The management of software engineering Part V: Software engineering management practices

by R. E. Quinnan

The IBM Federal Systems Division software engineering program was organized to support the design and development of software products. This program includes design practices that deal with the systematic decomposition of software designs into hierarchically related programs. This procedure results in products with structural integrity that are easy to use, maintain, and adapt. Development practices in the software engineering program deal with software implementation and integration engineering. The discipline of management practices closes and strengthens the triangle model of software engineering.

This part of our paper focuses on these management practices and discusses the plans and controls they provide to monitor progress and performance during the software life cycle. The software engineering management practices reflect the experiences of successful management teams and are familiar to most software managers. However, the effectiveness of these practices is significantly improved when the associated design and development practices are implemented. Uniformity and consistency resulting from good design and orderly development underlie the predictability and responsiveness of the management practices.

Software engineering management model

Our software engineering management model is composed of three sets of practices: (1) technical review; (2) cost management; and (3) software program management. The technical reviews are conducted during the development of a software product at specified checkpoints and for well-defined purposes. Cost management prescribes a method of planning, estimating, measuring, and controlling a developing software product to meet a cost objective. The software program management practice establishes a project environment and management relationships that foster complete, precise, and efficient communications within and between groups. The model is based on the software life-cycle activities described by O'Neill in Part II of this paper.

Table 1 Software life cycle

Activity	*Work components*	*Outputs*
System definition	Software requirements definition	Software system requirements specifications
	Software system description	Software system description document
	Software development planning	Software development plan
	Engineering change analysis	Engineering change proposals
Software design	Functional design	Functional design specifications
	Program design	Program design specifications
	Test design	Test design specification
	Software tools	Utilities, debugging aids
	Design evaluation	
Software development	Module development	Development (module) libraries
	Development testing	Development test procedures/ reports
	Problem analysis and correction	Program modifications
Software system test	Software system test procedures	Software system test procedures
	Software integration and test	Software system integration library, test reports
System/ acceptance test	System test support	System library, test reports
	Acceptance test support	Delivered software system/ acceptance library
Operational support	System operation support	Level of effort assistance, maintenance
	Training	Level of effort training manuals, courses
	Site deployment support	Level of effort assistance
General support	Project management	Level of effort
	Configuration management/ control	Procedures, standards, library control
	Software cost engineering	Cost management practice support
	Quality assurance	Audits, quality assurance plans
	Administration centers/technical publications	Level of effort

Each of these activities is in turn composed of the set of work components shown in Table 1, which identify the work performed, the expected end products, and criteria for completion. In a typical project, these activities and components overlap; baseline releases are defined to indicate when a component output has satisfied its completion criteria.

FSD projects can cover complete software product life cycles from concept formulation through end-of-life. Quite often, however, our responsibility spans system definition through acceptance test with a limited responsibility for operational support. The customer does his own requirements definition and runs his own operations in these cases. There are also some projects, usually large ones, in which several software subcontractors share

Table 2 Technical reviews within the software life cycle

Software life-cycle activities	*Related work components*	*Technical reviews*
System definition	Software requirements definition Software system description	System requirements Software system specification Test plan
	Software development planning	Documentation outline
Software design	Functional design	Software system design Integration plan
	Program design	Module design
	Test design	Test plan Test specification
	Software tools	
	Design evaluation	
Software development	Module development	Module implementation Unit test procedure
	Development testing	Module qualification Test procedure
Software system test	Software system test procedures System integration and test	Test procedures Software system qualification
System and acceptance test	System test support Acceptance test support	Software system acceptance Documentation completion
Operational support	System operation support Training Site deployment support	

the workload. Our role in such situations can span any or all of the life-cycle activities. Thus the life-cycle model provides a standard for determining the scope of software engineering in a particular project and serves as an improved communications method. The latter is especially important on larger projects where there is significant interaction between software engineering and other functional groups, such as program management and systems engineering.

technical reviews

It is well known that early detection of problems and errors is the most cost-effective method of quality control.[30] Since the person who generates a problem or an error can easily overlook it, we rely on technical reviews for thoroughness. This procedure brings the talent of a wider group of people to bear on each work product, quickly and efficiently.

