentation, software maintenance programming, management, personnel, program logic, programmers, systems software maintenance, systems testing, software maintenance.

****

[LOVE77] T. Love, "An Experimental Investigation of the Effect of Program Structure on Program Understanding", General Electric Technical Information Series, Internal report, TIS-771SP006, GE Class 1, April 1977.

An experimental design was used to test the effect of two variables on program understanding. The independent variables were complexity of control flow and paragraphing of the source code. Understanding was measured by having the subjects memorize the code for a fixed time and reconstruct the code verbatim. Also, some subjects were asked to describe the function of the program after completing their reconstruction. The two groups of subjects for the experiment were students from an introductory programming class and from a graduate class in programming languages.

The major findings were:

that paragraphing of the source had no effect for either group of subjects;

that programs with simplified control flow were easier for the computer science students to understand;

that the dependent variable, rated accuracy of their description of the programs's function, did not differ as a function of either independent variable.

Key words: structure, psychological complexity, reliability, software psychology, structured programming.

****

[LYON81] M.J. Lyons, "Salvaging your Software Asset (Tools Based Maintenance)", AFIPS 1981 National Computer Conference Proceedings, pp. 337-342, May 4-7, 1981.

Software is a valuable asset embodying processes of an organization and contributing directly to the means of production. Maintenance is the mechanism for combating deterioration of the software asset, which over time tends to become inflexible. Though extremely costly, maintenance is essential to insuring the viability of the organization. Both rewrites and purchased software, with ensuing conversions, are usually not a cost-effective solution to software decay. Structured retrofit is an effective alternative, using a software tools-based methodology for combating decay and the high costs of software maintenance. The critical tool is the COBOL structured programming engine. This tools mechanically transforms spaghetti code to well-structured programs, whose ongoing software maintenance reaps the benefits of the structured programming methodologies.

Key words: COBOL structured programming engine, software maintenance, structured methodologies, structured retrofit.

****

[MAJO84] M. Major, "Can Software Vendor Support Really Help?", Software News, pp. 51-53, May 1984.

This article discusses different approaches to vendor support. In the introduction the author states some reasons why todays end-users need vendor support. One of the reasons for an increased need of vendor support is due to the arrival of sophisticated software, which allows for more complex usage of personal computers. Also, at a Softcon Convention, 95% of the dealers claimed that 'difficulty of use' was the most frequent complaint about software products. Some of the different types of vendor support are as follows:

1. Training center concept - analyzes DP needs, purchases, and sets up the equipment, and trains all users of the prescribed selected system.

2. Tutorials - allows the user to learn about the product through testing.

3. Telephone support - technicians will answer the user's questions about the package.

This paper provides end-users with some useful advice in the area of vendor support.

Key words: end-users, personal computers, software packages, training, tutorials, vendor support.

****

[MALM77] A.F. Malmberg, "Maintenance for the NET-2 Network Analysis Programs", BDM Corporation, April 1977.

Maintenance activities for the release 9 version of the NET-2 Network Analysis Program are described. These activities include correction of logical and programming errors. Improvement of numerical techniques as required, and provision of engineering assistance and consultation in application of NET-2 as directed by the Harry Diamond Laboratories.

Key words: electronic circuit analysis, network analysis, programming errors, software maintenance, system analysis.

****

[MAMR77] S. Mamrak and J.M. Randal, "A Analysis of a Software Engineering Failure", <u>The Computer Journal</u>, vol. 20, no. 4, pp. 316-320, November 1977.

Historical analyses of software engineering projects that fail are rare commodities in the computer science field today. This paper describes the rise and fall of a professionally engineered project that was doomed to failure by strategical, technical and tactical errors which were clearly revealed by analytic hindsight.

Key words: software engineering, software engineering failure, software engineering projects.

****

[MANC83] P. Manchester, "The Programming Paradox", <u>Computer Management</u>, pp. 39-42, June 1983.

This article analyzes various methods for increasing productivity in software maintenance. The author compares different methods for organizing a data processing department. Advantages and disadvantages of two of these methods, the applications-oriented team and the project-oriented team, are given. Certain procedures are recommended for use in the design and implementation phase in order to ensure that the system will be easily maintainable. The author discusses new technology to aid software maintenance programmers. The use of network systems have led to remote support service for software support. This service enables the software maintenance staff to dial into a user installation and obtain relevant information (i.e., dumps, user displays and program code) for debugging a software problem.

Key words: design phase, implementation phase, maintainability, network systems, productivity, software maintenance.

****

[MANT81] M. Mantei, "The Effect of Programming Team Structures on Programming Tasks", Communications of the ACM, vol. 24, no. 3, pp. 106-113, March 1981.

The literature recognizes two group structures for managing programming projects: Baker's chief programmer team and Weinberg's egoless team. Although each structure's success in project management can be demonstrated, this success is clearly dependent on the type of programming task undertaken. Here, for the purposes of comparison, a third project organization which lies between the other two in its communication pattern and dissemination of decision making authority is presented.. Recommendations are given for selecting one of the three team organizations depending on the task to be performed.

Key words: chief programmer team, group dynamics, project management, software engineering.

****

[MARS83a] N.L. Marselos, "Human Investment Techniques for Effective Software Maintenance", AFIPS 1983 National Computer Conference Proceedings, pp. 131-136, May 1983.

This paper presents methods for improving the maintenance of software by addressing the psychological issues that impact on software maintenance personnel. The emphasis in this paper is on making software maintenance developers more effective through goal setting, by using team-building approaches, through support personnel, and by using skill profiles to plan for their technical growth.

Key words: goal setting, software maintenance, support personnel, team-building.

****

[MARS83b] R.E. Marsh, "Application Software Maintenance: One Shop's Experience and Organization", AFIPS 1983 National Computer Conference Proceedings, pp. 145-153, May 1983.

Several years of data on software support activity at Dow Corning are analyzed to illustrate the problems of managing this function. Most software changes are small from the user's point of view, but few changes are small from the software maintenance point of view. Several organizational models have been tried for the management of support. None were entirely successful.

Key words: software maintenance, software support management.

****

[MART79] G.N. Martin, "EDP Systems Maintenance", Journal of Systems Management, pp. 18-21, September 1979.

This article analyzes several factors which influence systems software maintenance. It provides a list of the characteristics of software maintenance management which includes such things as: scheduling system modification, assigning work, and controlling a large number of projects. The author also discusses methods of organizing systems software maintenance and suggests that:

-the best method of organization is to set-up two groups, (software maintenance and development) each with its own management;

-the prerequisites for systems software maintenance, include long-range planning and software maintenance project justification.

-the project should consist of five phases: project initiation, survey, construction (analysis, design, and development), implementation, and a final evaluation prior to production acceptance.

A list of performance standards and controls for the software maintenance function is provided.

Key words: management, performance standards, planning, project management, software maintenance, systems software maintenance.

****

[MART82] J. Martin, Application Development Without Programmers, Prentice-Hall, 1982.

The author describes various methods that aid in the development of data processing applications. The following topics are discussed: the programming dilemma, software for application development (without the use of conventional programming), productivity, problems with conventional application development, changes needed in DP management, the role of data bases, the changing role of the system analyst, preprogrammed application packages, application generators, and end-user computing. The author recommends:

1. A thorough logical data-base analysis.
2. A well-designed data-base environment.
3. Use of high-level application generators, query languages, and report generators rather than

COBOL or PL/1. The application-creation facilities should be chosen such that they are self-documenting and applications can be modified quickly and easily.

4. Avoidance of hardware configuration that may look cost-effective, but cause conversion problems.
5. Redevelopment of old applications with application generators to eliminate the need for COBOL and PL/1 reprogramming.

Key words: development, application generators, application packages, database environments, end-user computing, productivity, query languages, software management.

****

[MART83] J. Martin and C. McClure, Software Maintenance- The Problems and Its Solutions, Prentice-Hall, 1983.

This book presents an overview of the software maintenance problem and offers ideas and tools for rectifying it. The authors describe the software maintenance problem, define and categorize software maintenance, discuss the underlying causes of software maintenance, characterize and measure maintainability, and examine methods for building maintainability in software. They suggest that much of the software maintenance problem arises from the historical tendency to subordinate 'data' to 'process' in software systems. An indepth discussion of data systems, data normalization, and data modeling is provided. They recommend the use of fourth-generation languages for solving the software maintenance problem. Their conclusion is that users can create their own software using these languages.

Key words: data modeling, data normalization, fourth- generation languages, maintainability.

****

[MART84] R.J. Martin and W.M. Osborne, "The Ideal Maintainer: A Profile", Data Management, pp. 39-40, March 1984.

This article explains some of the characteristics and personality traits which are necessary for a successful software maintenance programmer. A programmer must be: flexible, self-motivated, responsible, creative, disciplined, analytic, thorough, experienced, and knowledgeable about the existing system. Recommendations for management of software maintenance environments are also provided.

Key words: maintainer, personality, software maintenance programmers, software maintenance.

****

[MAUL83] C.R. Maule, "Clearing Up Software Chaos", Systems and Software, pp. 125-129, October 1983.

Configuration Management (CM) keeps track of changes in the source code that typically follow initial development. The CM system must preserve every change made because such information is vital for tracking errors, auditing progress, and suggesting future changes. The article introduces Vista (VOS Integral Source Tracking and Analysis), a software system that uses a different approach than previous CM systems. The author discusses the advantages of Vista over previous CM systems. The present CM systems either store copies of the source files or create special database files that store only code changes. Since the information is never stored in formats that can be read by operating systems, programmers must remove the files from the database to update, debug or document the contents. This approach removes the files from CM control and is particularly error prone since programmers often ignore or misuse the cumbersome procedures. A detailed discussion of the VISTA system is provided.

Key words: configuration management, errors, files, operating system, programmers, VOS Integral Source Tracking and Analysis (Vista).

****

[MCCA76] T.J. McCabe, "A Complexity Measure", Second International Conference on Software Engineering, IEEE Computer Society Press, October 13-15, 1976.

This paper describes a graph theoretic complexity measure and illustrates how it can be used to manage and control program complexity. The paper first explains how the graph theory concepts apply and gives an intuitive explanation of the graph concepts in programming terms. The control graphs of several actual Fortran programs are then presented to illustrate the correlation between intuitive complexity and the graph theoretic complexity are then proved which show, for example, that complexity is independent of physical size (adding or subtracting functional statements leaves complexity unchanged) and complexity depends only on the decision structure of a program. A characterization of non-structured control graphs is provided and a method of measuring the "structuredness" of a program is developed. The relationship between structure and reducibility is illustrated with several examples.

Key words: Complexity measure, control flow, decomposition, graph theory, independence, linear, modularization, programming, reduction, software testing.

****

[MCCA77] J.A. McCall, P.K. Richards, and G. Walters, "Factors in Software Quality", General Electric Technical Information Series, June 1977.

The objective of the study was to establish a concept of software quality and provide an Air Force acquisition manager with a mechanism to quantitatively specify and measure the desired level of quality in a software product. Software metrics provide the mechanism for the quantitative specification and measure of quality. Volume I, Concept and Definitions of Software Quality, describes the process of developing our concept of software quality and what the underlying software attributes are that provide the quality, and defines the metrics which provide a measure of the degree to which the attributes exist. Volume II, Metric Data Collection and Validation, describes the application of the metrics to software products and the validation of the metrics' relationship to software quality. Volume III. Preliminary Handbook on Software Quality for an Acquisition Manager, is a preliminary stand-alone reference document to be used by an acquisition manager to implement the techniques established during the study.

Key words: quality assurance, software development, software metrics, software quality.

****

[MCCA83] J.A. McCall and M.A.Herndon, "Quality Assessment: A Missing Element in Software Maintenance", COMPSAC Proceedings, IEEE Computer Society Press, pp. 87-88, November 7-9, 1983.

The authors explains the need for software metrics in software software maintenance. Software measurements used in the development of software should also be used in the maintenance of software. If a baseline set of metrics is used to describe the current software, as the software is modified, the value of these metrics will change. These changes can assist management and programmers in assessing whether the modifications made have improved or degraded the quality of the software. This provides management with a useful technique for establishing quality control during software maintenance. A basic set of measurements used to perform this quality assessment are listed. The authors suggest that software metrics can reduce future software maintenance cost, provide insight to management on the quality of software and are a good source of information to the software maintenance programmer.

Key words: management, metrics, programmers, software maintenance, software measurements, software metrics, software quality.

****

[MCCL81] C.L. McClure, Managing Software Development and Maintenance, Van Nostrand Reinhold, 1981.

This book presents practical project management methods that focus on the management and software maintenance issues of the software lifecycle. Principles and methods for controlling rising software costs by the application of engineering principles and methodologies for software maintenance are presented, as are procedures for building and preserving the quality of maintainability into a software system. A number of case studies of actual projects have been included.

Key words: maintainability, management, project management, software, software development, software engineering, software lifecycle, software maintenance.

****

[MCCO84] P.R.H. McConnel and W.B. Strigel, "Results of Modern Software Engineering Principles Applied to Small and Large Projects", AFIPS 1984 National Computer Conference Proceedings, vol. 53, pp. 273-281, May 1984.

