Software Engineering Project Standards

MARTHA BRANSTAD AND PATRICIA B. POWELL

Abstract—**Software Engineering Project Standards (SEPS) and their importance are presented in this paper by looking at standards in general, then progressively narrowing the view to software standards, to software engineering standards, and finally to SEPS. After defining SEPS, issues associated with the selection, support, and use of SEPS are examined and trends are discussed. A brief overview of existing software engineering standards is presented as the Appendix.**

Index Terms—**Project management, software development, software engineering, software engineering standards, software management, software standards.**

I. Introduction

A STANDARD can be: 1) an object or measure of comparison that defines, represents, or records the magnitude of a unit, 2) a canonical form or characterization that establishes allowable tolerances or constraints for categories of items, 3) a degree or level of required excellence, confidence, assurance, or attainment. Fig. 1 displays this breakdown and examples from each category. Measurement standards provides a metric by which an entity can be measured or compared, e.g., a standard meter or standard time. An interchange standard provides a norm for product compatibility, e.g., standard light socket or language standard. A performance standard provides a categorization or descriptive framework, e.g., a grading system.

Standards are definitional by nature, either established to further understanding and interaction, or as observed norms of exhibited characteristics or behavior. Standards are formulated when there is motivation for control of variability. Many institutionalized standards have been established to enable commercial interaction and division of labor (e.g., standards for weights, machine components, etc.). Other standards clarify quality aspects implied by the use of given terms (e.g., U.S. meat grades).

Early software standardization efforts emphasized interchange of products. Standards were defined to control variability and to facilitate software transportability. Programs and data produced or accumulated at one site needed to be moved to computers, perhaps produced by a different manufacturer, at another site. The Code for Information Interchange (ASCII) standard (Federal Information Processing Standard 1-1) and the Cobol language standard (Federal Information Processing Standard 21-1) are examples of standards established to facilitate interchange. In contrast, current software engineering standards are motivated primarily by a desire to establish levels of confidence or definitional aspects of software rather than to foster global interchange of software [27]. Many of the current software standardization efforts are motivated by

Manuscript received July 13, 1982.

The authros are with the Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC 20234.

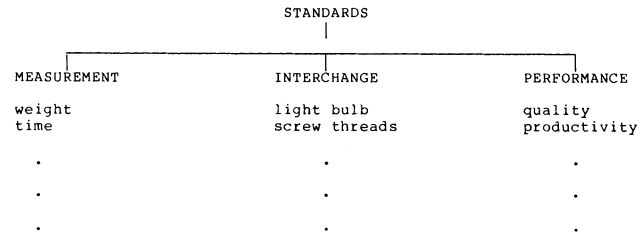


Fig. 1. Categories of Standards.

management's desire to control software quality and improve productivity in order to achieve better control of economic issues, for example, resource estimation (how to estimate software development cost, development time, required resources, and reliability).

Efforts to establish software engineering standards for large, heterogeneous populations must reach a compromise between the desire for specific, detailed standards and the diverse needs of the constituent bodies. Software engineering standards issued by IEEE, Federal Information Processing Standards (FIPS), and the American National Standards Institute (ANSI) are examples. Particularly when quality control rather than product interchange is the underlying motivation, "global" standards tend toward the definition of general frameworks, planning guides, or tailorable standards. These are carefully defined to serve as a basis for communication, general quality control, and the establishment of norms of good practice, while providing leeway for the use of diverse development techniques and approaches.

The rather general, global standards serve a very real need for industry-wide norms and definitions; however, for use within a software development project more specific standards are appropriate. It is in this domain, that of a specific software development project, that Software Engineering Project Standards (SEPS) should be established and used. Within the project, the need for specificity does exist, for materials must be communicated and shared among members of the development team. SEPS serve the needs for both quality control and information interchange to support division of labor. SEPS can be very specific since the standards are being established for a particular project with known characteristics. SEPS are (or should be) the embodiment of project development policy. They should establish the development methods to be used; the specific requirements, design, and coding techniques and languages; the verification, validation and testing (VV&T) approach; the form and type of records to be kept and controlled; and the official documents that should be produced. The specific nature of the SEPS and the limited domain in which they are applied create both opportunity and challenges which are discussed later in the article.

To synopsize the preceding discussion, Software Engineering Project Standards are norms of required practice established

by the project manager in order to produce a software system of uniformly high quality and to facilitate communication and interchange within the project. The SEPS comprise a collection of standards which cover both management and technical aspects of software development and provide the general framework in which software development is performed. When defined in their entirety, SEPS represent an embodiment of project development policy.