Technical reviews develop a strategy within the software life cycle that permits the assessment and control of software activ-

ity. The strategy is associated with the life cycle just mentioned so that the technical results of each activity can be evaluated. Table 2 illustrates the correspondence of the technical reviews with the software life-cycle activities and the resulting work components. The number of actual reviews may vary, depending on individual project characteristics. Some reviews reoccur each time an event occurs, such as completion of a document outline. Multiple reviews can be conducted at one time; documentation, test, and integration reviews are normally conducted in conjunction with other reviews. In some cases, it may be convenient to run a review of nontechnical issues, such as contract compliance, budget tracking, and resource plans, immediately before or after a technical review. By such scheduling, technical problems may be completely resolved then and there by lining up all the resources and administrative approvals that might be needed.

Each review in Table 2 has a stated purpose outlined as follows:

- *System requirements.* Determine that software requirements for system capability are completely and correctly stated to permit development and use by the planned system user.
- *Software system specification.* Determine that software system specifications are complete and correct; ensure that each requirement for system capability can be traced through to the delivered software product.
- *Test plan.* Determine that the implementation of the software system is tested against the software system specification and that all requirements are checked out.
- *Documentation outline.* Determine that the outline for any planned document satisfies its objectives.
- *Software system design.* Determine that module designs comply with the software system specification and collectively implement the software system specification.
- *Integration plan.* Verify that there is a systematic approach to the implementation and testing of the software system.
- *Module design.* Determine that program and data designs comply with their module designs and implement their intended function.
- *Test specification.* Verify that test methods and materials comply with an approved test plan; evaluate functional and performance details of the tests versus the test objectives.
- *Module.* Verify that programs and modules are correctly implemented in accordance with their design and that unit test procedures have been established.
- *Test procedure.* Verify that test methods and materials comply with an approved test specification; evaluate the test operational scenario and machine execution control details.
- *Module qualification.* Determine that the module complies with the software specification; certify the module so that the code can be promoted from the project development library to the integration library.

- *Software system qualification.* Determine that the implemented software system complies with the software system specification; certify the system so that the code can be promoted from the integration library to the release library.
- *Software system acceptance.* Verify that the software system complies with all project deliverable objectives; certify the system so that the code for the software can be released to the customer (or the integration activity in a hardware/software system project).
- *Documentation completion.* Verify that the completed documentation satisfies its objectives and complies with its approved outline.