This paper discusses the software development environment, tools, techniques, and methodology as applied in two mediums to large real-time software projects. Both quantitative and qualitative measures of success obtained in these projects are discussed. The qualitative measures are statistics representing the size of produced code. the manpower over the project lifecycle, and other data relevant to software engineering management. The qualitative evaluation is more concerned with results obtained from walkthroughs and various aspects of the applied methodology. Results are compared with those reported in the literature. Recommendations and suggestions for further improvements are presented.

Key words: software development environments, software engineering, software projects, tools.

****

[MCGR73] B. McGregor, "Program Maintenance", Data Processing, vol. 15, no. 3, pp. 172-174, May-June 1973.

This article addresses the difficulties faced by the DP manager in handling software maintenance personnel and activities. Senior personnel feel software maintenance programming is downgrading and are more interested in development projects. Junior programmers do not have the experience to handle difficult software maintenance projects. This puts the DP manager in a terrible situation. The author suggests that the solution is to use consultant programmers to handle the software maintenance task. Advantages of using consultant programmers are presented and suggested as an alternative to the DP manager, faced with persistent problems due to an inadequately skilled staff or to the poor image of software maintenance.

Key words: programmers, software maintenance.

****

[MCKE84] J. McKee, "Maintenance as a Function of Design", AFIPS 1984 National Computer Conference Proceedings, vol. 53, pp. 187-193, May 1984.

Changing one's point of view on the software maintenance function can lead to a better understanding of the relationship between software maintenance and other aspects of software products. This can lead to an improved allocation of effort when building software products.

Key words: database management systems, maintainability, software maintenance, software maintenance costs, software lifecycle.

****

[MERR78] S. Merritt, "Standard Definitions: A Missing and Needed Software Tool", 17th Annual ACM/NBS Technical Symposium Proceedings, pp. 193-198, June 15, 1978.

This article attempts to give clear and concise definitions to such terms as software maintenance, conversion, and documentation. For each term, alternative definitions are given before the final definition is stated. Software maintenance is defined as any work done on software after its first production of user output, except conversion. This author provides yet another view of what constitutes software maintenance.

Key words: conversion, documentation, end-users, modification, software maintenance.

****

[MIAR83] R.J. Miara, J.A. Musselman, J.A. Navarro, and B. Shneiderman, "Program Indentation and Comprehensibility", Communications of the ACM, vol. 26, no. 11, pp. 861-867, November 1983.

The consensus in the programming community is that indentation aids program comprehension, although many studies do not back this up. The authors tested program comprehension on a Pascal program. Two styles of indentation were used-blocked and non-blocked in addition to four possible levels of indentation (0,2,4,6 spaces). Both experienced and novice subjects were used. Although the blocking style made no difference, the level of indentation had a significant effect on program comprehension. (2-4 spaces had the highest mean score for program comprehension). The authors recommend that a moderate level of indentation be used to increase program comprehension and user satisfaction.

Key words: indentation, maintainability, Pascal, program comprehension, programmers, programming.

****

[MILL77] C.R. Miller, "Software Maintenance and Lifecycle Management", Software Lifecycle Management Workshop, IEEE Computer Society Press, pp. 53-61, August 1977.

This paper discusses the importance of software maintenance in lifecycle management and why the software maintenance phase constitutes a major portion of the cost of software. Software maintenance problems are identified and their causes are discussed. Some of the key problems identified are: tasks involved in software maintenance have not been defined, skills have not been taught and exact values to measure accomplishment are not available. Factors that contribute to readability and modularity are presented and new methods in software development that help to alleviate the software maintenance problem are discussed. The author suggests that software maintenance must be given higher priority within organization objectives.

Key words: lifecycle management, modularity, readability, software development, software maintenance.

****

[MILL83] H.D. Mills, <u>Software Productivity</u>, Little, Brown and Company, 1983.

This is a collection of 19 published papers and internal IBM memoranda written between 1967 and 1981. These articles present a good foundation of software engineering to the reader a foundation in both senses of the word in that the articles provide the historical and fundamental underpinnings of the field. The earliest articles are from the era when "software engineering" was only a clever phrase denoting a problem area. These discussions and their suggestions for development of an engineering discipline were on target and are highly recommended for those in the field today.

Key words: productivity, program complexity, programming, software development, software engineering, software productivity, software quality, structured programming.

****

[MILS81] J.H. Milson, "Automating Documentation Aids in Software Maintenance", Data Management, pp. 15-17, April 1981.

This article discusses a documentation tool as an aid for software maintenance. The automated documentation system (ADS) generates system and program documentation from both the source code of the programs and the JCL (Job Control Language). Since the purpose of documentation is to assist the software maintenance staff in performing its duties, it is essential that documentation be accurate, adequate and current. Proper documentation can lessen the time for a new programmer to become productive, thereby reducing the cost of software maintenance. The author describes three different types of system and program modifications (emergency, routine, and enhancements) along with the quality and quantity of the documentation for each of these modifications. ADS helps establish standards and documents the system in a systematic and consistent fashion. A detailed description of ADS and the macro-level and micro-level reports are provided. Macro-level information consists of: programs in the system, and files in the system, while micro-level information consists of data element definitions.

Key words: automated documentation system (ADS), automation, documentation, enhancements, program documentation, program modifications, software maintenance, tools.

****

[MOON75] J.W. Mooney, "Organized Program Maintenance", Datamation, vol. 21, no. 2, pp. 63-64, February 1975.

This article demonstrates for management new possibilities for organizing and handling software maintenance personnel. Differences between the old way of solving software maintenance problems and the new improved methods are compared. Innovations put into use include: (1) a team of senior and junior programmers to work on software maintenance only, (2) a project manager to organize the team and to ensure that standards were met, (3) explain to programmers that their task is more important than any other programming function, and (4) assure team members that they will always receive the largest merit increase allowed by company policy. All of these changes aided in keeping high morale among these programmers. The result was a large decrease in software maintenance time and a significant increase in time spent on new development. An excellent article for DP managers, it provides valuable information for handling DP personnel.

Key words: maintenance personnel, management, organization, programmers, project managers, software maintenance, standards.

****

[MUNS81] J.B. Munson, "Software Maintainability: A Practical Concern for Life-Cycle Costs", Computer, vol. 14, pp. 103-109, November 1981.

One of the major problems in software maintenance is the modification of software which is difficult to maintain. This article recommends different methods and techniques for creating software that facilitates software maintenance. The main theme of this report is that modifiability must be built into the software from the start. This implies that software modifiability must be planned during the development phase. Techniques for ensuring modifiability during the development phase are discussed. These techniques are as follows: (1) establish a policy so that certain standards are upheld, (2) develop the requirements specification document to define a baseline for modifiability and to discuss system expandability, (3) design flexible user interfaces, (4) establish guidelines for building modifiable computer programs and modules, and (5) develop a dictionary to control and handle data. Post-delivery techniques to be used during operation and software maintenance are also described.

Key words: contract, development phase, documentation, maintainability, modifiability, software maintenance, standards, user interfaces.

****

[MUNS82] J.B. Munson, "Plan Software Maintenance Now; Save Time and Dollars Later", Data Communications, vol. 11, pp. 103-107, June 1982.

The planning for future software maintenance early in the software development cycle is an important step for reducing the cost and effort involved with software maintenance. This article describes a software maintenance plan which requires a formal understanding between the users and the software developers. It also addresses the need to identify guidelines and procedures in software contracts for software modifications and programming techniques. Modifiability must be explicitly planned by the employer and the contractor, and then managed from start to finish. The author suggests that the contract should include both a software modifiability development plan and a software modification support plan. The former plan should define design techniques, set programming standards, and specify management controls. The latter plan should contain the post-delivery resources to support computer program modifications (i.e., test facilities, support and test tools, and configuration-management procedures). Upon acceptance of the final product, the author recommends that the developer should be required to deliver all development tools, test materials, and software maintenance documentation. Each plan is discussed in detail.

Key words: modifications, modifiability plan, lifecycle, software maintenance, software maintenance plan, software maintenance costs, support plan.

****

[MURP82] W. Murphy, "Documentation: Writers/Editors Vital", MIS Week, p. 23, December 22, 1982.

This article identifies problems with documentation and offers some innovative ideas to solve these problems. The author defines documentation as that which supports a system or process either as a learning or reference tool that is useful, accurate, concise and current. Some of the suggested solutions include: the assignment of an editor who would have responsibility for providing information on system documentation; the use of word processors and diskettes to eliminate the heavy reliance on paper would not only save space and money but would also be easier to maintain and understand; the establishment of a publications group, responsible for documentation and user information, to support systems development and software maintenance.

Key words: diskettes, documentation, editors, software maintenance, word processors.

****

[MURR83] J.P. Murray, "Separating Maintenance From Development Function", Computerworld, pp. 51-52, November 21, 1983.

The main theme of this article is that the software maintenance and development functions should be separated. The author recommends that the word "software maintenance" be eliminated from our vocabulary due to its negative connotations. Its use degrades the importance of the work done to improve operational systems. The following reasons are given:

1. Until these two functions are separated the development phase will receive more attention and respect. Workers will believe that the judgement of their performance is connected to development projects causing a lack of interest in software maintenance.

2. Due to major differences in each function, the work in one function can cause distractions in the other function.

3. It enables management to focus on one area.

4. If the two groups can work together, they can improve the performance of the entire system.

5. The development group benefits because once the software maintenance group accepts the system they are no longer responsible for it.

Key words: development, management, software maintenance,

****

[MURR84] J.P. Murray, "Maintenance Considerations", <u>System Development</u>, vol. 4, no. 5, pp. 4-6, July 1984.

In DP management, the development of new systems receives the most attention. Formal methods have been established for managing new development projects. However, in the software maintenance function, which consumes 50% to 70% of the total systems and programming effort, the use of some type of management control is often lacking. This article describes an approach for the management of systems and programming software maintenance. The key steps are as follows: (1) separate software maintenance from development within the DP department. This will assist management in achieving the goal of creating an environment where the software maintenance of operational programs has the same level of importance as the development of new systems. (2) develop a method which can be used to identify and control the software maintenance workload. This would include the installation of a formal process for requesting work on a particular program or system (i.e., completing a form for each request).

Key words: development, management, productivity, software maintenance.

****

[MUSH84] M. Mushet, "Life After Implementation: Managing System Maintenance", <u>Journal of Information Systems Management</u>, vol. 1, no. 2, pp. 55-65, Spring 1984.

This article gives an overview of the software maintenance field. A discussion of system life and how systems can wear out or become deficient is presented. The author suggests that the key to prolonging system life is to improve system software maintenance. The different phases of the system lifecycle - analysis, design, construction, implementation, and software maintenance - and the effects of the first three phases on future software maintenance are also described. Some of the techniques used in the software maintenance phase to improve system effectiveness are identified. Steps typically involved in the software maintenance process are as follows: (1) promote a positive attitude among software maintenance workers, (2) implement equal pay scales for development and software maintenance workers, (3) ensure that there are a wide variety of skills in the software maintenance group, and (4) require the involvement of users in software maintenance tasks. Recommendations for a successful software maintenance program are also provided.

Key words: analysis, construction, design, implementation, lifecycle, software maintenance.

****

[MYER78] G.J. Myers, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections", Communications of the ACM, pp. 760-768, September 1978.

This paper describes an experiment in program testing, employing 59 highly experienced data processing professionals using seven methods to test a small PL/1 program. The result show that the popular code walkthrough/inspection method was as effective as other computer-based in finding errors and that the most effective methods (in terms of errors found and cost) employed pairs of subjects who tested the program independently and then pooled their findings. The study also shows that there is a tremendous amount of variability among subjects and that the ability to detect certain types of errors varies from method to method.

Key words: code inspections, code walkthroghs, debugging, errors, program verification, software reliability, testing, testing techniques.

****

[MYER79] G.J. Myers, The Art of Software Testing, John Wiley and Sons, 1979.

This book describes software testing techniques. The text includes discussions of various testing techniques and checklists of errors to look for during activities such as walkthroughs and inspections. Some of the other topics covered in this book are: the testing of modules, subroutines, and systems, tools for testing, how to write effective test cases, and program debugging. In the chapter on program debugging, the author discusses various methods of debugging, debugging principles, and performing an error analysis to improve subsequent programming efforts.

Key words: debugging, error analysis, errors, inspections, testing, test cases, testing, testing tools, walkthroughs.

****

[NARR84] B. Narrow and J. Kelly, "Two Perceptions of Software Maintenance Performed by an On-Site Contractor", AFIPS 1984 National Computer Conference Proceedings, vol. 53, pp. 235-242, 1984.

This paper describes lessons learned by a customer and an on-site contractor. Software software maintenance is a difficult task under the best of circumstances. Having work performed by an on-site contractor adds a additional layer of complexity to the customer's task. This type of relationship places greater emphasis on formal work procedures and detailed reports of the work in progress. It also promotes the use of performance norms for evaluating contractor performance. These factors are all on the positive side. However, such a relationship also calls for a special awareness of contractor ploys calculated to increase their performance evaluation.