II. Issues

The successful establishment and use of SEPS involves significant understanding and insight into the state of current software technology, human nature, people's ability to deal with change, and the goals of the particular organization and project. A number of specific issues associated with the selection, introduction, support, and use of SEPS are presented below. See [10], [26], [29] for additional discussion of standards issues.

Standards and Measurability

By their very nature, standards involve measurement. Since a standard is a model or rule against which other objects are compared or measured, it is essential that it be possible to determine if the candidate complies with the standard or is within an acceptable tolerance of the standard. Although this may seem obvious, it is often overlooked. This is particularly true in the software engineering arena where the desire for increased quality and productivity is a strong force behind SEPS. Software quality is usually defined as a collection of characteristics, e.g., efficiency, maintainability, testability [18], with the importance of each specific characteristic varying from project to project. However, the quality characteristics are difficult to measure directly. Two different approaches to this dilemma have arisen. The first approach standardizes those properties which are amenable to measurement even though it is recognized that the properties are secondary or support characteristics and do not guarantee quality software. Module size, control structure complexity, and naming conventions are examples. The second approach concentrates upon the process by which the software product is produced rather than on the characteristics of the product itself. To effect the second approach, specific steps in the development process are standardized both with respect to their occurrence and to the techniques used to accomplish the step. Although both types of software engineering standards are used in the collection comprising the SEPS, the process standards are currently viewed as the most important [3], [30].

A mild warning should be given concerning measurement of project characteristics. People are very effective at almost unconscious behavior modification in response to what is perceived as desirable or rewarded behavior. If lines of code are measured and coders believe that more is better, more will be produced. Only important and meaningful characteristics should be emphasized.

Standards and Technology

Software engineering standards are dependent upon the technology that they represent and serve. They cannot outpace the technology, neither should they curtail or suppress it. The standards should incorporate stable technology which is available for distribution and use. Technical feasibility alone is insufficient; the underlying techniques and the support required to implement the standards must be readily available. Standards for dynamic technology involve a fundamental conflict between the stability implied by the standard and the change inherent in the technology. Software engineering standards must deal with this issue. Not only is computer hardware still evolving, but the software engineering tenets that the standards should embody are still under development. Although there is general agreement about the more fundamental principles, there are many open issues. The solutions have a significant likelihood of incorporating new insights, rather than being only further refinements of existing "truths." The need for future modification must be anticipated when establishing software standards. Since the implied duration is more limited for project standards, this problem of adapting the standards to a dynamic technology is somewhat less severe than for global standards. Each project manager has the responsibility for defining SEPS; however, the cost associated with introducing SEPS that differ significantly from those used previously must be addressed. For both software engineering standards and SEPS the cost of change must be weighed against the cost of not changing.

Introduction and Implementation of Standards

Management reaps the potential benefits of SEPS: decreased variability, increased product quality, increased worker productivity, facilitated communication, and better control [2]. Management must make a definite commitment for a SEPS effort to come to fruition and be effective. The introduction of SEPS will, in most cases, involve a change in the manner in which work has been performed previously [31]. Change is difficult for most people and hence they tend to resist it. Therefore, anytime a change is made and it is desired that it be successfully incorporated, effort must be expended to facilitate the transition. The change, in this case the introduction of SEPS, must be motivated. (It is assumed that the SEPS were carefully selected so that they are reasonable and understandable for the given project environment.) An education program should be initiated. People must understand the SEPS in order to use them. If the SEPS represent a significant departure from previous software development procedures, they should be introduced in a phased manner. Automated support for the SEPS should be provided. Software tools that actually embody the standards are most effective. Structure preprocessors and compilers are examples. When appropriate automation is available, it becomes easier to perform the work the standard way than by any alternative means. In such instances, standards audit or enforcement becomes transparent, since the development process incorporates the standard.

Global Software Engineering Standards Versus Local SEPS

In most cases, the project manager does not have complete license in defining the SEPS since at least a portion of the standards are imposed upon him by the organization in which the project exists and the client for whom the work is being done. In addition, the industry is developing voluntary standards which should be considered. Reconciliation of sometimes

conflicting sets of software standards is an initial step in defining the SEPS to be used during a software development project. Frequently, the process involves tailoring general software engineering standards to provide specific standards for the given project. The limited domain and specific nature of SEPS provides opportunity for the selection of particular development techniques. While agreement upon a single technique is unlikely (and inappropriate) in the global arena, there is economic pressure to have SEPS extend beyond the boundary of a single project (unless the project is extremely large and of significant duration.) The overhead costs for training and automated support necessary to introduce and implement the SEPS successfully are more attractive when amortized over several projects within an organization. To the extent permitted by the compatible nature of projects within an organization, the economics suggest that SEPS should be supported throughout the organization.

