SOFTWARE QUALITY ASSURANCE IN THE NATIONAL WATER INFORMATION SYSTEM

By George R. Dempster, Jr., and Charles F. Merk

U.S. GEOLOGICAL SURVEY
Open-File Report 91-218

Reston, Virginia
1991

U.S. DEPARTMENT OF THE INTERIOR
MANUEL LUJAN, JR., Secretary
U.S. GEOLOGICAL SURVEY
Dallas L. Peck, Director

For additional information write to:
Assistant Chief Hydrologist for Scientific
Information Management
U.S. Geological Survey
440 National Center
Reston, Virginia 22092

Copies of this report can be purchased from:
U.S. Geological Survey
Books and Open-File ReportsSection
Federal Center, Bldg. 810
Box 25425
Denver, Colorado 80225

CONTENTS

ILLUSTRATIONS

Figures:

SOFTWARE QUALITY ASSURANCE IN THE NATIONAL WATER INFORMATION SYSTEM

By George R. Dempster, Jr., and Charles F. Merk

ABSTRACT

The U.S. Geological Survey's automated information system's software for processing, storing, and retrieving water data is developed and maintained by the National Water Information System Program of the Water Resources Division. As part of that responsibility, a new system is currently being designed to replace the systems that run on a mainframe computer located at headquarters in Reston, Va., and on minicomputers located in Division offices across the Nation. The new system is to run on workstations and file servers to be installed in all of the Division offices. A software life cycle provides the framework for defining, developing, and maintaining the new software system and for helping to meet software development challenges.

Software quality assurance is needed in building the new system to provide confidence that the software is engineered to meet users' needs and conforms to established technical requirements. Successful software quality assurance depends on the application of uniform terminology and effective communication among users, developers, and managers. Software quality control is a planned and systematic process that focuses on establishing software standards and procedures, reviews and inspections, testing, and software configuration management throughout the software life cycle. This report explains the background for software quality assurance and the implementation of software quality assurance and quality control in the National Water Information System Program using a generic workbench approach, and presents an example workbench for the requirements-analysis phase of the software life-cycle.

INTRODUCTION

The U.S. Geological Survey's National Water Information System (NWIS) Program of the Water Resources Division (WRD) has the responsibility of defining, developing, and maintaining computer software for water-data processing, storage, and retrieval. The current effort is to design and develop software for a workstation system that employs file servers and local area networks to be located in WRD District and Subdistrict offices nationwide. The new system, called NWIS-II, will replace the existing headquarters mainframe system, called WATSTORE (WATer STOrage and REtrieval), and the Districts' minicomputer system, called NWIS-I. The new system is to provide a more powerful level of desktop computing capability than that of NWIS-I and will remain distributed nationally using an existing wide area network of communications. The result of the new software development will be an automated information system that processes water data functionally (input and edit, compute, verify, retrieve, and output) by integrating technical disciplines (biology, ground water, sediment, surface water, water quality, and water use). While defining and developing the new system, such intermediate software life-cycle products as (1) system planning, definition, and design documents, (2) computer source and executable code, and (3) test data are created or implemented. These intermediate products are subsequently used throughout the software life cycle. The life cycle is made up of distinct phases or steps that provide the framework for guiding and accomplishing the software definition, development, operation, and maintenance.

Quality assurance (QA) and quality control (QC) are established in the NWIS Program to assist in meeting today's software design and development challenges, fulfilling users' needs, and producing correct, reliable, and usable software. The quality-assurance effort applies and works throughout the

software life cycle. The effort has to be flexible because the life-cycle phases are iterative, and is successful when users, developers, and managers all participate in and support quality assurance. The quality-assurance effort for the software life cycle initially focuses on four quality-assurance cornerstone activities and quality-control workbenches (Quality Assurance Institute, 1988, p. 90). The four cornerstone activities consist of (1) preparing standards, (2) technically reviewing new software products, (3) testing software code, and (4) managing the configurations of the software products. The quality control workbench is visualized as a bench at which software work is done by software builders or developers. Software products, such as documentation, code, and data are built or developed at the workbench using technical procedures and tools that help to build in such quality attributes as correctness, reliability, and usability. An effective workbench operates by having software products delivered to it as input, work is done using the input products, and newly created or revised products are delivered from the workbench as output. Upon entry to and exit from the workbench, the software products are evaluated for quality (the quality control). The workbench evaluation criteria are developed based on the principles, processes, and methods used in each of the quality-assurance cornerstone activities.

Purpose and Scope

This report explains the background and need for software quality assurance and quality control in the NWIS Program, the use of four NWIS Program quality-assurance cornerstone activities, and the quality-assurance/quality-control workbenches to be implemented during current software development efforts. A software life cycle is implemented to form a structure for defining, developing, and maintaining the software. The quality assurance is viewed as a set of enveloping activities that surround and interweave with the life cycle.

The objective of the NWIS quality-assurance cornerstone activities is to establish a foundation or base for quality assurance and quality control of new software. A generic workbench approach that is applicable for all software life-cycle phases is used to describe and explain the implementation of quality assurance and quality control. Each workbench has four parts that rely upon the quality-assurance cornerstone activities for accomplishing the quality control. A software product will have reasonable assurance that it meets users' needs and conforms to technical requirements if the product is processed within a workbench. An example workbench for the requirements-analysis phase of the software life-cycle is presented to illustrate the practicality and effectiveness of the approach for implementing quality assurance and quality control.