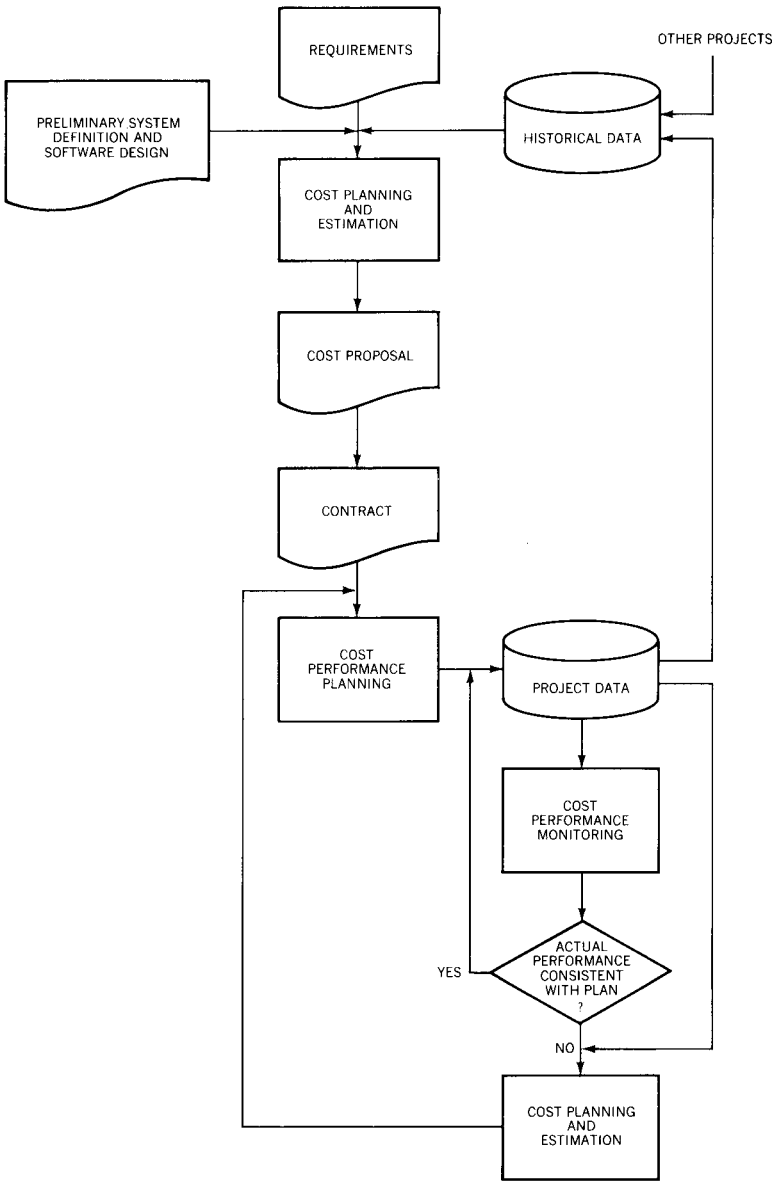
All these reviews are tools for project managers to use in assessing how well objectives are being met. To a large degree, the reviews deal with documents—specifications, test plans, test reports, procedure descriptions, and, ultimately, code. Reviewers can read the documents to assess the content and quality; they can talk to the developers to assess the intent of the implementation and to clarify unclear statements. Intuition and judgment are required, besides technical knowledge. Reviewers must spot weaknesses in the work products, propose fixes, and establish criteria for subsequent reviews to verify that the fixes are successful. The final acceptance reviews certify that we are confident that the software product is ready for the customer and can pass his acceptance test.

cost management

Software systems are built to provide specific capabilities for the user. Inevitably, the capabilities delivered depend on how much the user can afford to spend. In government contracts, as in business information systems, value/cost tradeoffs reflected in project budgets place constraints on software development plans. Our goal is to give our managers planning and control procedures that permit them to manage technical progress and cost at the same time.