SEPS and Software Quality Assurance

Software quality assurance (SQA) is usually approached by control of the development process. The process is specified by developing standards, while quality assurance auditing determines compliance with the standards. Software standards and SQA are companion processes. SEPS provide the basis from which to perform SQA. When developing SEPS, it is of prime importance to select standards which are quantifiable and hence measurable for an objective audit by the SQA group. The SQA group should be organizationally independent from the development effort to maintain its functional perspective [16].

III. Candidates for SEPS Selection

The SEPS should be the embodiment of the development policy for the project. As such, the specific standards established will vary from project to project, but should address key aspects of software development. Since the SEPS are comprised of a collection of standards, each relating to a phase or specific aspect of the development process, the individual standards must be compatible and consistent with one another. For example, coding techniques must be compatible with the language being used. The following sections discuss categories of candidate standards for inclusion in SEPS. A number of software life cycles with somewhat different phases have been defined and used within the industry. For illustrative purposes, a four-phase life cycle is used in the following discussion: requirements, design, construction, and maintenance. Activities which span the entire life cycle are termed overview activities.

Overview Activities

Three areas of overview activities are candidates for standardization: verification, validation, and testing (VV&T); configuration management; and documentation. VV&T activities take place throughout the life cycle to ensure the correctness and quality of the software. VV&T items that should be defined in the SEPS include: VV&T planning documents, review points, specific verification techniques for each life cycle phase, verification techniques to ensure consistency of products between phases, testing standards including coverage, and records for completed VV&T activities. Configuration management controls documents and products produced during development and the changes made to them. Aspects of configuration management that should be defined in the SEPS include: version identification codes, change control methods, identification of the documents that are to be controlled, and auditing procedures. Documentation is a key to the success of software development. Two categories of documents exist, development records and user-oriented manuals. Development documentation provides the information needed to manage and control the project. SEPS should define the types of documents required, the material they should include and the formality of the presentation. Management and planning documents are included as a subset of the project documentation standards. The overview activities are particularly critical areas for SEPS since they impact all phases of development and are central to record keeping, communication within the project, management control, and general quality control issues.

Requirements

The requirements phase begins with a project proposal and ends with a requirements specification document which provides the baseline for product validation. Although the complete requirement development process is not amenable to standardization at this time, at the project level, a specific method for recording the requirement specification can be standardized. The method should support both documentation and analysis of the requirements and should facilitate the development of a complete, consistent, and comprehensible requirement specification. See [1] for a discussion of candidate methods and techniques to be used throughout the development cycle.

Design

During the design phase, the specified requirements are transformed into detailed designs from which code can be produced. Frequently, this phase is organized as a two step process with preliminary, high-level designs being produced and verified before being further developed into detailed designs. The SEPS should designate a particular method to be used for design, and should further clarify terminology, structuring, complexity, size, and interface constraints as appropriate to the method selected.

Construction

The construction phase includes coding, integration, test, and installation of the accepted system. This phase has highly visible products which can be measured against predetermined criteria, e.g., module size. SEPS for this phase should establish the high-level language to be used (including special exceptions concerning the use or exclusion of specific language features), naming conventions, module size, code complexity, desired code structuring, commenting and in-line documentation conventions, and interface constraints. Testing standards for the construction phase should be established as a subset of the VV&T standards. Most organizations have standards which delineate the method for code development. Use of automation to determine compliance (code auditing) is becoming more common. Although much standardization activity centers upon this phase, the characteristics being controlled are

of somewhat less importance to the quality of the final software product than are the proper development of requirements and design.

Maintenance

The maintenance phase starts when the software has been installed and accepted and ends when the software is replaced or discarded. Although different management is usually responsible for the software during the maintenance phase, the SEPS are still important. Studies show [17] that 50 percent or more of maintenance activities are software enhancements, work that could easily be described as development in a "constrained" environment. The remaining maintenance activities can be categorized as corrective (error fixing) and adaptive (e.g., modifications required by a changing hardware for software systems environment.) Throughout all these maintenance operations, the original SEPS should be used and enforced. In many cases, the most significant economic return on the investment in software standards comes during the maintenance phase. Unless the use of SEPS continues, however, the improved control and manageability will degrade with each modification to the software.