National Water Information System Program

The NWIS Program, located at Headquarters in Reston, Va., manages and develops software for WRD's automated water information systems. The Program is organized with a Chief and three operational units, as shown in figure 1. The Program Chief has overall responsibility, and the unit names reflect their respective responsibilities. The following five WRD work groups are involved in the new software development effort:

1. Strategic Planning—WRD senior-level managers.
2. Users—Technical personnel in WRD District, Region, and Headquarters offices.
3. Quality Assurance—Quality Assurance and Configuration Management Unit personnel.
4. Integration—Data Base and Software Design and Development Unit personnel.
5. Maintenance—Data Management and Operations Unit personnel.

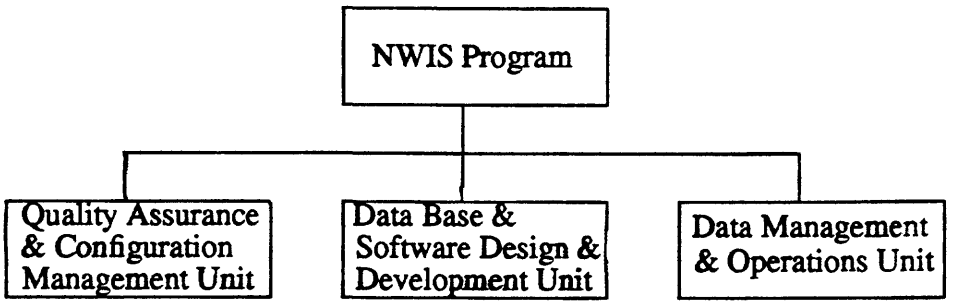

Figure 1.-Organization of the National Water Information System Program

The data base and software design and development also is assisted by technical personnel from WRD offices assigned to the Integration Group. All work group personnel communicate, integrate efforts, and work together to build the software system.

Terms and Definitions

Agreement on major software quality-assurance terms and definitions, and understanding differences between terms, is critical for effective and successful quality-assurance work and communication. Brief definitions of seven important terms used in this report and the NWIS Program are presented below. The first three are general, and the last four are specific to computer system software.

1. Quality—Attributes of a product or service that enables it to satisfy or meet stated or implied technical or functional needs.
2. Quality Assurance—Planned or systematic actions necessary to provide confidence that a product or service fulfills given quality attributes and improves upon processes and methods used to build products and provide services.
3. Quality Control—Operational techniques and activities used to fulfill the requirements of quality.
4. Software Engineering—Planned and systematic approach or discipline to the definition, development, maintenance, and retirement of computer systems software.
5. Software Quality Assurance—Planned process that systematically applies managerial precepts or principles, structured design and development disciplines and techniques, and improved technical processes and methods during the software engineering to ensure that software fulfills specified attributes of quality.
6. Software Quality Control—Planned process that applies all operational precepts, disciplines, and improvement methods to fulfill the requirements of software quality.
7. Software Configuration Management—Planned process that identifies, defines, and controls the change in software products; records and reports status of products and change requests; and verifies completeness and correctness of the products.

Understanding these terms and definitions by involved personnel facilitates more effective communication, management, and integration of software engineering and quality-assurance efforts.

Software Life Cycle

A software life cycle provides a sequence of phases (steps) to follow for software development from initiation through operation and maintenance. Figure 2 shows the current software life cycle and phases implemented for the NWIS Program. The overall process allows for overlap and iteration of phases. The

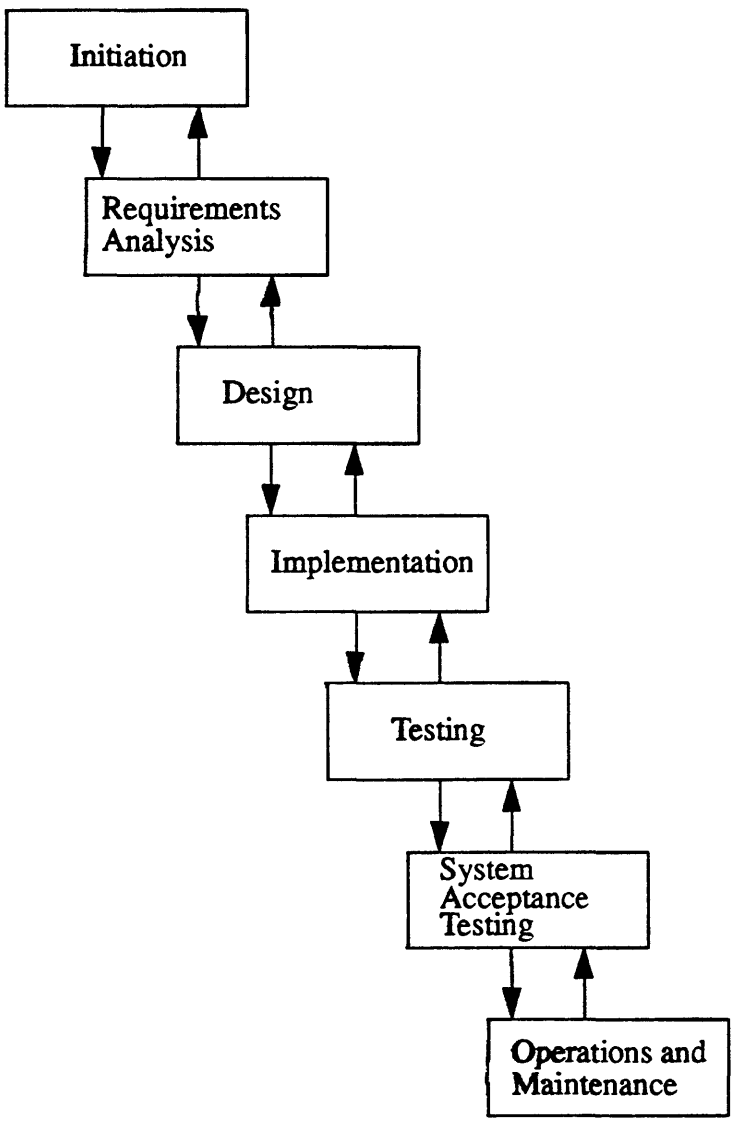

Figure 2.-Software life-cycle phases for the National Water Information System Program

output from one phase is input to the next, with feedback as necessary. The process is dynamic because it has to handle unexpected problems and planned changes. Following is a summary of the main activity for each phase.