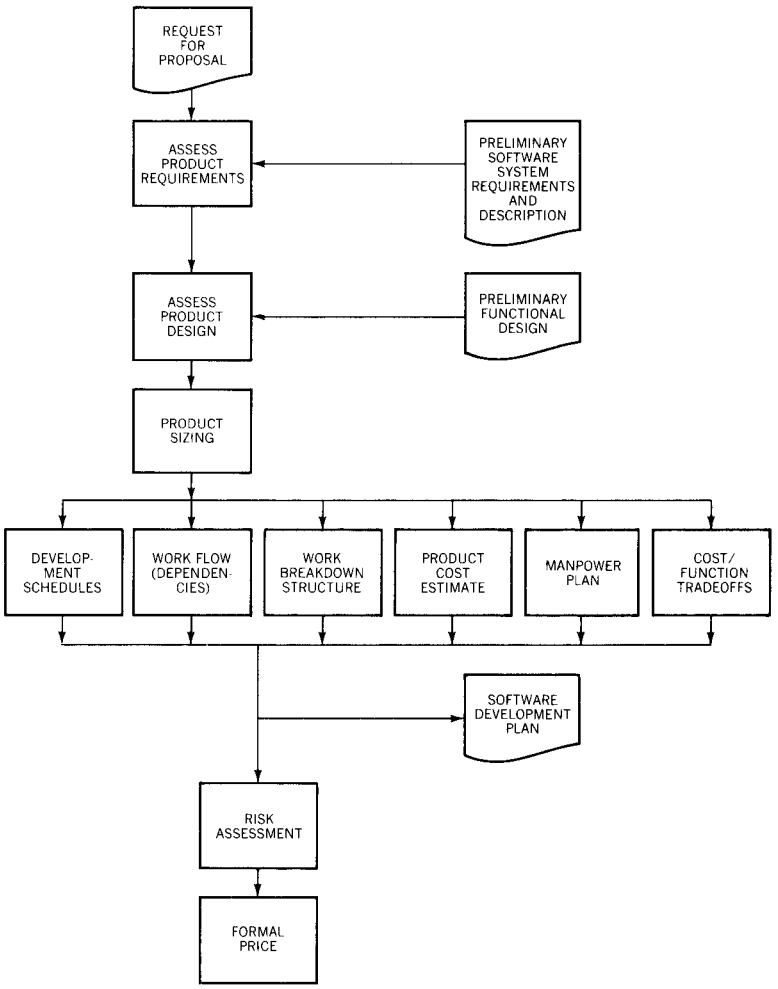
The cost management process, illustrated in Figure 1, starts when the software design is sufficiently detailed to support cost estimates and identify areas of risk. The planning and estimation steps are depicted in Figure 2. By spelling out all these steps in the cost management practice, we tend to avoid oversights. Looking back to 1964, when many large projects were severely underestimated, we now see that simple oversights can be identified as major sources of error.[31] Estimating methodology at that time focused only on actual programming activity; technical support, administration, and management were routinely omitted. Since the omissions were large—typically 150 percent of the direct programming—cost target misses were large. The solution for many years was to adjust the basic programming estimate by a factor that compensated for omissions. Meanwhile, an effort was

Figure 1 Cost management process



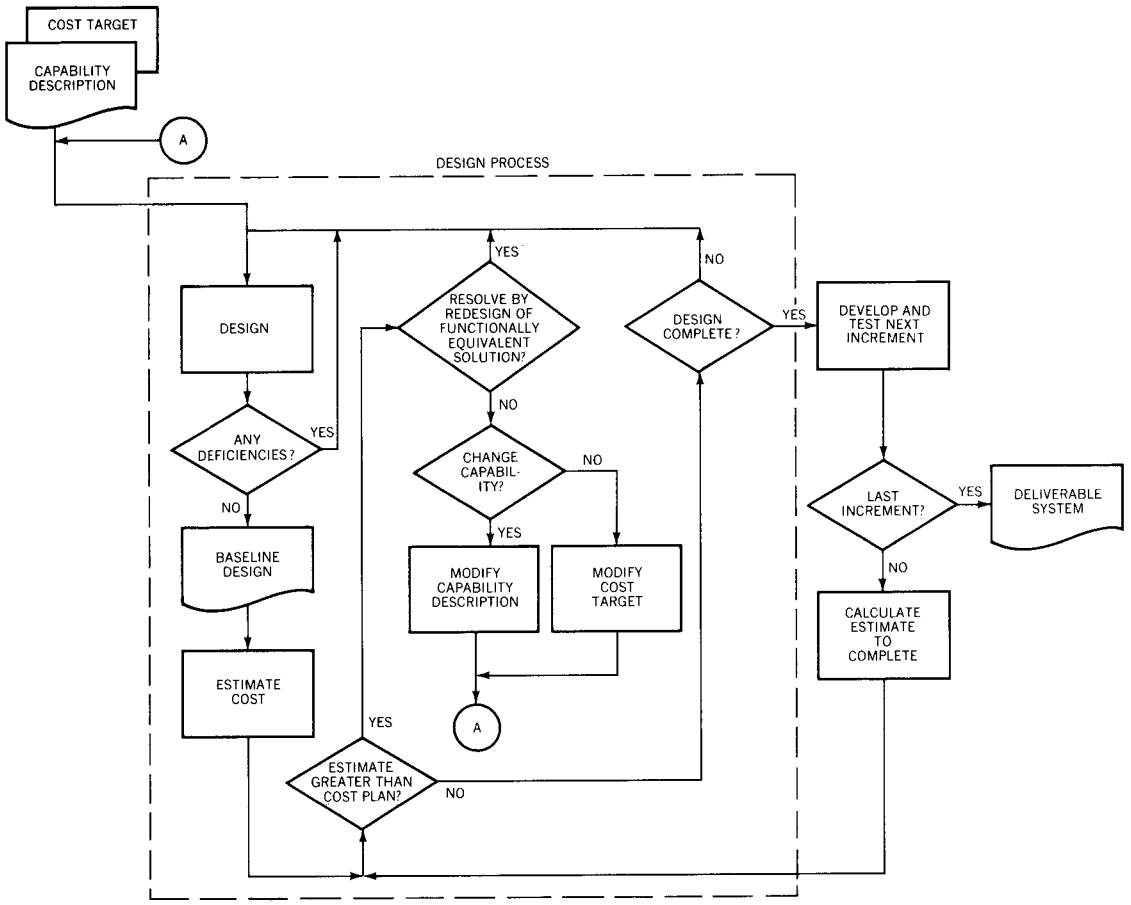
made to collect enough historical project data to replace the rule-of-thumb adjustment with more reliable methods. Now, we can tell our managers how to estimate each stage of the life cycle and how to deal explicitly with overhead support activities.[32] Our cost management practice, moreover, reminds managers to carry out each step without omissions. As a result, our proposals to customers contain a cost plan that can be tracked throughout the life cycle.