From the contractor's point of view, being on-site imposes a more disciplined environment and places special importance on the manner and means of dealing with the customer. Another special feature is that the contractor receives formal feedback from the users, through periodic performance evaluations, indicating how well the software maintenance group measures up to expectations.

Key words: management, performance evaluations, performance metrics, personnel, software development, software maintenance.

****

[NAVE79] "Computer Software Life Cycle Management Guide", Naval Electronic Systems Command, March 1, 1979.

The purpose of the Computer Software Life Cycle Management Guide (CSLCMG) is to provide information to the NAVELEX program manager (PM) to allow him to understand and be able to meet sound software development practices for a NAVELEX software system acquisition. The objectives of the CSLCMG are to:

- o Provide the PM with detailed information that will enable him to manage the lifecycle software acquisition process.
- o Provide the PM with insight into software engineering methods so that he can understand and be able to require their implementation by a software contractor.
- o Provide management techniques whereby the PM can ensure high visibility of the software development process.
- o Make innovative technical and management contributions, based on major lessons learned from previous software development projects.

The CSLCMG provides amplifying procedures for carrying out the policy and practices specified in NAVELEXINST 5200.22. The CSLCMG is the first of a series of NAVELEX Computer Software Acquisition Management Guidebooks for program managers.

Key words: lifecycle management plan, management, software acquisition, software development.

****

[NBS74] R. Houghton, "Features of Software Development Tools", NBS Special publication 500-74, February 1981.

Software tools are powerful productivity and quality aids that in many cases are not being used effectively. This report discusses an effort to lessen this problem by providing a formal way in which tools can be classified according to the features that they provide.

Key words: productivity, software development, software development tools, software quality, tools.

****

[NBS98] P.B.Powell, Editor, "Planning for Software Validation, Verification, and Testing", NBS Special Publication 500-98, November 1982.

This document is for those responsible for directing and implementing computer projects. It addresses the process of obtaining increasing levels of confidence in a solution through a series of checkpoints and reviews during software development. It discusses the selection and use of validation, verification, and testing tools and techniques for software development. It also explains how to develop a plan to meet specific software validation, verification, and testing goals.

Key words: automated tools, lifecycle, software verification, test coverage, test data generation, testing, validation.

****

[NBS99] T. McCabe, "Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric", NBS Special Publication 500-99, December 1982.

Various applications of the Structured Testing methodology are presented. The philosophy of the technique is to avoid programs that are inherently untestable by first measuring and limiting program complexity. Part 1 defines and develops a program complexity measure. Part 2 discusses the complexity measure in the second phase of the methodology which is used to quantify and proceduralize the testing process. Part 3 illustrates how to apply the techniques during software maintenance to identify the code that must be retested after making a modification.

Key words: measures, metrics, program complexity, software maintenance, software testing, structured testing.

****

[NBS106] R.J. Martin, W.M. Osborne, "Guidance on Software Maintenance", NBS Special Publication 500-106, December 1983.

This report addresses issues and problems of software maintenance and suggests actions and procedures which can help software maintenance organizations meet the growing demands of maintaining existing systems. The report establishes a working definition for software maintenance and presents an overview of current problems and issues in that area. Tools and techniques that may be used to improve the control of software maintenance activities and the productivity of a software maintenance organization are discussed. Emphasis is placed on the need for strong, effective technical management control of the software software maintenance process.

Key words: adaptive software maintenance, corrective management, perfective software maintenance, software engineering, software maintenance, tools.

****

[NBS114] S. Frankel, Editor, "Introduction to Software Packages", NBS Special Publication 500-114, April 1984.

This document provides an introduction to applications software packages. It encourages the use of software packages as an alternative to in-house development and directs potential users of software packages to sources of useful information. Application areas which are currently supported by software packages are reviewed and the benefits of software package use versus in-house development are discussed. An annotated list of publications is also provided.

Key words: applications software packages, off-the-shelf software, software packages.

****

[NBS129] J.A. McCall, M.A. Herndon, W.M. Osborne, "Software Maintenance Management," NBS Special Publication 500-129, October 1985.

This report focuses on the management and maintenance of software and provides guidance to Federal government personnel to assist them in performing and controlling software maintenance. It presents an overview of the various aspects of software maintenance including the problems and issues identified during the ICST sponsored survey of Government and private industry maintenance organizations. Techniques, practices, tools, and procedures which aid in reducing these problems and which help to insure that quality software is developed for and by the Federal ADP community are identified. An integrated approach to software maintenance is described with suggestions for improving the software maintenance process. These suggestions provide a basis for improving both the software quality and the productivity of the organization.

Key words: cost control measures, decision aids, management, software configuration management, software maintenance management, software maintenance tools, software quality assurance, test plans.

****

[NBS130] W. Osborne, "Executive Guide to Software Maintenance," NBS Special Publication 500-130, December 1985.

This Guide provides answers to sixty-four key questions about software maintenance. It is designed for Federal executives and managers who have a responsibility for the planning and management of software projects. It is also intended for federal staff members affected by, or involved in making software changes and who need to be aware of steps that can reduce both the difficulty and cost of software maintenance.

Issues addressed in the Guide include the feasibility and applicability of software reuse, the development of maintainable software, as well as the improvement of existing software, achieving programmer and software productivity, and the three key attributes of maintainable software: correctness, understandability, and reliability. Finally, it discusses software tools that can aid in making existing code more maintainable.

Key words: maintainability, program structure, languages, software maintenance, standards.

****

[OGDI72] J.L. Ogdin, "Designing Reliable Software", Datamation, pp. 71-78, July 1972.

This article analyzes some of the problems and issues in designing reliable and maintainable software. The author identifies some common causes for the development of unreliable software. First, most computer programs are simply coded, not designed. A major cause of this problem is that programmers are taught a language without being taught good design principles and practices. Proper program design minimizes the effort required to accomplish an objective and returns more than the invested time and effort. It is estimated to be five times easier to change a well-designed program than a poorly coded one. Another cause of unreliable programs is the lack of discipline by both the program and system designer in adhering to a single design strategy. Two major problems in design strategy: designing a system to satisfy all perceived and projected needs is unrealistic and designing a system that tries to do too much are discussed.

Key words: communication, maintainability, program design, programmers, reliability, software maintenance, system design.

****

[OSBO84] W. Osborne, "A Framework for Improving Software Maintenance Throughout the Software Lifecycle, Third Software Engineering Standards Application Workshop Proceedings, IEEE Computer Society Press, pp. 137-138, October 2-4, 1984.

One of the problems in the software maintenance field is the lack of planning which it receives during the phases of the software lifecycle. This article addresses the issue by suggesting a framework for software maintenance throughout the lifecycle. This framework should consider the maintainability of software during each software lifecycle phase and should influence any decisions made during that phase. Software lifecycle requirements which should be included in this framework are:

1. ease of understanding the software
2. ease of making and controlling changes
3. use of improved techniques and tools
4. ease of using software
5. establishment of a quality assurance plan.

In addition, the software maintenance staff, upper management, and users should be involved in this framework throughout the software maintenance process. These three groups should jointly: (1) establish a centralized approval point, (2) ensure that proposed changes are not incompatible with the orginal system intent and design, (3) require that all software maintenance be performed with effective techniques, procedures, and tools, and (4) require that maintenance of routine changes be scheduled (if possible).

Key words: maintainability, management, quality assurance, software lifecycle, software maintenance, tools.

****

[OSBO86] W.M. Osborne, A.L. Hankinson, "Developing Federal Software Standards: A New Direction", Computer Standards Conference 1986 Proceedings (Addendum), IEEE Computer Society Press, May 13-15, 1986.

The software community has made great strides in the application of engineering methods and tools to improve software quality and to enhance the productivity of the software development process. Much of the credit for improvement in this area can be attributed to the development and use of software development standards. The improvements in software development are in stark contrast to the lack of progress in maintaining operational software.

This discussion describes efforts underway at NBS/ICST to use software standards as a framework for developing, acquiring, and maintaining software during its operation life. These standards:

1. have a software management focus;
2. build upon existing Federal, national, and international standards;
3. are integrated as part of a comprehensive policy, guidance, and training program.

Key words: productivity, software development standards, software quality, techniques, tools.

****

[OVER73] R.K. Overton, et.al., "Research Toward Ways of Improving Software Maintenance: Ricasm Final Report", CIRAD, January 1973.

This paper discusses research efforts to improve software maintenance. Studies of some fundamental aspects of the work indicate that (1) before a person can modify a program efficiently, he needs to be able to trace the structure into which the modification has to fit, and recognize the conceptual blocks of which the structure is built; (2) it should be possible to specify, and set some standards for maintainability-affecting features of programming languages, and the style and structure of programs; (3) physical characteristics of terminal displays can handicap or help the software maintenance programmer, and displays of lists of cues and probabilities may also help him. Future development, based on these points, were recommended.

Key words: maintainability, programmer, program structure, languages, software maintenance, standards.

****

[OVER74] R.K. Overton, et.al., "Developments on Computer Aided Software Maintenance", AMS Incorporated, September 1974.

Data were collected on two aspects of software maintenance programming (which, according to publish estimates, costs the U.S. approximately five billion dollars a year). Aspects were (1) arrangement and sources of information at graphics consoles, and (2) the value of "conceptual groupings" to software maintenance programmers using Fortran and PL/1. Research indicates that at consoles, there is a need for a better matching of problem-solving facilities for the level of abstraction or detail needed by the programmer. Scattered sources of needed information were a handicap, as were distraction and other factors. Pilot programs were developed to automate display of "conceptual groupings." In at least some cases, such programs decidedly improve software maintenance efficiency. Further development and wider usage of such programs is warranted.

Key words: computer aids, conceptual groupings, graphics consoles, Fortran, software maintenance.

****

[PARI80] G. Parikh, Techniques of Program and System Maintenance, Winthrop publishers, 1982.

This book is a compilation of important and useful material on software maintenance, published in the computer periodicals, conference proceedings, reports and books. The book is divided into seven sections. The first section introduces the problems of software maintenance and provides some perspective. The second section covers 'how to' aspects for a software maintenance programmer. Techniques for managing software maintenance are presented in the third section. The application and impact of structured technologies on software maintenance are described in section four. Section five, an extension on section four, indicates possible future developments in this vital area. It includes a chapter related to 'structuring engine,' a software package that automatically transforms an unstructured program into a structured program. Section six is an extensive, annotated bibliography, containing works on software maintenance, testing, structured technologies, tools, etc.

Key words: debugging, structured technologies, structuring engine, system software maintenance, tools.

****

[PARI81] G. Parikh, "Structured Maintenance - The Warnlier/Orr Way", Computerworld, pp. 11-18, September 21, 1981.

The Warnier/Orr methodology is very useful for designing, modifying, and documenting existing programs or systems. This article explains this methodology and reveals how it can be used for the software maintenance phase of the lifecycle. Warnier is the only person who provides guidelines for maintaining structured systems. His techniques can also be used for maintaining existing, unstructured systems. Warnier invented a procedural method for designing programs and systems called the Logical Construction of Programs (LCP) and the Logical Construction of Systems (LCS). The principal designing tool of his methodologies is the Warnier diagram. Orr added to the LCP notation developed by Warnier. The advantages of the Warnier/Orr methodology are as follows:

1. It helps design systems, programs and data files;
2. It is effective, efficient and economical both for development and software maintenance;
3. Any program can be documented with the Warnier/Orr diagram.

This report includes a case study of a company that chose Warnier/Orr diagrams for redevelopment.

Key words: design, documentation, Logical Construction of Programs (LCP), Logical Construction of Systems (LCS), software maintenance, structured software maintenance, Warnier/Orr diagrams, Warnier/Orr methodology.

****

[PARI82] G. Parikh, "Cost-Effective Software Maintenance", ICP Interface Data Processing Management, pp. 37-39, Summer 1982.

This article presents several methods for performing cost- effective software maintenance. The author suggests:

1. Buying software packages from reliable vendors with software maintenance contracts included, or using consultant/contract programmers to maintain the organization's software;

2. Developing maintainable software by utilizing an appropriate structured methodology, engineering principles, and proper development tools;

3. Keeping records of all software maintenance activities to assist in future software maintenance and to inform management where software maintenance money and effort is being spent;

4. Understanding and using Gerald M. Weinberg's 80/20 rule and his 'worst-first' approach for redesigning software (20% of the code causes 80% of the software maintenance effort therefore, problem code should be redesigned);

5. Using the following structured techniques in the software maintenance phase: programmer librarian, walkthroughs, structured programming techniques and structured documentation techniques.

Key words: documentation, software maintenance contracts, management, productivity, programmers, software packages, software maintenance, structured programming, tools, training.

****

[PARI83a] G. Parikh and N. Zvegintzov, "Managing Stability and Change: Software Maintenance Training", Data Training, pp. 11-12, May 1983.