1. Initiation—Formulate new computer system software objectives and goals, concepts, plans, tasks, and establish NWIS Program work groups.
2. Requirements Analysis—Prepare user software requirements, project development procedures and guidelines, technical and functional requirements specification, and begin preparing user's manual.
3. Design—Establish methods to perform system software and data base designs, prepare test plans, and begin maintenance manual.
4. Implementation—Perform coding and debugging and continue preparation of manuals.
5. Testing—Perform module (low-level) to total integrated system (high-level) tests and complete preparation of manuals.
6. System Acceptance Testing—Perform final user-acceptance tests of the system and complete preparation of system test reports, version description report, and system administrator's manual.
7. Operations and Maintenance—Train users and release the new workstation-based system.

In this report, the software development process contains the requirements-analysis through the testing phases. The above phases are complemented by other activities, including transfer of data from existing to new systems, quality assurance, quality control, and configuration management of the software products.

Software Development Challenges

Software development problems during the building of NWIS-II can lead to high costs, lack of productivity, schedule delays, and ineffective or unusable software. Problems may arise for the following reasons:

1. Inadequate software requirements definition.
2. Undocumented or imprecise software design specification.
3. Varied or inconsistent software implementation/coding practices.
4. Short-cut or inadequate software debugging (searching for errors) and testing.
5. Inadequate system interfacing and data access conflicts that result in poor system performance.
6. Unmet or unfulfilled user expectations or low user-satisfaction levels.

Figure 3 shows the "iceberg of software costs" due to all or some of these potential problems. The development costs of software may be recognizable and believed acceptable or controllable (for example, tip of the iceberg); however, unforeseen or unplanned problems in the software development can cause the operations and maintenance costs to become much more extensive (hidden part of the iceberg).

The solution to controlling and lowering total system costs lies in meeting today's software development challenges. The challenges are:

1. Adequate initial software engineering.
2. Automation of software engineering processes and methods.
3. Realistic planning, effective communication, and integration of efforts.
4. Narrowing the way the software is developed by defining (a) the processes and methods to be used and the sequence in which the methods are applied, (b) the deliverables required, (c) the controls that help to assure quality and coordination, and (d) the milestones that enable managers to assess progress.

5. Instituting software quality assurance, quality control, and configuration management of the software products.

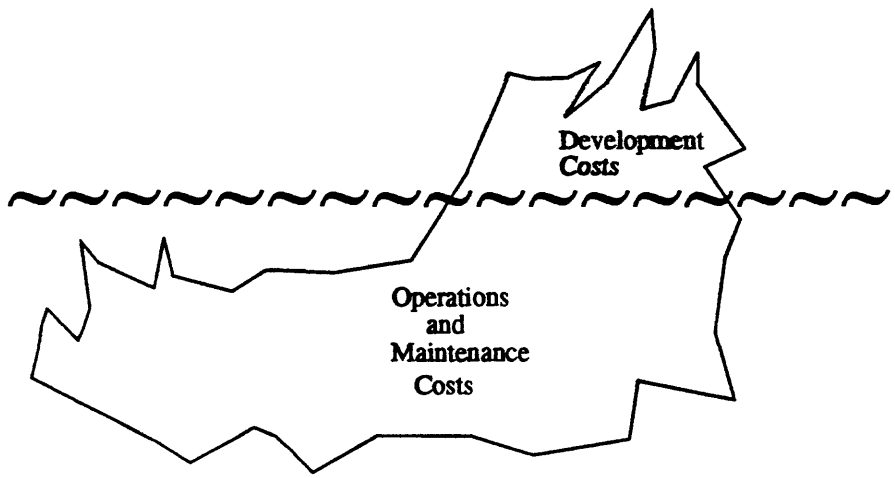

Figure 3.-The iceberg of software costs. (Modified from Quality Assurance Institute, 1989a, p. 62.)

These challenges are met by attacking the development as a whole rather than piecemeal, establishing a proper software engineering development environment, standardizing the software development processes, and making a commitment to developing quality software. Without the quality commitment, total software costs can remain high; with a commitment to quality, the distribution of preventive, appraisal, and failure costs (the costs of quality) can be changed and total costs reduced. Preventive costs are associated with planning for software quality before the software is built; appraisal costs are associated with reviewing software after it is built and before it is released to the users; and failure costs are associated with repairing problem software after it is released to the users.

QUALITY-ASSURANCE FRAMEWORK

Quality assurance is needed in conjunction with the NWIS software life cycle to assist in avoiding, controlling, and lessening the effects of definition, development, and maintenance problems. To provide a quality-assurance framework, the software quality-assurance activities are depicted as forming an envelope that surrounds the software life cycle (fig. 4). Most quality-assurance activities influence and become interwoven with the life-cycle phases, while others are central to a specific phase, such as testing. The cornerstone quality-assurance activities are shown at the corners in figure 4 and are discussed later. These four cornerstone activities are the basis for initial and continuing quality-assurance work in the U.S. Geological Survey's NWIS Program.