V. Trends

There has been a significant increase in interest in software engineering standards in recent years, as indicated by the number of IEEE standards committees and working groups that have been organized. The emphasis on software engineering standards is expected to continue in the future with a collection of global software engineering standards being established by IEEE. For the Federal government, a number of FIPS guidelines for various aspects of software engineering are either recently completed [8], [9], in process, or planned. Updated guidance for software documentation, additional guidance for VV&T and acceptance testing, maintenance guidelines, and recommendations for the use of tools throughout the development process are planned. U.S. military standards continue to exhibit an increased emphasis and concern for software engineering standards. The effort that has gone into the Joint Services Defense Systems Software Development Standard is representative.

It is expected that the future will bring an increased effort in planning, organizing, and standardizing the software development process. Software developers will continue to realize the economic benefits of standardization of their processes while national standards will provide modifiable guidelines that can be adapted to specific projects. The trend toward standardizing parts of the software development process will continue, motivated in part by the use of automation to support the development process.

Appendix
Software Engineering Standards

Several national standards producing bodies have established a number of global software standards. Software engineering, however, is a relatively recent area for standardization activity. Nevertheless, some software engineering standards already exist, some have been adapted from hardware standards, and others are currently in development. Summarized below are global software engineering standards, in use or being drafted, from three national sources of standards: IEEE standards, FIPS, and military standards. These software engineering standards are candidates for use during software development and hence impact the SEPS to be established by the project manager. The summaries are extracted from materials available in the Fall of 1983.

IEEE Software Engineering Standards

IEEE Standard for Software Quality Assurance Plans (ANSI/IEEE STD 730-1981).

"The purpose of the standard is to provide uniform, minimum acceptance requirements for preparation and content of Software Quality Assurance Plans. This standard applies to the development and maintenance of critical software. . . . For noncritical software, or for software already developed, a subset of the requirements of this standard may be applied" [14]. The standard includes the following major topics: purpose; referenced documents; management; documentation; standards, practices, and conventions; review and audits; configuration management; problem reporting and corrective action; tools, techniques, and methodologies; code control; media control; supplier control; and records collection, maintenance, and retention.

IEEE Standard Glossary of Software Engineering Terminology (ANSI/IEEE Std. 729-1983).

The document is a glossary of terms in general use in the field of software engineering [13].

IEEE Guide for Software Requirement Specifications (IEEE Std. 830-1983).

"This document is a guide for writing software requirements specifications. It describes the necessary content and qualities of a good Software Requirements Specification (SRS) and presents a prototype SRS outline" [15]. The guideline discusses what a requirement is; qualities of requirements such as unambiguity; completeness, verifiable, consistency, and traceability; a typical outline for a SRS; the evolution of a SRS; methods used to express software requirements; and tools for developing a SRS.

IEEE Standard for Software Configuration Management Plans (IEEE Std. 828-1983).

"The purpose of this standard is to provide minimum requirements for the preparation and content of Software Configuration Management (SCM) Plans. . . . This standard applies to the development and maintenance of any kind of software" [11]. The standard discusses the SCM environment overview, i.e., system description, life cycle, and SCM roles, responsibilities, and interfaces; SCM organization, e.g., organizational structures responsibilities and authorities; SCM activities; tools; releases and libraries; SCM throughout the life cycle; and SCM resource requirements.

IEEE Standard for Software Test Documentation (ANSI/IEEE Std. 829-1983).

"This standard describes a basic set of test documents

which are associated with the process of analyzing computer programs in order to detect faults and estimate the risk of failure. . . . It is applicable to commercial, scientific, or military, software which runs on any digital computer." [12] The basic documents discussed are the test plan, test design specification, test case specification, test procedure specification, test item transmittal form, test log, test incident report, and test summary report.

New IEEE standards working groups are continually forming. Those which have project authorization are: *A Standard for Software Reliability Measurement* (P982), *A Guide for Software Quality Assurance* (P983), *A Guide for the Use of Ada as a PDL* (P990), *Software Engineering Standards Taxonomy* (P1002), *A Standard for Software Unit Testing* (P1008), *A Standard for Software Plans* (P1012), *A Guide for Software Design Documentation* (P1016), and a revision to *A Standard for Software Quality Assurance Plans* (P730-1).

Federal Information Processing Standards and Guidelines

Guidelines for Documentation of Computer Programs and Automated Data Systems (FIPS PUB 38).