This article provides an overview of software maintenance and discusses some approaches for training personnel in the software maintenance field. Statistics are provided on the cost, level, etc. of software maintenance. The laws of growth and evolution of large systems are discussed. Five elements of software maintenance training are identified as follows: (1) study techniques from other installations and research groups, (2) use software maintenance tools, (3) review your change methodology, (4) use a technical partnership workshop (i.e., maintainers working in different functions of the company brief each other on the technical content of their own assignments), and (5) develop a functional partnership consisting of software maintenance staff, operations staff, and users.

Key words: maintenance programmers, management, personnel, software software maintenance, software maintenance management, software maintenance tools, software maintenance training.

****

[PARI83b] G. Parikh and N. Zvegintzov, Tutorial on Software Maintenance, IEEE Computer Society Press, May 1983.

Software software maintenance, the work done on a software system after it becomes operational, consumes at least half of all technical and management resources expended in the software area. This Tutorial approaches software maintenance not only as an essential element in the life of a software system but also as a process with its own rules and techniques. Thirty-one papers by thirty-seven leading authorities on software maintenance papers are contained in this tutorial. This selection represents papers most often requested and papers in hard-to-find sources. Included in the tutorial are an introduction, an epilogue, an annotated bibliography, and name and topic indexes.

Key words: development, software maintenance activities, software evolution, software maintenance, software modification, software system, tools, understanding software.

****

[PARI84] G. Parikh, "Software Maintenance: Questions and Answers", Data Management, pp. 25-26, June 1984.

This article examines problems in maintaining microcomputer software and identifies differences between its software maintenance and the software maintenance of mainframe software. The author discusses some problems in the software maintenance of microcomputer software packages (i.e. answering customer questions and distributing updates to scattered customers) and offers electronic mail and software maintenance by telephone as solutions. Creating tools for software development and software maintenance on microcomputers, is briefly reviewed. Both the modifiability of the software and the method of software development should be considered when buying a software package.

Key words: electronic mail, mainframe software, microcomputer software, modifiability, software development, software maintenance, software packages, telesoftware maintenance.

****

[PARN72] D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", <u>Communications of the ACM</u>, vol. 15, no. 12, pp. 1053-1058, December 1972.

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and both a conventional and unconventional decomposition are described. Unconventional decompositions have distinct advantages for the goals outlines. The criteria used in arriving at the decompositions are described. The unconventional decomposition, if implemented with the conventional assumption that a module consists of one or more subroutines, will be less efficient in most cases. An alternative approach to implementation which does not have this effect is sketched.

Key words: KWIC index, modularity, modules, software design, software engineering.

****

[PARN79] D.L. Parnas, "Designing Software for Ease of Extension and Contraction", <u>IEEE Transactions on Software Engineering</u>, IEEE Computer Society Press, vol. SE-5, no. 2, pp. 128-137, March 1979.

Designing software to be extensible and easily contracted is discussed as a special case of design for change. A number of ways that extension and contraction problems manifest themselves in current software are explained. Four steps in design of software that is more flexible are then discussed. The most critical step is the design of a software structure called the "uses" relation. Some criteria for design decisions are given and illustrated using a small example. Identification of minimal subsets and minimal extensions can lead to software that can be tailored to the needs of a broad variety of users.

Key words: contractibility, extensibility, modularity, software design, software engineering, subsets, supersets.

****

[PATT81] M.B. Patterson, "Motivating Your Staff", Data Management, pp. 23-25, April 1981.

The lack of morale among maintenance programmers is a major problem for DP managers. This article offers recommendations on motivating DP workers and improving their morale. The findings of a 1979 report by Diebold Group, Inc. which shows the influence of management on programmer productivity are discussed. This report also states that productivity will increase if management can improve the attitudes and morale of DP workers. Five suggestions are explained for improving job satisfaction: task identify, skill variety, task significance, autonomy, and feedback. The author also describes the need of structure of the programmer/analyst and its importance for proper motivation.

Key words: autonomy, feedback, management, motivation, productivity, programmers, skill variety, task identity, task significance.

****

[PEER81] D.E. Peercy, "A Software Maintainability Evaluation Methodology", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-7, no. 4, pp. 343-351, July 1981.

This paper describes a conceptual framework of software maintainability and an implemented procedure for evaluating a program's documentation and source code for maintainability characteristics. The evaluation procedure includes use of closed-form questionnaires completed by a group of evaluators. Statistical analysis techniques for validating the evaluation procedure are described. Some preliminary results from the use of this methodology by the Air Force Test and Evaluation Center are presented. Areas of future research are discussed.

Key words: evaluation by questionnaire, evaluation reliability, quality metrics, software engineering, software maintainability evaluation, software quality assurance.

****

[PEER84] D.E. Peercy, "A Framework for Software Maintenance Management Measures", Seventeenth Hawaii International Conference on System Sciences, pp. 453-461, 1984.

Some of the important issues of problems of software maintenance management are discussed within the context of a proposed software maintenance framework. This framework consists of four elements: software products, software maintenance environment, software maintenance management, and software maintenance measures. Emphasis is upon the need for a data base of accurate measures to support the management decision process. The measures are used to determine which characteristics, techniques, tools, and requirements have the most effect on maintenance resource requirements and allocations. Elements of software product quality, software maintenance environments, and software maintenance activity are briefly discussed.

Key words: software maintenance, software maintenance environment, software maintenance management, software maintenance measures, software product quality.

****

[PENN80] R.H. Pennington, "Software Development and Maintenance-Where Are We?", COMPSAC Proceedings, IEEE Computer Society Press, pp. 419-422, 1980.

Current practice and state-of-the-art in software development and maintenance are discussed. A series of assertions is given which summarize the current status in the view of the author. Among the major assertions are:

- o Most software now in use or being written is unstructured and poorly documented.
- o Currently there is no structured programming language available.
- o Although most of the cost of software is in maintenance, there are no rewards for writing maintainable code.
- o We research on toy problems, but the world presents large, complex problems.

Key words: documentation, maintainability, software costs, software development, software maintenance, structured programming.

****

[PERL81] A. Perlis, F. Sayward, and M. Shaw, Software Metrics: An Analysis and Evaluation, The MIT Press, 1981.

Software metrics is an area of computer science that enables programmers or analysts to assign quantitative indexes of merit to software. This book surveys the field by measuring its present extent, describing its characteristic features and indicating directions of potential expansion. It contains fifteen articles on software metrics which were edited or written by the authors. These articles focus on the problems and methods in software metrics.

The role of statistical theory and application is explored in order to indicate how statistics can help guide future research on needed software maintenance tools. Areas in which research is needed are outlined, together with some pitfalls to be avoided in pursuing the research challenges.

Key words: lifecycle model, programmers, software maintenance tools, software maintenance statistics, software metrics.

****

[PERR84] W.E. Perry, "Software Must Last Another 20 Years - But How?", Government Computer News, p. 68, March 1984.

This article addresses problems in software maintenance and offers some suggestions on how to handle these problems. A GAO report is cited which states that maintenance is an unplanned, unbudgeted, and unmanaged function. According to the author, one of the major challenges for persons working in the software maintenance field in the 1980's and 1990's will be extending the life of existing applications. In addition, DP organizations are advised to develop action plans for dealing with system maintenance. The action plan should include these four tasks: (1) establish an inventory of operational application systems (so that organizations have good records of their software systems), (2) rate the portfolio of application software, (3) develop a maintenance policy, and (4) appoint an individual (i.e., manager) responsible for software maintenance.

Key words: maintenance policy, management, planning, software maintenance, software replacement cost.

****

[PETE77] D.R. Peterson, "Software Acquisition Management Guidebook: Software Development and Maintenance Facilities", Electronic Systems Division, Hanscom AFB, April 1977.

This document is one of a series of guidebooks covering important aspects of software acquisition. The guidebooks are prepared for use by Air Force program office personnel responsible for the management and planning of software development. This guidebook focuses on the management decisions and technical issues related to planning and acquisition of software development and maintenance facilities.

Key words: acquisition, computers, development, facility, hardware, maintenance, management, planning, software maintenance.

****

[PETE84] R.O. Peterson, "Maintenance Isn't Maintenance Anymore", <u>Computerworld</u>, pp. 27-36, June 11, 1984.

This article describes methods for improving the attitudes of workers towards software maintenance, including changing the term to `production support' rather than maintenance. The author warns managers about the placement of novice programmers in maintenance. The novice programmers learn bad traits from the old programs and, inevitably, have to be retrained. The differences between the personality traits of development and maintenance programmers are described. Since these two groups of programmers require such different personality traits, the author recommends that development and production support be separated. The methods, standards, and other factors associated with project turnover (transferring a project to maintenance) are also discussed. The following steps are suggested as a way to improve the effectiveness of the production support group: (1) the production support group must be given the responsibility and authority to keep the departmental goals intact; (2) workers in the production support group must be involved with the initial design of the system; (3) some programming standards should be enforced by the support group; (4) the career opportunities for the development and productions support groups should be the same.

Key words: maintenance, maintenance attitudes, maintenance programmers, personality traits, production support, programmers, programming, standards, project turnover, software maintenance.

****

[PIZZ84] A. Pizzarello, <u>Development and Maintenance of Large Software Systems</u>, Lifetime Learning publications, 1984.

This book provides guidelines, suggestions, and tools for developing and maintaining large software systems. The author defines 'large software systems' as associations of several individual programs which are expected to jointly achieve some global goals. These systems deal with complex problems, are of considerable size and require the cooperative labor of several people. Some of the topics covered in this book are as follows: program design and testing, hierarchical development and design, data abstractions, program specifications and techniques for understanding programs. Lastly, the author presents the most recent findings on special characteristics of large software systems.

Key words: data abstractions, development, large software systems, maintenance, program design, program specifications, program testing, software maintenance, techniques for understanding programs.

****

[PODO77] J.L. Podolsky, "Horace Builds a Cycle", Datamation, pp. 162-168, November 1977.

This article introduces an alternative to the classical system development cycle called the Recursive Development Cycle. Fictional characters are used to discuss the advantages and disadvantages of both lifecycles. Some of the problems with the traditional lifecycle are: users change their minds during the development, people responsible for the project's development often leave or get assigned to another project, and maintenance cost two to four times as much as 'development.' The classical system development cycle assumes things will only be built once which conflicts with the workings of the real world. The foundation for the Recursive Development Cycle is a hypothesis that the system development cycle should be constructed so that it recognizes that a system will probably require substantial, continuing change after the user begins live use of the system. In the author's new cycle, project planning does not end at installation. Since the cycle assumes that modifications will be made after the system is operational, the system must be constructed to facilitate future maintenance. Therefore, the author suggests using the following techniques: (1) use high level programming languages, (2) comment programs extensively, so it is clear what the program is doing, (3) structure programs following approved standards with careful use of modules and carefully defined interfaces, (4) use tables and files rather than embedded code, and (5) carefully document changes. The author concludes that this cycle will allow developers to become involved with the users, to control costs, and to provide high quality service over the entire life of the system.

Key words: development, lifecycles, maintenance, programming, project planning, recursive development cycle, standards, system development cycle, users.

****

[PRES81] L. Presser, "Reversing the Priorities", Datamation, pp. 208-210, September 1981.

This article presents a view that productivity should be measured by the quality of code and not by the number of lines produced (quantity). An example is provided which illustrates the cross-over point (the point at which programmers start resigning because more time is spent on maintenance than on developing new software) can be lengthened by increasing the quality of software. The relationship between software quality and software productivity is highlighted.

Key words: cross-over point, productivity, programmer productivity, quality, software productivity, software quality.

****

[PRES82] R.S. Pressman, Software Engineering: A Practioner's Approach, McGraw-Hill, 1982.

This book analyzes each step in the software engineering process. Early chapters present the planning phase, emphasizing system definition, software planning, and software requirements analysis. Specific techniques for software costs and schedule estimation are also included in these chapters. In subsequent chapters emphasis shifts to the software development phase. Later chapters address with software testing techniques, reliability, and the managerial and technical aspects of software maintenance.

Key words: coding style, reliability, requirements analysis, schedule estimation, software costs, software design methodology, software development, software engineering, software lifecycle, software maintenance, software testing, software tools.

****

[PUNT75] M. Punter, "Programming for Maintenance", Data Processing, vol. 17, no. 4, pp. 292-294, September/October 1975.

This article explores programming practices that facilitate software maintenance. The author discusses the importance of software maintenance to the organization and suggests sound practices that should be used during the system development stage. He also recommends the use of system diagrams and glossaries during the design stage. Since program listings are considered to be the maintenance programmer's primary tool, comments, blanks, and indentations should be included to improve the program's readability and comprehensibility. Maintenance should be considered during the development stage as well as when changes are incorporated.

Key words: development, design, maintenance programmers, program complexity, program listings, programming, readability, software maintenance, software quality.

****

[PUTN80] L.H. Putnam, Tutorial: Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers, IEEE Computer Society Press, August 1980.

All software projects exhibit lifecycle behavior. This tutorial reviews the nature of the cycle, explaining its characteristics and emphasizing the dominant influence of the independent variable of time. This tutorial presents a quantitative methodology of cost estimating, and discusses economics, trade-off opportunities, and investment strategies in a description of the managerial practices necessary for effective planning and control of software development.