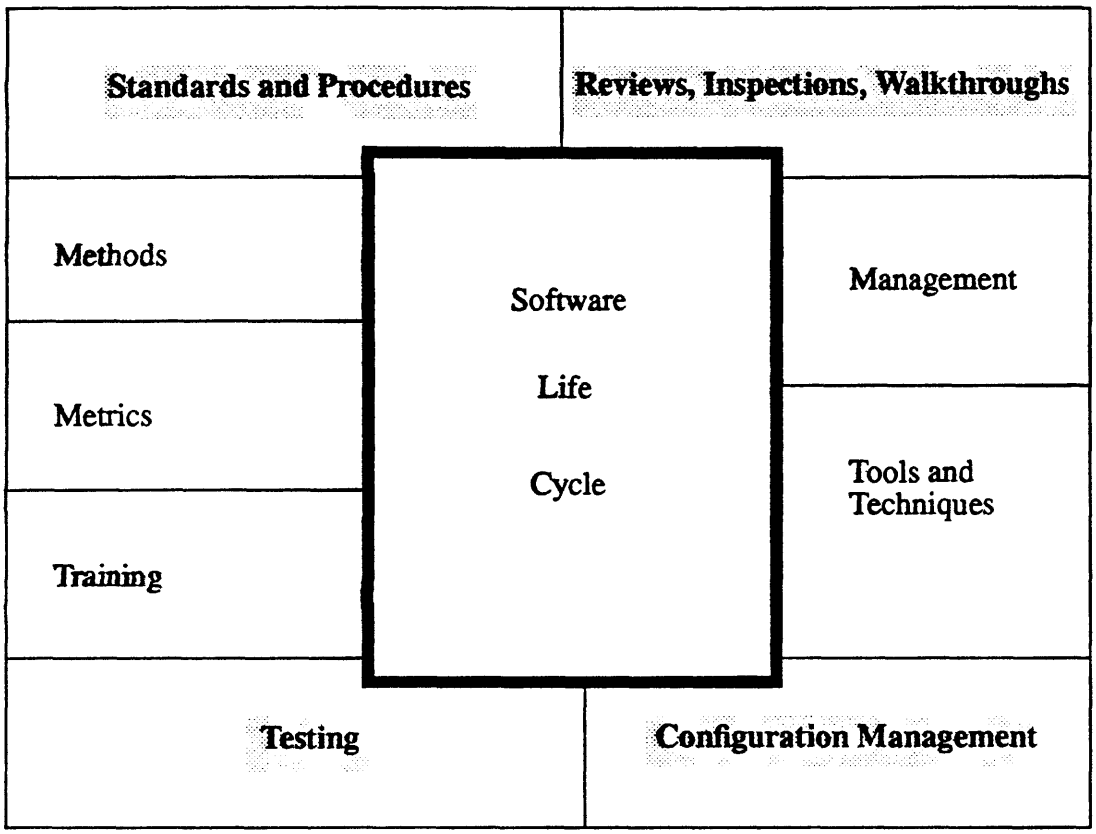

Figure 4.-Software life cycle and quality-assurance activities.
(Modified from Quality Assurance Institute, 1989b, p. 8.)

Quality Assurance and Quality Control

Quality assurance and quality control are two distinctly different functions that complement one another. Quality assurance helps to assure that the processes or methods are in place and used so that the software developers can produce quality software. Quality control is concerned with building the quality into specific software products and is the responsibility of the developers during the course of the software life cycle. An analogy to a manufacturing process is helpful for further understanding of the distinction between the two functions.

To create a software system that satisfies the stated technical or functional needs, there is a stepwise life cycle or assembly line. During the life cycle, there are specific phases to follow and software products to create or build that exhibit quality. To build in the quality, such quality-control activities as reviews, inspections, and walkthroughs that use checklists are performed at strategic points during the life cycle to make certain that (1) quality-assurance actions are being taken, and (2) software products fulfill the specified attributes of quality. If the software products do not exhibit quality, the quality-assurance and quality-control activities are analyzed and measurements (metrics) are taken for study toward improvement of the processes or methods used for building the software. The improvement of the processes or methods and building quality into the software involves concentrating on the quality control

rather than the software developers' productivity. When the software is built error-free, subsequent maintenance is less thereby increasing the developers' productivity. To build in quality, the process takes on a different perspective. As software is produced, effort must be spent on identifying sources of error within the software. Once an error is located, the effort is to permanently resolve the error-producing condition or situation. This is the time spent to improve the software development processes or methods. The result is a continuous and measurable improvement in both quality and productivity.

Often, the software developers believe they are creating the best computer system available, but what do the users say? Here lies the final and true measurement of quality. All the quality precepts, disciplines, and methods may have been adhered to, but if the users cannot easily and efficiently implement the system, it will not meet their expectations and is not quality software. If the users are unable to readily implement the system or if the system fails to meet their needs, the processes or methods by which the system was created are to blame, not the developers. Improvement of the processes or methods is a main objective of the quality-assurance function.

People Involvement

Everyone involved in, or responsible for, NWIS software development and maintenance benefits from software quality assurance. Within the software life cycle, requirements need to be communicated and understood, design needs to be traceable to the requirements, and results from several levels of testing using different strategies need to be evaluated. At the same time, quality-control checking and software change-control, auditing, and reporting must be effective. Successful quality assurance therefore relies upon people and the application of a mixture of quality assurance precepts, disciplines, methods, and proven techniques such as quality-assurance/quality-control workbenches. The involvement by key groups of WRD people is summarized in the following sections.