Figure 2 Planning and estimating



The cost plan contained in the proposal is refined further after contract award. At this time, detailed budgets are prepared and checkpoints are established. Procedures are also set up to collect performance measurements to permit an assessment of progress versus plan at checkpoints. The challenge in this process is that of obtaining a direct link between costs and technical progress. The fact that the expected amount of money has been spent by a given date does not necessarily indicate that the project is on schedule. Here again, our emphasis on tying the cost plan to the activities of the life cycle helps us. Expenditures are not merely projected on a month-by-month basis; they are related to specific work components and completion dates. Thus, reviews assess cost status, technical status, and expected cost status for the given technical status. A variance between actual results and expected results indicates potential problems or areas for improvement.

Figure 3 Design-to-cost



Modifications or growth in the estimated scope of work, deviations in expected productivity, or erroneous initial assumptions may require another cost planning and estimation cycle. For each iteration, the planning data are retained in the project data file for subsequent cost performance monitoring. The life-cycle model provides a checklist for assessing all the implications of changes, and the project data file provides actual performance data for cost planning and estimation.

design-to-cost

Cost management, as described, yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and management practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method, illustrated in Figure 3, consists of developing a design, estimating its cost, and ensuring that the design is cost-effective. To do this, design-to-cost goals are estab-

lished, based on an understanding of the capabilities of the software and the related design solution. Plans to achieve these goals are developed by allocating costs to particular work components of the software system life cycle.

Design is an iterative process in which each design level is a refinement of the previous level. At each stage, design and cost alternatives are examined. Those that best satisfy the project objectives are prepared for review and selection by the project sponsor. If no alternative fits the cost target, several courses of action are available. The most common one is to go back to the designers and ask for a less costly, and perhaps less attractive, design. If the target has been missed by a large amount—and cost is critical—redesign may not produce an answer. In this case, the sponsor has to consider giving up some of the planned capability of the system. Otherwise, he has to recognize that the capability cannot be acquired without increasing the cost target. The design process is followed until the program design for a specific software increment has been completed. From that point, development of each increment can proceed concurrently with the program design of the others.