Key words: economics, investment strategies, lifecycle, management, methodology of cost estimating, software.

****

[RAMA84] C.V. Ramamoorthy, A. Prakash, W. Tsai, and Y. Usuda, "Software Engineering: Problems and Perspectives", Computer, pp. 191-209, October 1984.

This article discusses aspects of the software engineering field and speculates on its trends and future needs. The following areas of software engineering are covered: software lifecycle, requirements and specification, software design, software testing, software maintenance, software quality assurance - which includes software reliability and metrics, software reusability, rapid prototyping, and cost estimation. Sources of maintenance problems are identified as: (1) insufficient or incomplete documents, (2) inconsistency between the documents and the code, (3) design difficult to understand, modify, and test, and (4) insufficient record of past maintenance. The authors offer three ways of reducing maintenance cost.

1. Develop the system with maintenance in mind.

2. Maintain the system with future maintenance in mind.

3. Upgrade the system to cope with future technology.

Key words: adaptive maintenance, corrective maintenance, maintenance, maintenance costs, perfective maintenance, preventive maintenance, rapid prototyping, requirements and specification, software design, software engineering, software lifecycle, software maintenance, software metrics, software quality assurance, software reliability, software reusability, software testing.

****

[RAY83] H.N. Ray, "A Planned Approach to Making the User Useful", Data Management, p. 18 and pp. 37-38, March 1983.

A major problem in the DP industry is the lack of user involvement in the design and development of application systems. This is an important issue in software maintenance because many problems result from a lack of communication with users and from users having a lack of understanding about their systems. The author recommends a Planned User Involvement Program to rectify this problem. This program is a continuous, 'forward-looking' organizational undertaking which must be totally integrated into the information systems environment. However, significant alterations to the traditional systems life cycle are needed for a successful program." In addition, effective user participation in systems development requires a minimal level of training and experience to reduce the fears which may inhibit successful user involvement. The eight phases in the planned user involvement systems lifecycle are as follows:

1. Periodic ongoing user indoctrination. This phase is the most crucial to the success of the program.
2. User identified problem definition.
3. Preliminary analysis and feasibility study. This is a team effort consisting of users, system analysts, and other specialists.
4. Appropriate, concentrated user training in analysis and design.
5. Analysis and design that is user directed, with an emphasis on evaluation and review.
6. Development and testing (includes user operation and approval).
7. User training and the development of user procedures and documentation.
8. Implementations and operations (team effort, user directed).

Educating users and involving them during system design will improve relations between user and the information systems department.

Key words: analysis and design, development, Planned User Involvement Program, user involvement, user training, users.

****

[RAYN83] R.J. Raynor and L.D. Speckmann, "Maintaining user Participation Throughout the Systems Development Cycle", AFIPS 1983 National Computer Conference Proceedings, pp. 155-161, May 1983.

Effective user participation is well known to be an important aspect of good system development methodology. Specific tools and techniques for managing user participation in all phases of the system development lifecycle are illustrated by a large business system development project at Texas Instruments. Emphasis is placed on maintaining a constant level of communication between user and developer as the system design evolves.

Key words: management, system development lifecycle, tools, user participation.

****

[REUT81] J. Reutter, "Maintenance is a Management Problem and a Programmer's Opportunity", AFIPS 1981 National Computer Conference Proceedings, pp. 343-348, May 4-7, 1981.

This paper defines the categories and characteristics of maintenance and points out the opportunities for management success and job enrichment in software maintenance. The author identifies the seven categories of maintenance along with the estimated cost for each category. According to the author, fixing bugs constitutes between 8-15% of the total maintenance costs. A number of software maintenance management issues are addressed including: the need to require cost benefit analysis and tradeoffs, and poor programmer morale. A structured maintenance approach (including such concepts as a documented maintenance master plan and the use of project management techniques) is presented. The author recommends that the maintenance problem can be overcome through proper management that incorporates formal planning and reporting disciplines as part of the maintenance process.

Key words: documented maintenance, maintenance, maintenance costs, management, planning, programmer morale, programmers, project management, reporting, software maintenance, software maintenance management, structured maintenance approach.

****

[RICH83] G.L. Richardson and C.W. Butler, "Organizational Issues of Effective Maintenance Management", AFIPS 1983 National Computer Conference Proceedings, pp. 155-161, May 1983.

It is a continuing challenge to today's data processing (DP) organization to evolve a management structure that matches the technological advances of the system for which it is responsible. The goal of this paper is to synthesize the emerging role of a DP organization within its corporate environment, focusing particular attention of the issue of software maintenance. Attention is primarily on three dimensions of the required organization: the user view, organized by functional area; the technical view, organized by area of expertise; and the organizational view, arranged by planning horizon. The conclusion is drawn that a single group should have responsibility for integration and enhancement of all installed applications. The tasks of this group comprise software configuration control, operational integrity, performance tuning, and requirements analysis and planning support for installed systems.

Key words: environment, maintenance management, organizational issues, planning, software maintenance, software maintenance management.

****

[RICH84] G.L. Richardson and E.D. Hodil, "Redocumentation: Addressing the Maintenance Legacy", AFIPS 1984 National Computer Conference Proceedings, vol. 53, pp. 203-208, 1984.

Over the past decade or so there has been much attention paid to techniques and methodologies to produce high-quality systems. A concurrent development has been the emergence of software tools that aid in the production and maintenance of software systems; yet the maintenance environment continues to be littered with poorly written and poorly documented programs. The focus of this paper is to outline a conceptual approach to the allocation of software maintenance resources and the role of automated tools in this process. The author suggests that it takes an administrative activity to quantitatively decide which code units are best for resource allocation. A case study is presented to illustrate the utility of this approach.

Key words: documentation, maintenance, management, redocumentation, software, software maintenance, software maintenance resources, software maintenance tools.

****

[ROMB84] H.D. Rombach, "Design Metrics For Maintenance", Ninth Annual Software Engineering Workshop, pp. 100-116, November 28, 1984.

This paper describes results of a study to develop maintenance metrics based on structural software design characteristics. The intent of the study was to define a characteristic metric set, suited to explain and predict software maintenance behavior. The maintenance aspects investigated in this study are stability and modifiability. While stability refers to the average number of modules affected per change cause, modifiability characterizes the ease with which changes can be made within each of these modules. Additional attention is dedicated to the difference between characteristic design and implementation metric sets, and to the difference between change behavior during development and maintenance. Six software systems and controlled maintenance experiments using these systems are examined.

Key words: maintenance metrics, metrics, modifiability, software design, software development, software maintenance, stability.

****

[ROSS75] D.T. Ross, J.B. Goodenough, and C.A. Irvine, "Software Engineering: Process, Principles, and Goals", Computer, pp. 17-27, May 1975.

This paper attempts to define the principles and goals that affect the practice of software engineering. Its intent is to organize these aspects of software engineering into a framework that rationalize and encourage their proper use, while placing in perspective the diversity of techniques, methods, and tools that presently comprise the subject of software engineering.

Key words: software engineering, software engineering methods, software engineering techniques, software engineering tools.

****

[RYAN82] J.R. Ryan, "Software Product Quality Assurance", AFIPS 1982 National Computer Conference Proceedings, vol. 51, pp. 393-398, 1982.

Providing clear objectives, guidelines, and requirements in an environment conducive to high productivity is absolutely essential to designing and producing high-quality software. The Software Quality Branch of the Computer Systems Division of Texas Instruments is tasked with providing support functions that are vital to producing high-quality software. This paper explains the role of the Software Quality Branch in administering the development methodology of the Computer Systems Division. It describes an effort to define and monitor quality indices and use of software quality circle to encourage commitment to quality goals and to develop solutions to quality problems.

Key words: productivity, quality, software quality, software quality assurance.

****

[SARS77] T. Sarson, "Structured Systems Development", Computer Decisions, vol. 9, no. 8, pp. 26-30, August 1977.

The key cost factor in computer systems is the time spent in testing, maintaining, and changing these systems. This has led to the development and use of techniques which maximize the changeability of computer software and make computer programs easier to read. These techniques are collectively referred to as structured systems development techniques. The four phases of structured systems development are: structured coding, structured design, structured walkthroughs, and top-down implementation.

Structured coding is the use of a small number of logical constructs to produce programs consisting of a nested hierarchy of one-entry, one-exit modules. Structured design is a set of techniques to insure that the modules of a system are well formed and independent of one another. Structured walkthroughs are the formal review of code and other development products by the members of a team responsible for detecting logic and other errors at an early stage. The top-down development strategy, the highest level skeleton of the system is coded and tested and then the detailed, lower levels of the system are integrated.

Key words: structured coding, structured design, structured systems development, structured walkthroughs, top-down implementation.

****

[SCHN79] N.F. Schneidewind and H.M. Hoffman, "An Experiment in Software Error Data Collection and Analysis", IEEE Transactions on Software Engineering, IEEE Computer Society Press, vol. SE-5, no. 3, pp. 276-286, May 1979.

The propensity to make programming errors and the rates of error detection and correction are dependent on program complexity. Knowledge of these relationships can be used to avoid error prone structures in software design and to devise a testing strategy which is based on anticipated difficulty of error detection and correction. An experiment in software error data collection was conducted in which the error data was carefully defined and analyzed. The error data was defined in terms of a directed graph representation of a program using such criteria as: errors found, time between error detections, and error correction time. Significant relationships were found between complexity measures and error characteristics.

Key words: abstract data types, complexity measures, error detection, program complexity, programming languages, set languages, software errors, testing, very high level languages.

****

[SCHN82a] N.F. Schneidewind, "Software Maintenance: Improvement Through Better Development Standards and Documentation", Naval Postgraduate School, Internal Report, February 1982.

Software maintenance is frequently the most expensive phase of the software lifecycle. It is also the phase which has received insufficient attention by management and software developers. Software standards have improved the ability of the software community to develop and design software. Unfortunately, most standards do not deal with the maintenance phase in a substantive way. Since maintainability has to be designed into the software and cannot be achieved after the software is delivered, it is necessary to have software standards which explicitly incorporate requirements for maintainability. Accordingly, this report suggests design criteria for achieving maintainability and evaluates Weapons Specification WS 8506 and MIL-STD 1679 against these criteria. Using these documents as typical examples of military software standards, recommendations are made for improving the maintainability aspects of software standards.

Key words: documentation, maintainability, software maintenance, software standards.

****

[SCHN82b] N.F. Schneidewind, "Evaluation of Secnavinst 3560.1 Tactical Digital Systems Documentation Standard for Software Maintenance", Naval Postgraduate School, Internal Report, February 22, 1982.

Management and developers have given insufficient attention to software maintenance, the most expensive phase of the software lifecycle. Standards have improved the ability to develop and design software, but most standards do not deal with the maintenance phase in a substantive way. SECNAVINST 3560.1 Tactical Digital Systems Documentation Standard for Software Maintenance, was evaluated with respect to its usability for software maintenance. Recommendations are made for improving the maintainability aspects of this instruction.

Key words: documentation, software maintainability, software maintenance, software standards, traceability.

****

[SCHN82c] N.F. Schneidewind, "Usability of Military Standards for the Maintenance of Embedded Computer Software", Naval Postgraduate School, Internal Report, June 1982.

Several military software standards were examined and evaluated with respect to their applicability and usability for maintaining embedded computer software. These standards were discussed from three standpoints: (1) the degree to which they support the use of newer software development technologies (e.g., requirements analysis methodologies) for improving software maintenance; (2) the effect of the microcomputer and its software development environment on the application of these standards; and (3) the extent to which these standards enhance traceability (tracing the various levels of related documentation).

Key words: aircraft software redesign project, microcomputer software, requirements analysis methodologies, software design, software maintainability, software maintenance, software standards, traceability.

****

[SCHN83] G.R.E. Schneider, "Structured Software Maintenance", AFIPS 1983 National Computer Conference Proceedings, pp. 137-144, May 1983.

Many books are written about structured design and programming, but never about structured maintenance. True structured maintenance comprises four functional roles: the manager, librarian, archivist, and programmer. The manager manages. The archivist protects contents of computer files and stores information about these files in an archive library. The librarian organizes and stores software documentation package. The programmer, of course, programs, using versions of the archive and library documents with slightly altered contents, and records day-to-day activities in the programmer's notebook. A special tool used by programmers is emergency takeover, which is a procedure for taking maintenance control of a new program.

Key words: archivist, librarian, managers, programmers, software documentation, software maintenance, structured software maintenance.

****

[SHAR77] W.K. Sharpley, "Software Maintenance Planning for Embedded Computer Systems", <u>COMPSAC Proceedings</u>, IEEE Computer Society Press, pp. 520-526, November 8-11, 1977.

This paper addresses key issues in software maintenance for embedded computer systems (i.e. weapon systems). Software maintenance cost factors such as problems with inefficient use of labor and inadequate planning are discussed. Planning should include provision for prompt and complete error detection and reporting by the system users. Software management tools and special tools for software maintenance are included. The author suggests that the goals of software maintenance of embedded computer systems are:

support for initial development, support of new requirements, and normal operational support.