User Input

To meet the users' software needs and expectations, the users themselves must be involved in the software development process. This involvement first consists of stating new system needs and expectations; agreeing on the system's structure, processing functions, and data base content; and establishing hardware/software constraints with which they can live. The users must work together to provide these diverse inputs during the software development. Later, the users must approve and accept the delivered system.

User and Developer Integrated Efforts

Often a difference arises between what users say they need, expect, and receive from the software system, and what the developers can and do deliver. Figure 5 schematically shows these two potentially different or divergent views of the software system as time progresses during the development phases of the software life cycle. At any point in time or overall, the difference in views can cause an increase in costs and a software quality gap.

Without resolution of the two views, costs will continue to rise and quality objectives are not met. To close the gap between the two views to control costs and achieve quality, the users and developers must combine and integrate their technical knowledge with the development efforts. Along with this, planning, training, and a strong belief in quality assurance are required.

Manager Participation

Managers participate in software development and quality assurance by backing quality-assurance efforts; establishing both short- and long-range policies, goals, and plans; ensuring that all management

levels participate; and communicating with the users, developers, and quality-assurance personnel to overcome development delays, control costs, eliminate software errors early in development, and measure progress. This participation by managers is what establishes the software engineering environment and influences most every aspect of the development and quality of the software.

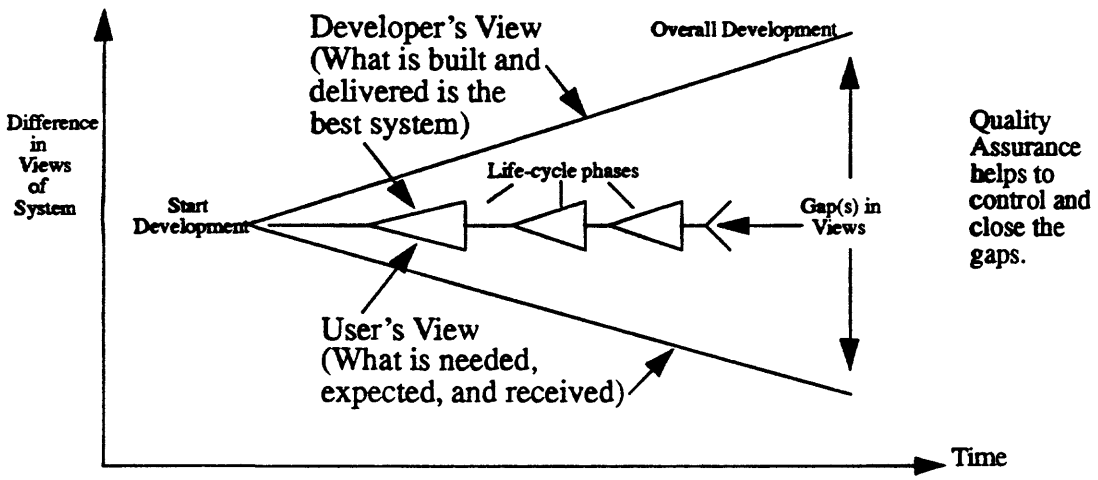

Figure 5.-Different views by users and developers during software development. (Modified from Quality Assurance Institute, 1988, p. 61.)

QUALITY-ASSURANCE ACTIVITIES

The initial NWIS Program software quality-assurance activities are based on four quality-assurance cornerstone activities (fig. 4), which work concurrently with the NWIS software life cycle to help produce quality software. The four cornerstone activities are discussed in the following sections.

Standards and Procedures

Standards are developed that state what work is to be done, what to check for, and what to measure. Procedures present the work steps taken to comply with standards. Because of their implied restrictiveness and need for enforcement, standards and procedures are more of a management challenge than a technical one (Quality Assurance Institute, 1989c, p. 1). This is because the main challenges of software standards and procedures are (1) gaining acceptance once prepared or purchased, and (2) managing and maintaining them after they are prepared or purchased. If these challenges are met and the standards and procedures are acceptable technically and followed, they protect the developers and managers and contribute to producing quality software. Full implementation of a software standards and procedures activity involves a standards hierarchy. This hierarchy consists of a series of ordered and interrelated subjects, which are summarized and defined as follows:

1. Policies—Originate with management and are prepared as three kinds of directives:
   - High-level—State software life-cycle objectives and goals, set direction, aimed at organizational mission.
   - Process (procedural) related—State what software life-cycle processes or procedures are involved or what is to be accomplished.
   - Product related—State what software products or outputs are expected.
2. Standards—State what to do, what makes the software products acceptable, and what measurements define software product quality. The standards:
   - Describe the software life-cycle policy issues.
   - Describe what the software products must consist of and what processes or methods to use to define, develop, operate, and maintain the software.
   - Describe what means are used to measure software quality and report results.
3. Procedures—Present methods of proceeding or steps taken to accomplish the software life-cycle tasks. The procedures:
   - Are followed step-by-step.
   - Are tasks that support and lead to compliance with the software standards.
4. Guidelines—Sets of instructions that describe the software procedural (item 3) methods. The guidelines:
   - Are optional, not enforced.
   - Act as guides or point the way to define, develop, and maintain the software.
   - May evolve into standards if prepared, followed, and fully accepted.