When the development and test of an increment are complete, an estimate to complete the remaining increments is computed. The algorithms used in this computation should reflect the various actual productivity rates experienced in developing and testing previous increments. An alternative plan is prepared and reviewed, as previously described, whenever a cost projection is inconsistent with its cost plan. This may also require changes to a baseline design.

Thus software cost management practices provide a uniform methodology for planning, estimating, measurement, and control. The life-cycle definition provides a structure for the identification of cost-estimating parameters and a standard set of references for the entire development process from proposal through contract performance. The design-to-cost practice describes the management control procedures that balance cost, schedule, and functional capability.

software program management

Practices described thus far are directed at all project participants and department managers. They deal with specific details of design and implementation. They also cover technical and cost-control procedures. One more set of practices is needed to hold the software engineering program together. This final set is directed at the program manager and identifies the project-level plans, controls, and technical management considerations that are necessary for effective software development in a functional organization. Since, in a functional organization, project resources are drawn from several discipline-oriented departments, the program

manager does not have direct line authority over all the participants. Program management is much like managing subcontractors in a building construction project. Each subcontractor is highly qualified in a fairly narrow area and is most effective when carrying out a task in that area. The program manager, then, must define tasks clearly, assign them to appropriate departments, define working relationships and technical interfaces between departments, and establish reports and controls to see that the functional groups are carrying out their assignments.

Program management responsibilities include developing and maintaining an organization plan that identifies departments involved, their reporting relationships, and individual department charters. A project work responsibility matrix lists each work task, the responsible manager, and the prime and support roles of the functional groups. The work responsibility matrix should include a dependency network that indicates the predecessor and successor relationships for each task. All cost accounting, status reporting, management accountability, and technical performance are structured around the listed tasks.

From a control viewpoint, program management responsibility includes ensuring system requirement traceability through design and test, providing an architecture control methodology, and managing computer resource loading reserves. These reserves are determined at design time to accommodate design and implementation uncertainties.

Each functional group—hardware engineering, software engineering, product assurance, etc.—must respond to the program manager with a plan of action and a commitment to carry out that plan to fulfill the assigned responsibility.

The software function produces a Software Development Plan (SDP) that describes each assigned task from the work responsibility matrix. The description should cover product schedule, intermediate milestones and schedules, and external dependencies with required scheduled dates. The program manager is responsible for monitoring and controlling the external dependencies specified by the software function. The SDP is updated monthly and incorporates accomplishments, problems, and plans for the subsequent month. It is used as a control document within the software function and as a coordination document in the program manager's office.

Concluding remarks

The software management practices describe the plans and controls for the software engineering environment. These plans and

controls reflect the business responsibility of the software function to increase the visibility and understanding of its developing product. Cost management emphasizes cost estimation planning and control. Program management emphasizes effective communication between the software function and the other functions on the project. The technical reviews and design-to-cost practices integrate the application of the design and development practices with the management of the software process. Collectively, these components of software engineering define a structured and predictable approach to managing software projects.

Application of these practices in the Federal Systems Division has improved our ability to predict program behavior. Capability, cost, and schedule still vary from initial program estimates; however, the variances are typically less than experienced in the past. Early identification of deviations from plan has led to timely corrective action. The net result is that the integrated software engineering practices of FSD permit us to deliver high-quality, cost-effective software products with low business risk.