Key words: embedded computer systems, maintenance cost factors, software management tools, software maintenance, software maintenance tools.

****

[SHEP77] S.B. Sheppard and L.T. Love, "A Preliminary Experiment to Test Influences on Human Understanding of Software", General Electric, Internal report, June 1, 1977.

Eight experienced programmers were each given three Fortran programs to memorize and reproduce functionally, without notes. Three levels of complexity of control flow and three levels of mnemonic variable names were independently manipulated. The experimental design was an incomplete split-plot factorial where each programmer was given one version of each program and all levels of the two primary independent variables. The participants correctly recalled significantly more statements when the complexity of control flow was reduced. The Pearson correlation coefficient was -0.81, over the 24 data points; thus indicating that Halstead's E is a powerful predictor of one's ability to understand a computer program.

Key words: complexity, control flow, Fortran, mnemonics, program style, programmers, software engineering, software psychology, software science, understanding of software.

****

[SHEP79] S.B. Sheppard, B. Curtis, and P. Milliman, "Factors Affecting Programmer Performance in a Debugging Task", General Electric, Internal Report, February 1979.

This report is the third in a series investigating characteristics of software related to psychological complexity. Three independent variables, length of program, complexity of control flow, and type of error, were evaluated for three different Fortran programs in a debugging task. Fifty-four experienced programmers were asked to locate a single bug in each of three programs. Documentation consisted of input files, correct output, and erroneous output. Performance was measured by the time to locate and successfully correct the bug. Small but significant differences in time to locate the bug were related to differences among programs and presentation order. Among measures of software complexity, Halstead's E proved to be the best predictor of performance followed by McCabe's v(G) and the number of lines of code. The number of programming languages known and familiarity with certain programming concepts also predicted performance. As in the previous experiments, experiential factors were better predictors for those participants with three or fewer years experience programming in Fortran.

Key words: control flow complexity, debugging, Fortran, modern programming practices, program errors, programmers, software psychology metrics, structured programming.

****

[SHEP81] S. Sheppard and E. Kruesi, "The Effects of the Symbology and Spatial Arrangement of Software Specifications in a Coding Task", General Electric Technical Information Series, Internal Report, July 1981.

Thirty-six participants were presented with specifications for each of three modular-sized computer programs. Nine different specification formats were prepared for each program. These formats varied along two dimensions: type of symbology and spatial arrangement. The type of symbology included natural language, constrained language (PDL), and ideograms (flowchart symbols). The spatial arrangement included sequential, branching, and hierarchical versions. Working from the specifications, the participants added and debugged about fifteen lines of code at the middle of each program. Substantial differences in performance were associated with the type of symbology. The natural language was considerably more difficult to code from than the constrained language or ideograms. The effect of spatial arrangement was not as great as the effect of symbology. Although not statistically significant, the branching arrangement appeared to be superior to the sequential and hierarchical arrangements. A comparison of the individual formats revealed that the constrained language presented in a sequential or in a branching arrangement resulted in the highest level of performance.

Key words: flowcharts, program design language, software documentation, software engineering, software experiments, software human factors.

****

[SHIF78] E.R. Shifflett, N.A. Hofland, and D.J. Schultz, "A Pragmatic Approach to Software Development", 17th Annual Technical Symposium, ACM/NBS, pp. 47-52, June 15, 1978.

This paper addresses and questions the fundamental theories inherent in the traditional way in which large, complex software systems have been built. Aspects addressed include the nature and characteristics of such systems, the methods used to get the technical work done, staffing philosophy, and management strategies. Based on an examination of the traditional concepts, a different philosophical approach is suggested and outlined. This includes viewing development of large systems as experimental in nature, reversing and changing the order of technical work activities to build such a system, using a new approach to staffing, and offering hope for management visibility and control of the process.

Key words: management, software development, software systems, staffing.

****

[SHNE80] B. Shneiderman, Software Psychology, Winthrop publishers, 1980.

This book reviews current trends and experimental results which have immediate application in software engineering and offers a model of human behavior which may be useful for further research. The author presents a definitive study on people who develop and maintain software. Some of the sections included in this book are aa follows: programming style, team organization, personality factors, and software quality evaluation.

Key words: personality factors, personnel, programming style, software development, software engineering, software maintenance, software psychology, software quality evaluation, team organization.

****

[SHNE86] B. Shneiderman, P. Shafer, R. Simon, L. Weldon, "Display Strategies for Program Browsing: Concepts and Experiment", IEEE Software, vol. 3, no. 3, pp. 7-14, May 1986.

The new, larger display screens can improve program comprehension - if the added space is used for more effective presentation, not just more code or larger type. Software maintenance is an important part of a programmer's work and a product's lifecycle, yet it remains one of the most troublesome of tasks. Even existing, newly developed techniques are not of much use, since only time can determine their value. Thus, instead of presenting another new maintenance tool or management technique, these pages focus on strategies for improving the presentation of information - specifically, on the new, larger display screens. An earlier version of the material in this article was presented at CSM-85. Recent experimental results have been added.

Key words: lifecycle, software maintenance, strategies, techniques, tools.

****

[SHOO83] M.L. Shooman, Software Engineering: Design, Reliability, and Management, McGraw-Hill, 1983.

This book presents software engineering methodologies for the development of quality, cost-effective, schedule-meeting software. The first chapter introduces the concepts of software development. This chapter focuses on software costs, the technical and managerial problems of software development and the techniques for designing high-quality software at a reasonable cost. Chapter 2 emphasizes modern software design methods such as modularity, structured programming, top-down design, and defensive programming. Chapter 3 develops complexity measures related to development cost and the number of program errors. Chapter 4 describes the various levels of testing (i.e. module and integration) and introduces several kinds of tests. Chapter 5 explains reliability concepts and develops models for predicting and measuring software errors, reliability, and availability. Chapter 6 deals with the basic principles of software management.

Key words: complexity measures, defensive programming, software costs, software design, software development, software errors, software management, software quality, software reliability, structured programming, testing, top-down design.

****

[SILV83] J. Silverman, N. Giddings, and J. Beane, "An Approach to Design for Maintenance", Honeywell Systems and Research Center, pp. 1-5, June 17, 1983.

Maintenance of a software system is enhanced when the system is viewed as a software architecture (an interconnection of parts). A Component Interconnection Language (CIL) has been defined for representing software architectures. In addition, several structural complexity metrics have been defined for evaluating architectures. The CIL supports maintenance in four basic ways: as a vehicle for recording design history, to help a maintainer evaluate the effect of a proposed design change, to help design retesting procedures, and as a basis for knowledge-based assistants for design and maintenance. The CIL and metrics have been evaluated in a design of a selected real-time, embedded software system.

Key words: component Interconnection Language (CIL), design, embedded software system, metrics, retesting, software architectures, software maintenance.

****

[SING80] L.M. Singer, "Attacking Maintenance Costs", Computerworld, p. 9 and pp. 12-16, September 8, 1980.

This article offers a method for reducing system maintenance costs. This method consist of three separate but overlapping stages: (1) measuring the existing maintenance effort, (2) summarizing and analyzing the maintenance activity, and (3) resolving the maintenance problems on a prioritized basis. In the first stage, the author suggests using a time project accounting system as a means for collecting data on system maintenance. In addition, personal logs that record specific activities and events should be filled out by the DP staff. The format of the log should be standardized, but the content should be left to the individual. The next stage, summarizing and analyzing the maintenance activity, can begin after two months of collecting data. This stage summarizes the quantitative and qualitative aspects of maintenance. In the final stage management should begin attacking the true causes of costly system maintenance on a priority basis.

Key words: changes, documentation, maintenance costs, management, operational support, programmers, software maintenance, system maintenance, user support, user training, users.

****

[SNEE84] H.M. Sneed, "Software Renewal: A Case Study", IEEE Software, pp. 56-63, July 1984.

This article presents the results of a test project to determine the usefulness of software engineering tools and techniques. This corporation had previously developed a commercial application system for distributing books and other publications throughout the world. Their system was developed using the HIPO method for design and a decision table generator, for coding. The data base, which contains 5000 data items, was constructed using the Adabas database system. Testing and documenting such a large system was a major problem.

The strategy of the project was to proceed in four stages: (1) the modules were to be statically analyzed and redocumented with the aid of a static analyzer; (2) the programs and data structures were to be formally specified using an automated specification tool; (3) the module test cases were to be written in a test specification language; and (4) The test specification was to be merged with the functional specification of the programs and their data structures. to create a production environment for maintenance and further development.

Although the testing project was not economically justified, it established a testbed for future testing. The project demonstrated that the postdocumentation and testing of programs can be done economically by using adequate tools.

Key words: database system, design, documentation, HIPO method, PL/1, redocument, respecify, reverify, software engineering, software engineering tools, specification tools, static analyzer, test cases, test specification, testing, tools.

****

[SOLO84] E. Soloway, S. Letovsky, B. Loerinc, and A. Zygielbaum, "The Cognitive Connection: Software Maintenance and Documentation", 9th Annual NASA/Goddard Workshop on Software Engineering, pp. 1-11, November 1984.

With the goal of trying to understand what software maintainers do, an experiment was conducted using audio, video-taped protocols with four expert maintainers as they were actively engaged in the process of enhancing a relatively small, interactive database program. The subjects exhibited a number of different types of information gathering strategies. Underlying these patterns of behavior, however, was the use of expectations about what should be seen in the program under examination. These expectations were generated on the basis of knowledge previously acquired as to the goals and programming plans that are typically employed in realizing interactive database programs. Thus, while the experts seemed to possess adequate programming knowledge, their actual code patches violated a basic principle of program structure. The failure by the programmers, at least in part, to was attributed to ineffective program documentation. Therefore improvements in the content of program documentation that should better facilitate software maintenance was recommended.

Key words: database system, documentation, maintenance programmers, modifications, program documentation, programming, software maintenance.

****

[SPIE76] M.J. Spier, "Software Malpractice - A Distasteful Experience", <u>Software - Practice and Experience</u>, vol. 6, no. 3, pp. 293-299, July-Sept 1976.

A sequence of events is described, which lead to the deterioration of an initially well-conceived and well-implemented compiler. This article presents a case study of how an initial "optimization" implanted a latent bug in the compiler, which was subsequently uncovered by a compiler modification. Details on how the compiler continued to deteriorated as a result of improper changes to correct the bug.

Key words: compilers, software engineering methodology, software maintenance, software production, structured programming.

****

[SPOK84] M. Spokony-Smith and R. Smith, "Getting the Best Software Support", Electronic Education, vol. 3, no. 6, p. 23 and p. 50, March 1984.

Software support and technical assistance are very important considerations when purchasing a software package. Three categories of software support and guidelines for achieving efficient software support service are provided.

1. No support - the buyer must figure out how to install and use the package.
2. Direct help from the software dealer - the buyer can call or go to the store to request aid for his problems.
3. Support from the software publisher - the buyer receives assistance via the telephone from a technical assistance department dealing directly with the end user.

Key words: end-users, software package, technical assistance.

****

[STAN77] J.R. Stanfield and A.M. Skrukrud, "Software Acquisition Management Guidebook: Software Maintenance", System Development Corporation, October 1977.

This report is one of a series of Software Acquisition Management (SAM) guidebooks. The scope of this document is limited to those acquisition and development activities, occurring throughout the SAM cycle, which impact software maintenance. It includes discussions of system turnover to the using command and the transfer of program management responsibility to the supporting command. The computer lifecycle is also considered. Most of the information provided in this report covers the implementing command's responsibilities during the SAM cycle. However, software maintenance during the Deployment Phase is also discussed to provide the background for proper planning. Current programming concepts are discussed as well as the military regulations, specifications, and standards. Within these constraints, this report emphasizes what the Program Office can do to specify and procure maintainable software, including procurement of the facilities, support tools, and documentation necessary to support software maintenance activities.

Key words: maintainability, software maintenance.

****

[SWAN79] E.B. Swanson, "On the User-Requisite Variety of Computer Application Software", IEEE Transactions on Reliability, pp. 221-226, August 1979.

In this model for software maintenance, user demand for enhancements and extensions is an integral component. Such demand is directed toward obtaining the user-requisite variety of the software, which manifests itself by recognizing errors of discrimination and omission during a history of data processing. The concept of variety in data and programs is introduced and developed, and measures are suggested. Behavior of the measures over a history of data processing is investigated. Validation and application of the model is discussed.

Key words: enhancements, errors, management, model, software maintenance, validation, user.

****

[SWAN84] E.B. Swanson, "Organizational Designs for Software Maintenance", Fifth International Conference on Information Systems, <u>IEEE Transactions on Software Engineering</u>, November 28-30, 1984.

Organization design theory is applied to issues in information systems (IS) management of application software maintenance. Following the suggestion of Galbraith (1973, 1977), it is argued that increased uncertainty in the maintenance task requires organizational adaptation to reduce the IS need for information processing in support of this task, or, alternatively, to increase the capacity for the same. Sources of uncertainty in the maintenance task are identified, and a number of related propositions are developed and enumerated. A corresponding set of design alternatives for maintenance management is also presented and given theoretical interpretation.