Implementing software standards and procedures based on the above hierarchy helps make one NWIS quality-assurance cornerstone activity a reality. Otherwise, nonquality software may be the result and the unanticipated software costs may grow (fig. 3). An NWIS Program Standards and Procedures Group develops, maintains, and administers software standards and procedures. The group consists of personnel from each of the NWIS Program units.

Reviews, Inspections, and Walkthroughs

Reviews, inspections, and walkthroughs within the NWIS Program are principal quality-control techniques. They consist of formal and informal evaluations and audits to alert management and the software developers of the occurrences of actual or potential problems in meeting software quality goals. These techniques are one of the most cost-effective means to detect software errors. The reviews and inspections are formal and are held at strategic points during the software life cycle, usually at the end of a life-cycle phase when specific software products are finished, and become input to another phase. Reviews usually consist of technical meetings attended by the software developers, users, and managers to evaluate software products and discuss progress to date. An inspection consists of a qualified technical person performing a detailed examination of a software product and reporting the examination results. Walkthroughs usually consist of informal meetings held as needed throughout the life cycle to discuss specific software products or system components. Walkthroughs are usually held as peer reviews one-on-one or in small groups to communicate, integrate, and assign software development or maintenance work tasks throughout the software life cycle. Because reviews, inspections, and walkthroughs are effective for technical assessment of software and are easily implemented by appropriate scheduling, they are heavily relied upon by the NWIS Program, thereby accomplishing another quality-assurance cornerstone activity.

Testing

The importance of system testing with respect to software quality assurance cannot be over-emphasized. It is a critical element of software quality assurance and quality control and represents the ultimate review of requirements analysis, design, and implementation. A general strategic approach to software testing (Pressman, 1987, p. 499) has the following characteristics:

1. Begin at a low-level (unit or module) and work toward total integration (high-level acceptance) of the system.
2. Use different testing techniques for different levels of testing at different points in time.
3. Have developers and users conduct low-level software tests and have an independent test group conduct high-level acceptance tests.
4. Perform testing and debugging. Testing and debugging are different; testing is formal, and debugging is informal. Testing is formal because it is part of the software life cycle and represents the accomplishment of major milestones in the software's development. Debugging is informal because it is performed to correct minor software errors on an as-needed basis during the life-cycle implementation (coding) and subsequent testing phases. The debugging consists primarily of walkthroughs to determine what errors to fix and how to fix them.

The NWIS Program testing strategy includes low-level tests, which verify that small coded modules or groups of modules are correct and satisfy the requirements; and high-level tests, which validate that major system components interface, integrate, and function as a whole and as planned. This strategy provides developmental task guidelines for the developers and a set of product and schedule milestones for the managers. Progress is measurable based on the milestones and problems are found before the software is released, thereby accomplishing another software quality-assurance cornerstone goal.

Configuration Management

All software products created as part of the NWIS software development effort are collectively called a software configuration. As development progresses, the number of products grows rapidly. Each life-cycle phase spawns additional products to create a hierarchy of information. Unfortunately, changes to products can occur at any time, for any reason. Software configuration management, as defined earlier, is a set of activities developed to identify, define, and control change; record and report status; and verify completeness of products throughout the software life cycle.

The process of software configuration management begins with a set of product baselines. A baseline is a first version from which all changes are made. Changes to the baseline version are requested and need to be handled. Overall handling and management of these changes leads to the following four NWIS configuration management tasks:

1. Identification—Assures meaningful and consistent naming for all software products from creation and throughout their life while part of the software configuration.
2. Change control—Provides a mechanism for the definition and control of changes to the software.
3. Configuration auditing—Checks where any particular software product is within the configuration, and assesses its characteristics of completeness and correctness to complement quality-assurance reviews and inspections.
4. Status accounting or recording and reporting—Provides answers to "what, who, when, and what else is affected by software changes" kinds of questions.

Effective configuration management by an NWIS configuration board diminishes the side effects of changes in the current development and existing software systems' products and improves the quality by

managing and controlling the software changes, thereby accomplishing the remaining quality-assurance cornerstone activity.

Other Activities

Accomplishment of all four NWIS quality-assurance cornerstone activities provides a solid foundation or base for ensuring that (1) the software satisfies the users, (2) the software satisfies established technical requirements, and (3) the software system functions as a whole. Other quality-assurance activities (fig. 4) are Methods, Metrics, Training, Management, and Tools and Techniques, which have the following objectives:

1. Improve the methods, tools, and techniques used to perform software engineering during the software life cycle.
2. Collect data (metrics) to measure quality of the software products during the software life cycle.
3. Train personnel to apply quality-assurance principles and methods during the software life cycle.
4. Manage risks of potential software failure and associated problem resolution during the software life cycle.

Implementation of all software quality-assurance activities will eventually form the basis for total quality assurance in the NWIS Program.

WORKBENCH APPROACH

The implementation of software quality assurance and quality control in the NWIS Program uses a generic workbench approach. The workbench operates by describing the flow of activity necessary to create or build software products that exhibit quality and also to evaluate their quality. Figure 6 shows the generic workbench for software quality assurance/quality control. Products are delivered to the workbench as input from other workbenches, work is performed, and products are delivered from the workbench as output to yet another workbench (software life-cycle phase or task). The workbench accomplishes the quality control for the associated product by incorporating the quality-assurance cornerstone activities into its four major parts (highlighted boxes in fig. 6). Each of the four major workbench parts are discussed later.