CITED REFERENCES

1. H. D. Mills, "Software development," *IEEE Transactions on Software Engineering* **SE-2,** No. 4, 265-273 (December 1976).
2. O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*, Academic Press, Inc., New York (1972).
3. C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM* **12,** No. 10, 576-583 (October 1969).
4. R. C. Linger, H. D. Mills, and B. L. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley Publishing Co., Inc., Reading, MA (1979).
5. N. Wirth, *Systematic Programming: An Introduction*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1976).
6. N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973).
7. A. B. Ferrentino and H. D. Mills, "State machines and their semantics in software engineering," *Proceedings of IEEE Comsac '77*, IEEE Catalog No. 77Ch1291-4C, 242-251, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854 (1977).
8. H. D. Mills, *On the development of systems of people and machines*, Springer-Verlag, New York (1975).
9. D. L. Parnas, "The use of precise specifications in the development of software," *Proceedings of IFIP Congress 77, Toronto, August 8-12, 1977*, B. Gilchrest, Editor, North-Holland Publishing Co., New York (1977), pp. 861-867.
10. P. Brinch Hansen, *The Architecture of Concurrent Programs*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1977).
11. C. A. R. Hoare, "Monitors: An operating system structure concept," *Communications of the ACM* **17,** No. 10, 549-557 (October 1974); "Corrigendum," *Communications of the ACM* **18,** No. 2, 95 (February 1975).
12. N. Wirth, "Toward a discipline of real-time programming," *Communications of the ACM* **20,** No. 8, 577-583 (August 1977).
13. H. D. Mills, "Software engineering," *Science* **195,** No. 4283, 1149-1205 (March 18, 1977).
14. G. M. Weinberg, *The Psychology of Computer Programming*, Van Nostrand Reinhold Co., New York (1971).
15. F. T. Baker, "Chief programmer team management of production programming," *IBM Systems Journal* **11,** No. 1, 56-73 (1972).

16. M. A. Jackson, *Principles of Program Design*, Academic Press, Inc., New York (1975).
17. B. W. Boehrn, "Software and its impact: A quantitative assessment," *Datamation* **14,** No. 5, 48-59 (May 1973).
18. R. W. Wolverton, "The cost of developing large-scale software," *IEEE Transactions on Computers* **C-23,** No. 6, 615-636 (1974).
19. T. C. Jones, "Measuring programming quality and productivity," *IBM Systems Journal* **17,** No. 1, 39-63 (1978).
20. C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Systems Journal* **16,** No. 1, 54-73 (1977).
21. R. Yeh, Editor, *Current Trends in Programming Methodology*, Vol. 1, Prentice-Hall, Inc., Englewood Cliffs, NJ (1977).
22. G. J. Myers, *Composite/Structured Design*, Van Nostrand Reinhold Co., New York (1978).
23. R. C. McHenry and C. E. Walston, "Software life-cycle management: Weapons process developer," *IEEE Transactions on Software Engineering* **SE-4,** No. 4, 334-344 (July 1978).
24. F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley Publishing Co., Inc., Reading, MA (1975).
25. C. L. McGowan and R. C. McHenry, "Software management," *Research Directions in Software Technology*, P. Wegner and W. Wolf, Editors; to be published.
26. E. A. Goldberg; "Applying corporate software development policies," *Software Development: Management, the Seventy-First Infotech State of the Art Conference*, London, May 12-14, 1980, Infotech, Ltd., Maidenhead, England (1980).
27. E. W. Dykstra, "Co-operating sequential processes," *Programming Languages*, Academic Press, Inc., London (1968), pp. 43-112.
28. M. V. Wilkes, "The outer and inner syntax of a programming language," *The Computer Journal* **11,** 260-263 (May-November 1968).
29. B. H. Liskov and S. N. Zilles, "An introduction to formal specifications of data abstractions," *Current Trends in Programming Methodology*, Vol. 1, R. Yeh, Editor, Prentice-Hall, Inc., Englewood Cliffs, NJ (1977), pp. 1-32.
30. M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal* **15,** No. 3, 182-211 (1976).
31. J. D. Aron, "Estimating resources for large programming systems," *Software Engineering, Concepts and Techniques*, P. Naur, B. Randell, and J. N. Buxton, Editors, Petrocelli/Charter, New York (1976).
32. M. R. Seldon, *Life Cycle Costing: A Better Method of Government Procurement*, Westview Press, Inc., Boulder, CO (1979).

The author is located at the IBM Federal Systems Division, 18100 Frederick Pike, Gaithersburg, MD 20760.

Reprint Order No. G321-5133.