Key words: design, information systems management, software maintenance.

****

[TANN80] M.R. Tanniru, "Achieving Integrity in the System Maintenance Phase", Nineteenth Annual Technical Symposium, pp. 35-39, 1980.

Much attention has been paid to ensure systems integrity during the development phase. Growth of modular programing work-stations make it easier for a system or program designer to access and update the program modules in real-time as they attempt to modify existing systems or design new systems. This paper develops a maintenance procedure that makes available to users, programmers, system designers and auditors a 'reliable' documentation of each system.

Key words: documentation, management, module, programmers, system designers, software maintenance, system maintenance, users.

****

[TAUT83] B.J. Taute, "Quality Assurance and Maintenance Application Systems", AFIPS 1983 National Computer Conference Proceedings, pp. 123-129, May 1983.

Modifications to application systems in production can have a devastating effect on the environment if the changes are not handled correctly. A comprehensive quality assurance (QA) approach can help minimize this potentially harmful effect. This approach involves all groups: users, data processing center, applications programming, and quality assurance.

The QA approach should address four areas:

1. Phased approach
2. Procedure flows
3. Maintenance guidelines
4. Implementation

This paper describes a QA approach which consists of the definition and implementation of eight phases, envisioned in a circular lifecycle. Emergency processing is considered separately. Procedure flows consist of diagrams and charts listing responsibilities of the participants. Maintenance guidelines contain helpful hints and checklists and provide direction to the participants.

Key words: applications programming, data processing center, maintenance, quality assurance, software maintenance.

****

[TAUT84] B.J. Taute, "Software Maintenance: A Phased Approach", Data Management, pp. 37-39, March 1984.

The purpose of this article is to give the four groups involved with maintenance (the user base, DP center, application programming and quality assurance function) a structured approach for handling maintenance. This approach consist of eight phases : the request phase, estimate phase, schedule phase, programming phase, test phase, documentation phase, release phase (replacement of the old system with the changed system), and operation phase. Each phase is clarified, and deliverable's from each phase to the next phase are described. In addition, the procedure for handling emergency changes is explained. The author provides valuable information not only for DP management, but for each of the groups involved in software maintenance.

Key words: documentation, emergency processing, maintenance lifecycle, software maintenance.

****

[TEIC77] D. Teichroew and E.A. Hershey, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, vol. SE-3, no. 1, pp. 41-48, January 1977.

PSL/PSA is a computer-aided structured documentation and analysis technique that was developed for, and is being used for, analysis and documentation of requirements and preparation of functional specifications for information processing systems. The present status of requirements definition is outlined as the basis for describing the problem which PSL/PSA is intended to solve. The basic concepts of the Problem Statement Language are introduced and the content and use of a number of standard reports that can be produced by the Problem Statement Analyzer are briefly described.

Key words: computer-aided documentation, Problem Statement Analyzer, Problem Statement Language, PSL/PSA, requirements analysis, structured documentation.

****

[THAY80] R.H. Thayer, A. Pyster, and R. Wood, "The Challenge of Software Engineering Project Management", Computer, pp. 51-59, August 1980.

This article discusses the problems and issues in software engineering project management (SEPM). It presents an analysis of the survey results of the twenty hypothesized problems in SEPM. 361 persons including project managers, programmers/analysts, technical leaders, R&D personnel, and educators were surveyed. The survey revealed that: (1) planning activities are considered as the most important SEPM problem, (2) no consensus exists on how to solve these problems, and (3) the only groups who rated planning for maintenance as an important issue were educators and research and development personnel. The authors contend that one of the reasons that commercial software is so difficult to maintain is that industry is not convinced that planning for maintenance is an important problem. The results of the survey are both informative and revealing for persons involved with software engineering.

Key words: maintenance, software engineering, software engineering project management (SEPM), software maintenance.

****

[THAY81] R.H. Thayer and A.B. Pyster, "Major Issues in Software Engineering Project Management", IEEE Transactions on Software Engineering, vol. SE-7, no. 4, pp. 333-342, July 1981.

Software engineering project management (SEPM) has been the focus of much recent attention because of the enormous penalties incurred during software development and maintenance resulting from poor management. To date, there has been no comprehensive study to determine the significant problems of SEPM, their relative importance, or the research directions necessary to solve them. The author conducted a survey of individuals from all areas of the computer field to determine the general consensus on SEPM problems. Twenty hypothesized problems were submitted to several hundred individuals for their opinions. The 294 respondents validated most of these propositions. All of the propositions were considered to be important. The respondents indicated a number of research directions.

Key words: management, software development, software engineering project management (SEPM), software maintenance, survey, university curriculum.

****

[TINN83] P.C. Tinnirello, "Improving Software Maintenance Attitudes", _AFIPS 1983 National Computer Conference Proceedings_, pp. 107-112, May 1983.

In the past, there has been little recognition of the significance of how attitudes affect the performance of maintenance functions. Investigation into the origin of these attitudes has led the author to formulate feasible solutions that foster productive attitudes through the educational and professional work environments.

Key words: environments, maintenance attitudes, maintenance management, maintenance programming, productivity, software maintenance.

****

[TINN84] P.C. Tinnirello, "Software Maintenance in Fourth-Generation Language Environments", _AFIPS 1984 National Computer Conference Proceedings_, vol. 53, pp. 251-257, May 1984.

It is often asserted that fourth-generation languages will resolve the problems associated with software development, and in particular the technical and morale problems of software maintenance. This paper suggests that fourth-generation languages can not solve all of the present problems of maintenance, and indeed they can introduce problems of their own. The successful user of fourth-generation languages will be the organization that takes appropriate countermeasures. Benefits and disadvantages of fourth-generation languages are explored.

Key words: documentation, fourth-generation languages, maintenance programming, product releases, quality assurance, software ownership, software selection, software standards, software warranty.

****

[TIPS80] E. Tipshus and J. Sanderson, "Rotating `Rival' Programmers Between Applications and Maintenance Dampens Antagonism, Improves Documentation", Data Management, pp. 42-46, June 1980.

This article contains an example of an organization's problems due to conflicts between the system development and maintenance groups. Problems with project turnover and staff rivalry are described. The solution implemented by this organization was to combine these two groups into one system development department. This new department had the following attributes: (1) a new group of maintenance programmers consisting of a group leader and seven rotating programmers who were responsible for emergency maintenance; (2) approximately 10% of the DP staff would be responsible for regular enhancements and modifications; (3) at implementation, a programmer who had worked on developing the system would be assigned to the maintenance group. A description of initial problems and their solutions during this reorganization is provided.

Key words: development group, documentation, maintenance group, maintenance programming, management, personnel, programmers, project

****

[WALK81] M.G. Walker, Managing Software Reliability - The Paradigmatic Approach, North Holland, 1981.

This book proposes a paradigm which transforms the management of software reliability from a craft into a controllable, scientifically based engineering discipline. Software developers can use the paradigm to organize software development, maintenance, and conversion into a predictable and reliable methodology.

The paradigm specifically guides software managers in the following areas:

1. It gives them an understandable picture of the software lifecycle.
2. It offers them a guide for selecting techniques for building and maintaining a system.
3. It provides them with a guide for managing their staffs.

Key words: management, software conversion, software development, software lifecycle, software maintenance, software reliability.

****

[WALT78] G.F. Walters and J.A. McCall, "The Development of Metrics for Software R&M", Proceedings of the Annual Reliability and Maintainability Symposium, DPMA, pp. 79-85, January 1978.

This paper describes the derivation and validation of software metrics which provide a means for quantitatively specifying and measuring software quality. The procedure for quantifying software quality is : (1) determine a set of factors which comprise software quality; (2) develop a set of criteria for each factor; (3) define a metric (measure) for each criterion and use a `normalization function' to combine all these measures for an overall rating of the factor; (4) validate the metrics and normalization functions by utilizing historical data; and (5) set guidelines that can be used by project management to specify the quality of the software product and to measure in the development stages whether the project is meeting that level of quality. Eleven software quality factors are listed. This article provides an explanation of the uses and benefits of software metrics for software maintenance.

Key words: correctness, flexibility, reusability, portability, software maintainability, software metrics, software quality, software reliability, testability.

****

[WASS78] A.I. Wasserman and L.A. Belady, "Software Engineering: The Turning Point", <u>Computer</u>, pp. 30-41, September 1978.

This article deals with the problems of software engineering in the late seventies. Steps to improve the software engineering field are described. The first step is to develop validated and well-tested software components (packages) instead of programming from scratch. These components should be easier to modify and maintain. The author suggests a software component inventory from which a software implementor could design a system. The second step is to learn from present systems and tools in order to improve their operation and to evolve them gradually into new configurations. Finally, there needs to be improvement in transferring technological information on software engineering to industry, in order to ensure widespread adoption of software engineering principles. This involves improving educational programs both within universities and industries. Future research directions in software engineering are described. They include: (1) Flexibility and modularity (2) module interfaces (3) validation, verification, and testing (4) specification techniques. The authors also identify some of the external pressures (i.e., economic and legal) which force people to change their present practices and set higher standards for their software system.

Key words: reuse, software packages, software engineering, testing, verification.

****

[WASS82] A.I Wasserman and S. Gutz, "The Future of Programming", Communications of the ACM, vol. 25, no. 3, pp. 196-206, March 1982.

The nature of programming is changing. These changes will accelerate as improved software development practices and more sophisticated development tools and environments are produced. This paper surveys the most likely changes in the programming task and in the nature of software over the short, medium, and long term.

In the short term, the focus is on gains in programmer productivity through improved tools and integrated development environments. In the medium term, programmers will be able to take advantage of libraries of software components and to make use of packages that generate programs automatically for certain kinds of common systems. Over the longer term, the nature of programming will change even more significantly as programmers become able to describe desired functions in a nonprocedural way, perhaps through a set of rules or formal specification languages. As these changes occur, the job of the application programmer will become increasingly analysis-oriented and software developers will be able to attack a large number of application areas which could not previously be addressed effectively.

Key words: integrated development environments, packages, prototyping, personal development systems, reuse, software components.

****

[WEIN71] G.M. Weinberg, The Psychology of Computer Programming, Van Nostrand Reinhold, 1971.

This book investigates the behavior and thought processes of programmers as they carry out their daily activities. The author discusses the following psychological factors:

1. The programmer's working quarters have a profound effect on his/her productivity.
2. Social factors that cause different levels of performance from different programmers.
3. 'Ego-less programming' and individual ownership of programs.
4. The creation and coordination of programming teams."

Key words: human factors, programmers, psychological activities, software maintainability.

****

[WEIS81] M. Weiser, "Program Slicing", Fifth International Conference on Software Engineering, IEEE Computer Society Press, pp. 439-449, 1981.

Program slicing is a method used by experienced programmers for abstracting from programs. Starting from a subset of a program's behavior, slicing reduces the program to a minimal form which still produces that behavior. The reduced program, called a "slice", is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior. A dataflow algorithm is presented for approximating slices when the behavior subset is specified as the values of a set of variables at a statement. Experimental evidence is presented that these slices are used by programmers during debugging. Experience with two automatic slicing tools is summarized. New measures of program complexity are suggested based on the organization of a program's slices.

Key words: debugging, program maintenance, program slicing, dataflow analysis, slice.

****

[WEIS82] M. Weiser, "Programmers Use Slices When Debugging", Communications of the ACM, vol. 25, no. 7, pp. 446-452, July 1982.

Computer programmers break apart large programs into smaller coherent pieces. Each of these pieces: functions, subroutines, modules, or abstract datatypes, is usually a contiguous piece of program text. The experiment reported here shows that programmers also routinely break programs into one kind of coherent piece which is not contiguous. When debugging unfamiliar programs, programmers use program pieces called slices which are sets of statements related by their flow of data. The statements in a slice are not necessarily textually contiguous, but may be scattered through a program.

Key words: Program decomposition, slice.

****

[WHIT77] D.C. Whitmore, R.D. Bivans, D.L. Bowie, and M.P. Kress, "Computer Program Maintenance: Software Acquisition Engineering Guidebook Series", Internal report, Boeing Aerospace Company, December 1977.

This report is one of a series of guidebooks whose purpose is to assist Air Force Program Office Personnel and other USAF acquisition engineers in the acquisition engineering of software for Automatic

Test Equipment and Training Simulators. This guidebook describes the software maintenance lifecycle, including maintainability, maintenance tasks and required maintenance resources.

Key words: software acquisition, acquisition engineering, software maintenance, software change control, software certification, contractor support, software lifecycle costs, support software.

****

[WHIT81] B.A. Whitesides, "The Hidden Costs of In-House Development", <u>Datamation</u>, pp. 173-175, September 1981.

The purpose of this article is to compare the in-house development costs against the external development costs. Many hidden costs for internal development are described (i.e., recruiting and training programmers, monthly benefits for each employee and rushing the implementation of a project). According to the author, there is less risk in obtaining services from an outside contractor than in hiring programmers. One reason is that payments can be withheld or recovered from a contractor while this is not possible with an unsuitable employee. Generally, a software house provides a variety of skills not always available in-house and thus, produces significant advantages in management control. This article identifies some of the benefits of using the services of a software house but does not mention any of the disadvantages.