If the input product to the workbench is unsatisfactory, it is reworked and reevaluated until satisfactory for use in the performance of work. During work, new products may be created or built based on the input products, or the products may be improved/changed to become revised products. Once the work is completed, the new or revised products are evaluated for readiness and fitness to be delivered as output from the workbench. If the product is not ready or fit, it is reworked and reevaluated until ready for delivery. At times, an output evaluation may or may not suffice as an input evaluation to another workbench, depending upon the number, type, and status of the products involved. For example: A product such as a document or group of coded modules is required in the next workbench but is not needed in the one after that. In this case, the current output evaluation may suffice for the next workbench's input evaluation. If the product is not required in the next workbench but is needed in the one after that, the current output evaluation of the product may be informal or nonexistent. As time passes and the product is needed in the subsequent workbench, the product may need to be evaluated or reevaluated for status and readiness as input to that particular workbench.

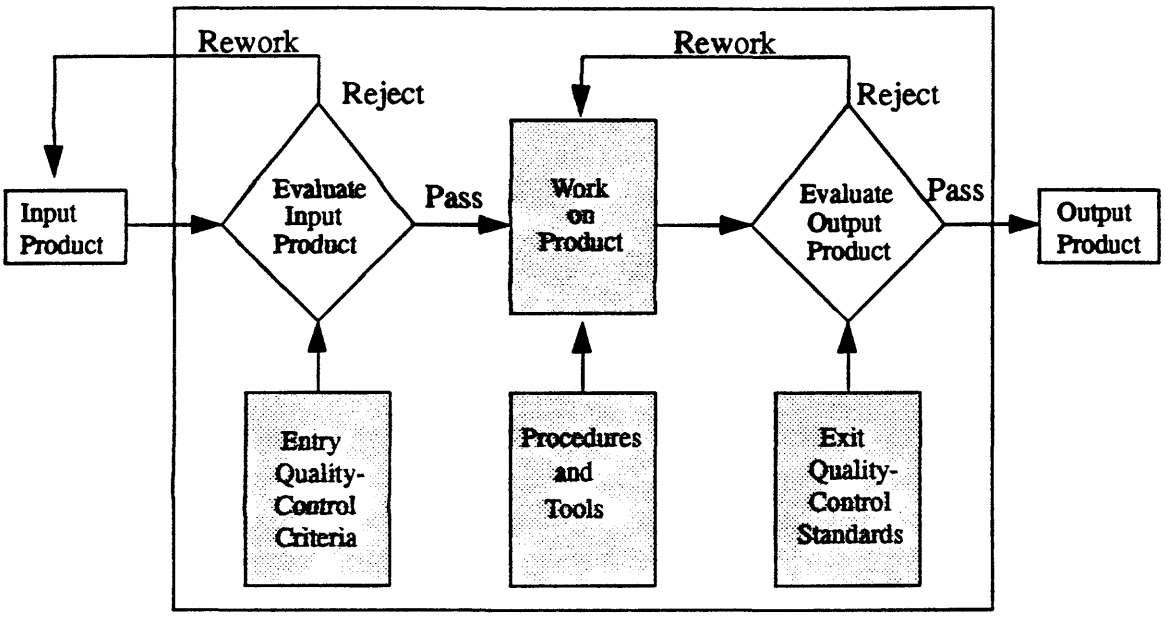

Figure 6.-Generic workbench for software quality assurance/quality control. (Modified from Quality Assurance Institute Seminars—Workshops, 1988, p. 90.)

Reasons for a Workbench

The workbench is a proven technique (Perry, 1986, p. 17) and provides a planned and systematic approach for implementing quality assurance and quality control. The generic workbench forms a template for any software life-cycle phase workbench or other software engineering task. Each workbench is viewed as having standard parts and appropriate sets of software engineering tools that are understandable and useful. Successful workbench passage usually ensures that software products meet needs and are ready or fit for subsequent use. In each workbench, the processes and methods, as defined by the procedures to perform the work, can be studied and evaluated for potential improvement (that is, accomplishing quality assurance).

Generic Workbench Parts

Each generic workbench consists of four parts: Entry Criteria, Work, Procedures and Tools, and Exit Standards. These parts form the template for establishing any workbench. The parts reflect and embody the quality-assurance cornerstone activities to evaluate and create or build quality software products. Each workbench part is discussed in the following sections.

Entry Criteria

Entry criteria are the quality-control criteria needed to ensure that the input products contain all of the necessary attributes that make them satisfactory and acceptable for processing by the workbench. These attributes for quality need to be defined for each workbench's input products. For example, document attributes may consist of such quality factors as correctness, reliability, usability, and maintainability. For data, the same factors as for documents, but add integrity. For code, the same factors as for documents and

data, but add efficiency and testability. The entry evaluation is accomplished mainly by an inspection of the products and/or study of review or test results from another workbench.

Work

Work is performed to create desired output products from input products. The work to be performed is defined by the software engineering procedures for that particular life-cycle phase or task. In the performance of the work, computer-aided software engineering (CASE) and other automated tools need to be used if available. These tools help to achieve standardization of the procedures and improve productivity. This standardization limits the variation and inconsistency in the ways software is defined, developed, and maintained. The major work tasks are (1) preparing system planning, definition, and design documents, source and executable code, and test data, (2) testing the system, and (3) approving and distributing the system.

Procedures and Tools

The procedures provide the how-to-do-it processes and methods used to perform the work and they enable personnel to create or build products that will meet predefined software standards. Procedures explain the series of actions taken to accomplish what is wanted, what needs to be done to obtain quality, and steps that will be taken to check the work. As mentioned, CASE and other automated tools are used to perform work. An example set of CASE tools to use in requirements analysis and design are ones that provide interactive capability to construct structured analysis and data flow diagrams and provide for definition of data dictionaries. In implementation/coding it is a compiler, while other tools may be an automated flowcharting package or a language syntax checker and testing package.