This guideline addresses the content of ten documents for the development phase of the life cycle. The documents are: functional requirements, data requirements, system/subsystem specification, program specification, database specification, user's manual, operations manual, program maintenance manual, test plan, and test analysis report [6].

Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase (FIPS PUB 64).

This guideline provides a basis for determining the content and extent of documentation for the initiation phase of software development. Guidance is given for the following document types: Project Request, Feasibility Study, and Cost/Benefit Analysis [7].

Guideline: A Framework for the Evaluation and Comparison of Software Development Tools (FIPS PUB 99).

This document is designed to be used as a reference and suggests a framework which aids in identifying, discussing, evaluating, and comparing software development tools. It is recommended for use when acquiring or implementing tools, developing policies or procedures for tools, and reviewing the current use of tools [8].

Guidelines for Lifecycle Validation, Verification, and Testing of Computer Software (FIPS PUB 101).

"This guideline recommends that validation, verification, and testing (VV&T) be performed throughout the software development lifecycle. The selection and use of a collection of validation, verification, and testing techniques to meet project requirements is presented. The guideline explains how to develop a VV&T plan which will fulfill a specific project's VV&T requirements" [9].

U.S. Military Standards

Specification Practices (MIL-STD-490).

The standard specifies how a specific type of computer program development specification should be prepared [24].

Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs (MIL-STD-483).

The standard provides for a configuration management plan, configuration identification, baselines, change control, and audits; it is hardware oriented [23].

Technical Reviews and Audits for Systems, Equipment, and Computer Programs (MIL-STD-1521A).

The standard calls for five reviews and two audits. These are: system requirements review, system design review, preliminary design review, critical design review, formal qualifications review, functional configuration audit, and physical configuration audit [20].

Software Quality Assurance Program Requirements (MIL-S-52779A).

The standard requires that the contractor develop and implement a Quality Assurance Program specifically for software. To accomplish this, the program must provide for detection, reporting, analysis, and correction of software deficiencies [25].

Weapon System Software Development (DOD-STD-1679A).

This standard address the complete software development process. It includes the following areas: software performance requirements; software design requirements, programming standards and conventions; software implementation; program regeneration; testing; software operation; software quality assurance; software acceptance; software configuration mangement; and software management planning [22].

Tactical Digital System Standard, Software Quality Assurance Testing Criteria (TADSTRAND 9).

The key requirements addressed are software endurance runs, third party conduct of endurance runs which are part of acceptance, allowable software errors, and allowable patches. Endurance runs include stress loading, degrading modes, and on-line maintenance support programs [28].

Configuration Control–Engineering Changes, Deviations, and Waivers (DOD-STD-480A).

This standard incorporates specific instructions for preparing software engineering change proposals [5].

Military Standard Defense Systems Software Development (draft of proposed MIL-STD-SDS).

The standard requires contractors to have software project management which provides an acceptable level of visibility into the development process and additional requirements for: structured and modular software architecture, requirements analysis, approved high order language, software integration testing, configuration management, software quality assurance, project planning and control, and subcontractor control [19]. This proposed standard began its approval process in 1982. Recommended changes from the Joint Logistics Commanders to MIL-STD-490, MIL-STD-483, and MIL-STD-1521A are included to make them compatible with MIL-STD-SDS.

Much of the material on military standards comes from [32] and [4].

References

[1] B. W. Boehm, "Software engineering," *IEEE Trans. Comput.*, vol. C-25, Dec. 1976.

[2] ——, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[3] M. A. Branstad and P. B. Powell, "Process standards for software engineering," in *Proc. Software Eng. Standards Application Workshop*, Aug. 1981, IEEE Comput. Soc. Press.

[4] J. D. Cooper, CACI Inc., Arlington, VA, lecture notes and conversations.

[5] *Configuration Control—Engineering Changes, Deviations, and Waivers*, DOD-STD-480A, 1978.

[6] *Guidelines for Documentation of Computer Programs and Automated Data Systems*, FIPS PUB 38, Feb. 1976.

[7] *Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase*, FIPS PUB 64, Aug. 1979.

[8] *Guideline: A Framework for the Evaluation and Comparison of Software Development Tools*, FIPS PUB 99, Mar. 1983.

[9] *Guideline for Lifecycle Validation, Verification, and Testing of Computer Software*, FIPS PUB 101, June 1983.

[10] G. N. Fostel, "Principles of software standardization," in *Proc. Software Eng. Standards Application Workshop*, Aug. 1981, IEEE Comput. Soc. Press.