Key words: in-house development costs, external development costs, software, training.

****

[WIEN84] W.K. Wiener-Ehrlich, J.R. Hamrick and V.F. Rupolo, "Modeling Software Behavior in Terms of a Formal Life Cycle Curve: Implications for Software Maintenance", <u>IEEE Transactions on Software Engineering</u>, vol. SE-10, no. 4, pp. 376-383, July 1984.

In this paper, a formal model of the software manloading pattern, the Rayleigh model, is described and then applied to four Bankers Trust Company (BTCo.) new development projects possessing complete lifecycle manloading data (maintenance phase included). To fit the Rayleigh curve to a project's manloading scores, (nonlinear) regression was used to obtain least squares estimates of the Rayleigh parameters, which, in turn, were used to generate the Rayleigh manloading curve. For all four projects, deviation from the Rayleigh curve was small and constant throughout the software development phases (i.e., preliminary design through implementation); however, the Rayleigh curve consistently deviated from the actual manloading during system maintenance, underestimating the amount of maneffort expended. Restricting maintenance effort to manpower expended repair of system faults ("corrective" maintenance) resulted in a single Rayleigh curve that could be applied over the entire BTCo. lifecycle. Furthermore, this corrective portion of the maintenance effort could be accurately forecasted from the Rayleigh curve fit to software development. Implications of these findings for software management are discussed.

Key words: corrective maintenance, development project, empirical, fitted curve, forecasting software maintenance, formal model of software lifecycle, projected curve, Rayleigh model, residual score.

****

[WILS81] L. Wilson, "The Do's and Dont's of Documentatlion", <u>Datamation</u>, pp. 185-186, September 1981.

This article discusses documentation utility. It provides helpful hints for organizations having difficulties using existing documentation or faced with the responsibility of writing their own documentation. Many types of documentation are addressed.

Key words: documentation, management, operations documentation, program documentation, system documentation, software maintenance.

****

[WINO79] T. Winograd, "Beyond Programming Languages", <u>Communications of the ACM</u>, vol. 22, no. 7, pp. 391-401, July 1979.

As computer technology matures, our growing ability to create large systems is leading to basic changes in the nature of programming. Current programming language concepts will not be adequate for building and maintaining systems of the complexity called for by the tasks we attempt. Just as high level languages enabled the programmer to escape from the intricacies of a machine's order code, higher level programming systems can provide the means to understand and manipulate complex systems and components. In order to develop such systems, we need to shift our attention away from the detailed specification of algorithms, towards the description of the properties of the packages and objects with which we build. This paper analyzes shortcomings of programming languages, and identifies some possible directions for future research.

Key words: programming, programming languages, programming systems, systems development.

****

[YAU78] S.S. Yau, J.S. Collofello, and T. MacGregor, "Ripple Effect Analysis of Software Maintenance", <u>COMPSAC Proceedings</u>, IEEE Computer Society, pp. 60-65, 1978.

Maintenance of large-scale software systems is a complex and expensive process. Large-scale software systems often possess both a set of functional and performance requirements. Thus, it is important for maintenance personnel to consider the ramifications of a proposed program modification from both a functional and a performance perspective. This paper describes the ripple effect which results as a consequence of program modification will be analyzed. A technique is developed to analyze this ripple effect from both functional and performance perspectives. A figure-of-merit is then proposed to estimate the complexity of program modification. This figure can be used as a basis upon which various modifications can be evaluated.

Key words: functional requirements, large software systems, maintenance personnel, performance requirements, program modification, program modification complexity, ripple effect analysis, software maintenance.

****

[YAU79] S.S. Yau and J.S. Collofello, "Performance Considerations in the Maintenance Phase of Large-Scale Software Systems", Rome Air Development Center, Griffiss, AFB, N.Y., Contract report, June 1979.

Maintenance of large-scale software systems is a complex and expensive process. Large-scale software systems often possess both a set of functional and performance requirements. Thus, it is important for maintenance personnel to consider the ramifications of a proposed modification from both a functional and a performance perspective.

This report describes the possible effect of program modifications during the maintenance phase on the performance of large-scale software systems is analyzed. Mechanisms for the propagation of performance changes from one part of the system to another are identified, and the relationship among these mechanisms, performance attributes, critical program sections and performance requirements is also investigated. The development of a maintenance technique for predicting which performance requirements in the system may be affected by a proposed modification is outlined. This technique will enable maintenance personnel to incorporate performance considerations in their criteria for selecting the type and location of software modifications needed. It also helps to identify which performance requirements should be verified to ensure that they were not violated by the modification.

Key words: critical sections, large software systems, mechanisms of propagation, performance attributes, performance changes, performance consideration, ripple effect analysis, software maintenance.

****

[YAU80a] S.S. Yau, "Self-Metric Software Summary of Technical Progress", Rome Air Development Center, Griffiss AFB, N.Y., F30602-76-C-0397, April 1980.

This report documents the research in the area of developing effective techniques for large-scale software maintenance, including those for the design, implementation, validation, and evaluation of reliable and maintainable software systems with a high degree of The research results which have been presented in previous papers and interim technical reports are summarized, and unfinished work is presented.

Key words: logical and performance ripple effect analysis, maintainability, software maintenance, self-metric software, stability, understandability.

****

[YAU80b] S.S. Yau and J.S. Collofello, "Self-Metric Software A Handbook: Part I and II, Performance Ripple Effect Analysis", Rome Air Development Center, Griffiss AFB, N.Y., April 1980.

This handbook consists of two parts on ripple effect analysis for large-scale software maintenance. Part I describes a ripple effect analysis technique for software maintenance from the logical or functional perspective is presented. Part II describes a ripple effect analysis technique for software maintenance from the performance perspective. The material is organized in several levels: a description of the technique; performance ripple effect analysis technique, as well as how this technique is interfaced with the user; a lexical analysis and tracing phase, and finally, integration of the processing steps involved in this analysis.

Key words: analysis, lexical analysis and tracing, mechanisms for modification propagation, performance ripple effect software maintenance.

****

[YAU82] S.S. Yau and J.S. Collofello, "Design Stability Measures for Software Maintenance", <u>COMPSAC Proceedings</u>, pp. 100-108, November 8-12, 1982.

The high cost of software during its lifecycle can be attributed largely to software maintenance activities, and a major portion of these activities is to deal with the modifications of the software. In this paper, design stability measures which indicate the potential ripple effect characteristics due to modifications of the program at the design level are presented. These measures can be generated at any point in the design phase of the software lifecycle which enables early maintainability feedback to the software developers. The validation of these measures and future research efforts involving the development of a user-oriented maintainability measure which incorporates the design stability measures as well as other design measures are discussed.

Key words: design stability, programming, ripple effect analysis, software maintenance.

****

[ZAK83] J.R. Zak, "When a Data Processing Department Inherits Software", <u>AFIPS 1983 National Computer Conference Proceedings</u>, pp. 163-172, 1983.

This paper discusses some of the problems that occur in dealing with inherited software and some of the basic procedures necessary to manage successfully a data processing department or group that is converting and/or maintaining software that has been imposed on them and that they neither designed nor implemented.

Key words: inherited software, management, software conversion, software maintenance.

****

[ZELK78] M.V. Zelkowitz, "Perspectives on Software Engineering", <u>Computing Surveys</u>, pp. 197-216, June 1978.

Software engineering refers to the process of creating software systems. It applies loosely to techniques which reduce high software cost and complexity while increasing reliability and modifiability. This paper outlines the procedures used in the development of computer software, emphasizing large-scale software development, and pinpointing areas where problems exist and solutions have been proposed. Solutions from both the management and the programmer points of view are given for many of these problem areas.

Key words: certification, chief programmer team, program correctness, program design language (PDL), development lifecycle, software engineering, structured programming, top-down design, top-down development, validation, verification.

****

[ZELL83] L. Zells, "Data Processing Project Management: A Practical Approach for Publishing a Project Expectations Document", <u>AFIPS 1983 National Computer Conference Proceedings</u>, pp. 181-187, 1983.

The mounting demand for proficient personnel and the parallel increase in salaries, has prompted management to look for ways to improve productivity in order to realize a higher return on their investment dollars. Knowing what to do, when to do it, and how to do it prevents costly retries. Given any kind of a project and 2 to N participants, there will be 2 to N views of the project. This paper reflects a practical method for transforming facts and conflicts into an approved development approach and publishing the results in what will be called a project expectations document.

Key words: management, productivity, personnel, project expectations document, project management.

****

[ZIRK78] A.L. Zirkle, "An Automated Software Maintenance Tool For Large-Scale Operating Systems", Navy Industrial Fund, December 1978.

A method for automating many of the tasks involved in maintaining a large computer operating system (SCOPE 3.4 on the CDC 6700) is described. The method is embodied in several procedures, each of which aids in one phase of operating system maintenance. These procedures are written in a manner that promotes ease of modification or enhancement. This automated maintenance tool can also be used in other programming applications where a large amount of effort must be expended in noncreative housekeeping tasks.

Key words: operating systems, programming, program libraries, programmer efficiency, software development methodology, software maintenance, software tools, systems programming.

****

[ZUCK83] S. Zucker, "Automating the Configuration Management Process", Softfair Proceedings, IEEE Computer Society Press, pp. 164-172, July 25-28, 1983.

An Automated Configuration Management System (CMS) was developed at General Electric in order to improve and enhance the mutual techniques for project tracking and change control and to allow for reliable management of large multi-facility projects. Because CMS works with any file, it can perform Configuration Management on all items in the configuration. The bookkeeping and status accounting forms are displayed on a terminal and the user is assisted in completing them correctly. New configuration items or changes are entered into the system only after approval by the proper authority. A common project data base is built by CMS which makes visibility of current system status available to anyone who has access authority. Standard report forms, as well as user defined report forms are used when viewing the current or historic system information. CMS not only controls the configuration, but also the paperwork, change approval cycle, and the quality of the project.

Key words: maintenance programmers, morale, management, personnel, software maintenance.

****

[ZVEG82] N. Zvegintzov, "The Eureka Countdown", Datamation, pp. 172-178, April 1982.

A five-step procedure(path) for understanding a software system referred to as the Eureka countdown include:

Step 5: the five questions to ask;
Step 4: the four actions to take with the answers;
Step 3: the three places to work. These are the places to penetrate the system;
Step 2: the two products - what you do with your knowledge;
Step 1: the one golden rule.

The five questions are: What?, Why?, How?, Where From?, and Where To? The most important step is to learn about the "what's" in your system. The "what's" in your system would include any functions important enough to have names. The four actions are: review, reflect, record, and react. The reflect action involves simplifying, correlating, and evaluating. These four actions repeat and form a cycle. The three places to work are at the top, the problem, and the edges. Working at the problem is equivalent to the technique of 'working backwards'. Working at the edges means working in areas where you have at least a partial understanding of that area. The two products of understanding are action and readiness. Finally, the golden rule is never give up! The Eureka Countdown can be integrated with other methodologies and tools. The author suggests that the Eureka Countdown is not so much a method as an attitude; it provides DP workers with information on how to approach a problem.

Key words: Eureka Countdown, management, personnel, programmers, software maintenance.

****

[ZVEG83] N. Zvegintzov, "Quick Response as a Maintenance Management Strategy", COMPSAC Proceedings, IEEE Computer Society Press, pp. 91-92, November 7-9, 1983.

In order to solve the problems of software maintenance management, a managerial/engineering strategy for a quick response should be implemented. The author lists software maintenance problems as expressed by the following groups: users, maintenance programmers, and first and second level managers. The strategy would be a package that includes such tactics as knowledge of the product, an easy-to-modify product, planned availability, team training, and

scenario exercises.

Key words: maintenance programmers, managers, quick response strategy, software maintenance, software maintenance management, users.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBS/SP-500/141	2. Performing Organ. Report No.	3. Publication Date Sept. 1986

4. TITLE AND SUBTITLE

Computer Science and Technology:
Annotated Bibliography on Software Maintenance

5. AUTHOR(S)

Wilma M. Osborne Ronnie T. Raigrodski

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE

Gaithersburg, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

Same as item 6.

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number 86-600579

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

This annotated bibliography contains summaries of two hundred and fifty software maintenance articles or papers from computer science journals, books, proceedings, Federal publications, computer newspapers, and other technical reports published during the ten various aspects of software maintenance including the problems and issues faced in most software maintenance environments. It also identifies techniques, procedures, methodologies, and tools employed throughout the lifecycle of a software system to improve the maintainability and quality of that system.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

configuration management; cost; documentation; errors; lifecycle; metrics; productivity; programmers; software; software packages; software quality; techniques; testing; tools; user.

13. AVAILABILITY

[✓ Unlimited]

☐ For Official Distribution. Do Not Release to NTIS

[✓ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.]

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

138

15. Price

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name ______________________________

Company ______________________________

Address ______________________________

City ______________ State ____________ Zip Code __________

(Notification key N-503)