Exit Standards

Exit standards are the quality-control criteria needed to evaluate and measure the output products against the standards for those products. The quality-control criteria are usually contained in checklists of questions based on the standards or ratings that pertain to the individual product standards. Questions are answered or products are rated during quality-assurance reviews or inspections. Also during reviews or inspections, testing results, if applicable, are evaluated as a critical element of the quality control before exiting the workbench. The quality control is the responsibility of the personnel producing the product. If the product fails to meet exit criteria (is defective or varies from the standard), it is reworked. If the product is newly created or revised, meets the standards, and is approved by managers and users, the product leaves the workbench and is baselined as part of the software configuration.

REQUIREMENTS-ANALYSIS PHASE WORKBENCH

An example workbench for the requirements-analysis (RA) phase of the NWIS software life cycle illustrates the practicality and effectiveness of the workbench approach for accomplishing quality assurance and quality control. The principal input and output products of the RA phase are documents. Figure 7, which is based on figure 6, shows the workbench for the requirements-analysis phase of the software life-cycle. The numbers shown in the figure indicate the order in which workbench items are discussed.

1. Input Documents—Consist of initial NWIS-II concepts and planning documents, initial drafts of eight water discipline-oriented users' requirements documents, and a single document outline for the functional and data base requirements specification based on an integration of the eight users' requirements documents. All of this information is produced in the software life-cycle initiation phase (fig. 2).

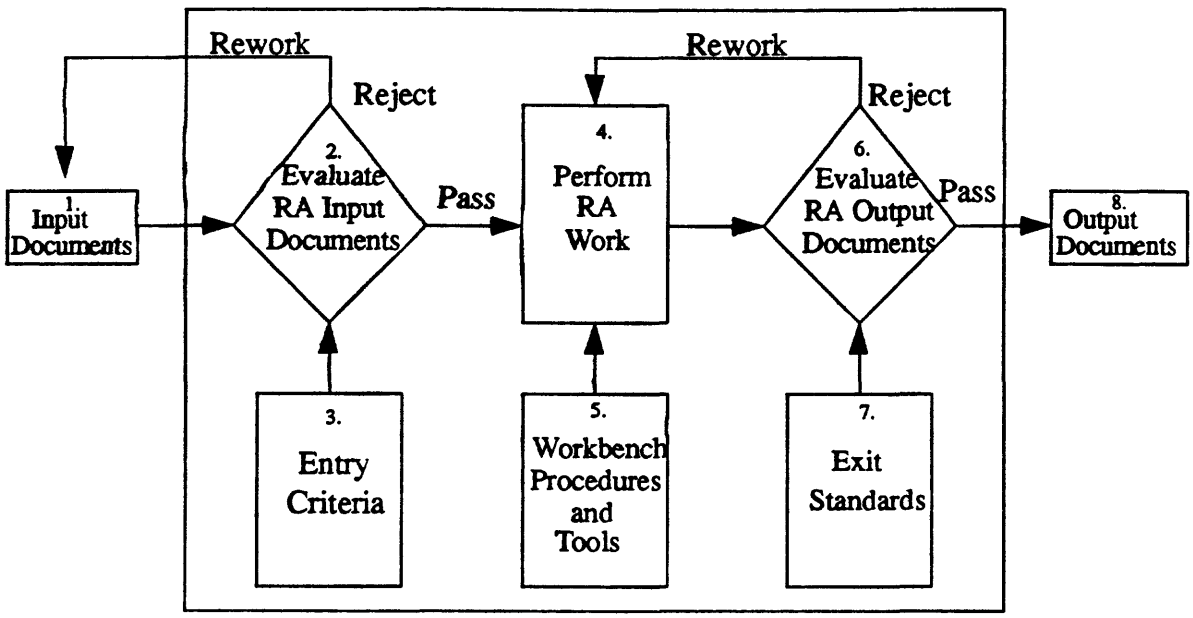

Figure 7.-Workbench for the requirements-analysis phase of the software life cycle.

2. Evaluate RA input documents—Use entry criteria (discussed next) to decide if RA input documents are acceptable for processing by the workbench. If necessary, have a review or inspection.
3. Entry criteria—The NWIS-II established quality-control attributes used to determine if the input documents are acceptable and exhibit quality. The attributes are correctness, reliability, usability, and maintainability. The documents need to specifically address the requested content, functionality, performance, computational, interface, accuracy, security, and scheduling issues. The attributes may be incorporated into a checklist for use in the evaluation.
4. Perform RA work—Use input documents to help establish the procedures and select the CASE tools required to perform the requirements analysis. Proceed to prepare standards for output products (see no. 8) and prepare procedures and guidelines in support of standards. Use the input documents to prepare the requirements specification for the new system and transfer of data from existing systems. Also, begin to prepare low- to high-level test scenarios and cases, and begin specified documentation, such as test plans and user's guide. The requirements specification is the principal product of the RA phase. Hold walkthroughs as needed to coordinate, integrate, and assign RA work tasks.
5. Workbench procedures and tools—Follow the established procedures (previous step) and use available tools to perform the work. These tools may consist of computer-aided software engineering (CASE) tools that (1) provide capabilities to formalize and verify requirements, (2) accomplish initial configuration management, and (3) produce initial supporting graphical and textual visualizations of system components and data flows and structures.
6. Evaluate RA output documents—Requires a decision based on use of a formal quality-control review or inspection that uses checklists to evaluate the requirements specification and other