[11] *IEEE Standard for Software Configuration Management Plans*, IEEE Std. 828-1983, IEEE Comput. Soc., New York, NY, 1983.

[12] *IEEE Standard for Software Test Documentation*, ANSI/IEEE Std. 829-1983, IEEE Comput. Soc., New York, NY, 1983.

[13] *IEEE Standard Glossary of Software Engineering Terminology*, ANSI/IEEE Std. 729-1983, IEEE Comput. Soc., New York, NY, 1983.

[14] *IEEE Standard for Software Quality Assurance Plans*, ANSI/IEEE Std. 730-1981, IEEE Comput. Soc., New York, NY, 1981.

[15] *IEEE Guide for Software Requirements Specifications*, ANSI/IEEE Std. 830-1983, IEEE Comput. Soc., New York, NY, 1983.

[16] B. M. Knight, "Organizational planning for software quality," in *Software Quality Management*. Petrocelli Books, 1979.

[17] B. P. Lientz and E. B. Swanson, *Software Maintenance Management*. Reading, MA: Addision-Wesley, 1980.

[18] J. McCall *et al.*, *Factors in Software Quality, Concept, and Definitions of Software Quality*, vol. 1, Nat. Tech. Inform. Service, AD/A-049 014, Nov. 1977.

[19] *Military Standard Defense System Software Development*, Proposed MIL-STD-SDS, Joint Logistics Commanders, Apr. 1982.

[20] *Technical Reviews and Audits for Systems, Equipment, and Computer Programs*, MIL-STD-1521A, USAF, Sept. 1978.

[21] *Weapons Systems Software Development*, DOD-STD-1679A, Navy, Feb. 1983.

[22] *Configuration Control—Engineering Changes, Deviations, and Waivers*, DOD-STD-480A, 1978.

[23] *Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs*, MIL-STD-483a, USAF, Dec. 1970.

[24] *Specification Practices*, MIL-STD-490A, Oct. 1968.

[25] *Software Quality Assurance Program Requirement*, MIL-STD-52779A, Army, 1977.

[26] D. T. Ross, "Homilies for humble standards," *Commun. Ass. Comput. Mach.*, vol. 19, Nov. 1976.

[27] *Software Engineering Standards Application Workshop*, Aug. 1981, Comput. Soc. Press.

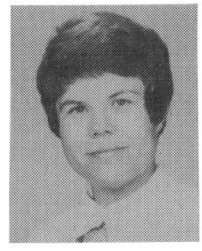
[28] *Standard Tactical Digital System Software Qualty Assurance Testing Criteria*, TADSTAND 9, Navy, 1978.

[29] L. L. Tripp, "Top-down, bottom-up approach to software engineering standards," in *Proc. Software Eng. Standards Application Workshop*, Aug. 1981, IEEE Comput. Soc. Press.

[30] R. L. Van Tilburg, "Process standardization versus product standardization," *Software Eng. Standards* 1981.

[31] G. H. Wedberg, "Implementation of software engineering standards," in *Proc. Software Eng. Standards Application Workshop*, Aug. 1981, IEEE Comput. Soc. Press.

[32] D. L. Wood, "Department of Defense software quality requirement," in *Software Quality Management*. Petrocelli Books, 1979.



Martha Branstad received the B.S. degree in mathematics and the Ph.D. degree from Iowa State University, Ames, and the M.S. degree in mathematics from the University of Wisconsin.

She is Manager of the Software Engineering Group at the Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC. She has primary responsibility for developing software engineering standards and guidelines which foster higher quality software within the Federal government. Her group provides consulting services on software engineering techniques and technology to other agencies within the Federal government. Research in automated support to program construction and knowledge-based software development workstations is under her direction. Prior to joining NBS, she was the manager of a software research group at the National Security Agency. She has focused on natural language interfaces to databases, knowledge-based systems, software engineering, software development tools, and computer security. Previously she was an Assistant Professor of Computer Science at Iowa State University.

Patricia B. Powell received the B.A. degree from Smith College, Northampton, MA, and the M.S. degree in computer science from the University of Maryland, College Park.

She is a Computer Scientist at the Institute for Computer Sciences and Technology within the National Bureau of Standards, Washington, DC. Her current responsibilities are in the areas of technology forecasting and cost-benefit analysis. She also participates in IEEE standards working groups. Previously, she worked in the areas of software engineering, artificial intelligence, and the development of high-order languages.

Mrs. Powell is a member of the IEEE Computer Society and the Association for Computing Machinery.