# Software System Engineering: A Tutorial

**Richard H. Thayer, Ph.D.**

## 1. INTRODUCTION

This article integrates the definitions and processes from the IEEE Computer Society Software Engineering Standards Series into the software system engineering process. Software systems have become larger and more complex than ever before. Hardware has advanced to a state of high efficiency and the need to reduce software size is no longer the primary design goal. These changes have led to the creation of extremely large and complex software systems. For example, Microsoft Word, which once fit on a 360-Kbyte diskette, now requires a 600-Mbyte CD.

The vast majority of these large software systems do not meet their projected schedule and estimated costs, nor do they completely fulfill the acquirers' expectations.* This phenomenon has come to be regarded as the "software crisis."[1,2] In response to this "crisis," different engineering practices have been introduced in developing software products. Simply tracking project status during the development phase is insufficient. Monitoring resources used, meetings schedules, or milestones accomplished will not provide sufficient feedback regarding project health. Management of the technical processes and products is essential. System engineering provides the additional tools needed to perform the technical management of the system under development.

This article describes the application of system-engineering principles to the development of a computer software system. The activities, tasks, and procedures that make up these principles are known as *software system engineering* (SwSE). The purpose of this article is to identify SwSE processes and tools, briefly describe them, and reflect on their contribution to the software development process.

## 2. WHAT IS A SYSTEM?

Before describing system engineering, the term "system" must be defined. A *system* is a collection of elements *related* in a way that allows the accomplishment of a *common objective.* The key words are *related* and *common objective.* Unrelated terms and elements without common objectives are not part of a system under study but belong to other systems.

A *man-made system* is a collection of hardware, software, people, facilities, procedures, and other factors organized to accomplish a common objective. A *software system* is, therefore, a man-made system consisting of a collection of programs and documents that work together, creating a set of requirements with the specified software.

### 2.1. What Is System Engineering?

*System engineering* (SE) is the practical application of scientific, engineering, and management skills necessary to transform a user's need into a system configuration description that most effectively and efficiently satisfies the need. SE produces documents, not artifacts.

SE is the overall technical management of a system-development project. IEEE Standard 1220-1998 discusses the environment and processes that compose system engineering:

> System engineering is the management function that controls the total system development effort for the purpose of achieving an optimum balance of all system elements. It is a process that transforms an operational need into a description of system parameters and integrates those parameters to optimize overall system effectiveness.[3]

*This article uses the definitions from IEEE/EIA Standard 12207.2-1998, where *acquirer* is used for *customer* and *supplier* is used for *developer* or *contractor.*

The SE process is a generic problem-solving process that provides the mechanism for identifying and evolving the product and process definitions of a system. This process applies throughout the system life cycle to all activities associated with product development, test/verification, manufacturing, training, operation, support, distribution, disposal, and human systems engineering.[4]

System engineering establishes a technical development management plan for the project. The developmental processes are identified and associated with the life-cycle model selected for the project. System engineering also defines the expected environment for the identified processes, interfaces, products, and risk management throughout project development.

System engineering provides the baseline for all software and hardware development. Requirements specific to the system must eventually be partitioned into those that apply to the software subsystem and those that apply to the hardware subsystem.

This article recognizes the difference between software system engineering (SwSE) and software engineering (SwE), just as it recognizes the difference between SE and all types of hardware engineering. SwSE supports the premise that the quality of a software product depends on the quality of the process used to create it. *The products of both SE and SwSE are documents.* This is in contrast to hardware (component) engineering, in which the products are the devices or components being produced, and SwE, in which the products are computer programs (software), and the documentation necessary to use, operate, and maintain the software.

### 2.2. What Are the Functions of System Engineering?

SE involves:

- *Problem definition,* which determines the needs and constraints through analysis of requirements and interfacing with the acquirer
- *Solution analysis,* which determines the set of possible ways to satisfy requirements and constraints, studies and analyzes possible solutions, and selects the optimum one
- *Process planning,* which determines the tasks to be done, project size and effort required to develop the product, precedence among tasks, and potential risks to the project
- *Process control,* which determines the methods for controlling the project and the process, measures progress, reviews intermediate products, and takes corrective action when necessary
- *Product evaluation,* which determines the quality and quantity of the delivered product through evaluation planning, testing, demonstration, analysis, examination, and inspection

## 3. WHAT IS SOFTWARE SYSTEM ENGINEERING?

SwSE, like SE, is both a technical and management process. The *technical process* of SwSE is the analytical effort necessary to transform an operational need into:

- A description of a software system
- A software design of the proper size, configuration, and quality
- Documentation requirements and design specifications
- The procedures necessary to verify, test, and accept the finished product
- The documentation necessary to use, operate, and maintain the system

SwSE is not a job description; it is a process. Many people and organizations can perform SwSE: system engineers, managers, software engineers, programmers, and, not to be ignored, acquirers and users. Many practitioners consider SwSE to be a special case of SE, others consider it part of SwE, and still others consider SwE part of SwSE. This article maintains the separation between SwSE and SwE.

SE and SwSE are often overlooked in software development projects. Systems that are entirely software and/or run on commercial, off-the-shelf computers are often considered software projects, not system projects; hence, no effort is expended to develop a SE approach. Ignoring the systems aspect of the project often results in software that will not run on selected hardware or will not integrate with hardware and other software systems. Ignoring systems aspects contributes to long-running software crises.

The term *software system engineering* is attributed to Winston W. Royce, a pioneer leader of SwE in the early 1980s.[5]

### 3.1. The Role of Software in Systems

Software is the dominant technology in many technical systems. Software often provides the cohesiveness and control of data that enables the system to solve problems. Software also provides the flexibility to work around hardware or other problems, particularly those discovered late in the development cycle.

Figure 1 shows how software ties system elements together.[6] For this reason, software is frequently the most complex part of any system, and therefore the greatest technical challenge. In the example in Figure 1, software lets radar, people, radios, airplanes, communications systems, and other equipment work together to provide an air traffic control system that safely guides an aircraft from its departure airport to its destination through all kinds of weather and over a variety of terrain. It should also be noted that management information systems (MIS) are software systems that tie organizational entities together.

### 3.2. Why Does Software Require System Engineering?

Most systems are a mixture of software and hardware. However, the solution space must be defined before functionality can be assigned to hardware or software. Solution space defines system boundaries and its interfaces with outside systems.

SE provides the mechanism for defining the product solution at the highest level. The systems-engineering approach provides an opportunity to specify the solution for the acquirer prior to separating the solution into hardware and software subsystems. This is similar to the software engineering practice of avoiding the specification of constraints for as long as possible in the development process. The further into the development process a project proceeds before a constraint is defined, the more flexible the implemented solution will be.

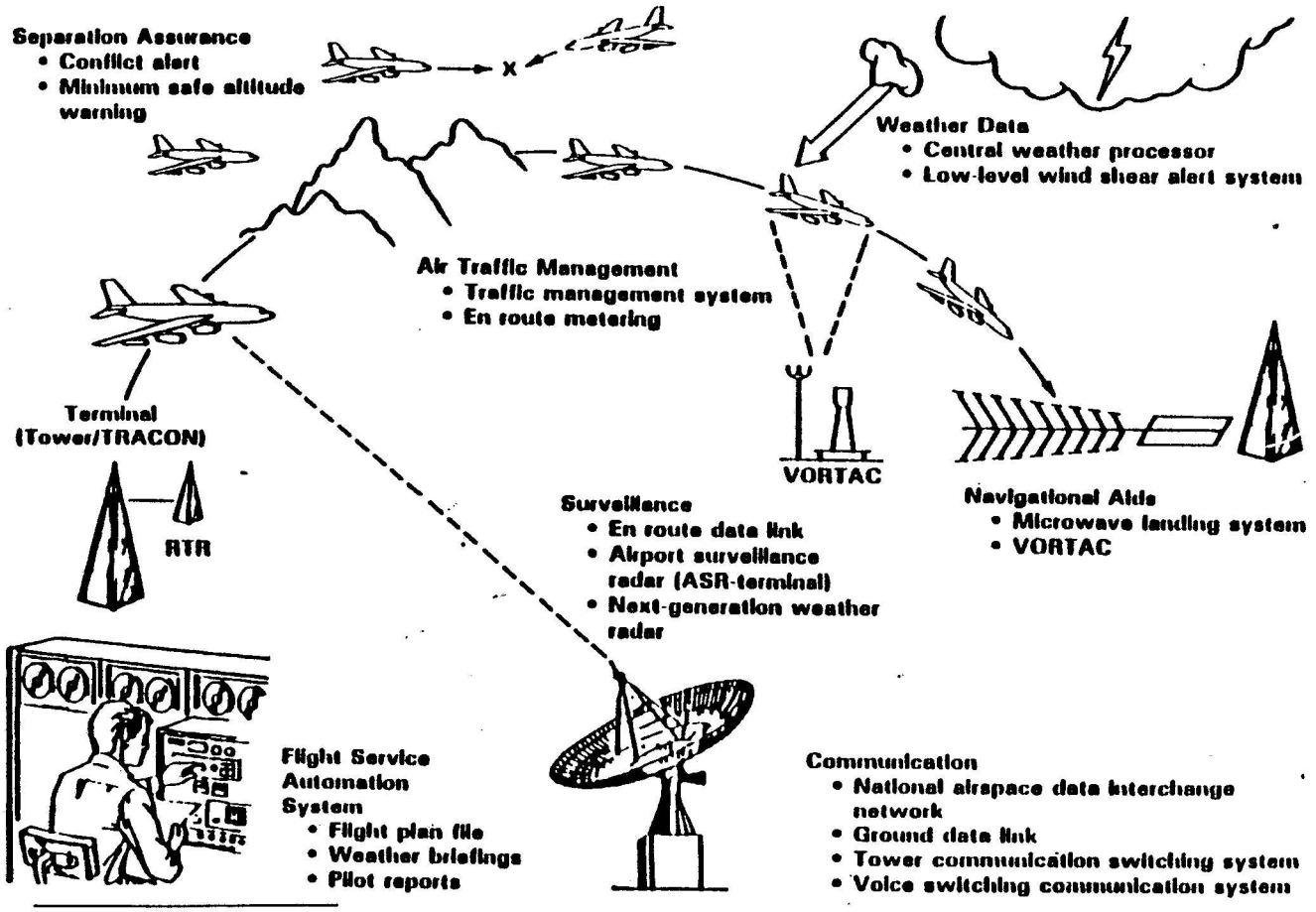


**Figure 1.** Software ties the system together. (Used with permission of Logicon, Incorporated)

### 3.3. Why Is Software System Engineering Necessary?

SwSE is responsible for the overall technical management of the system and the verification of the final system products.

New systems are being developed today. Many of these are highly dependent on properly operating software systems. Thus, software is larger and more complex now than at any time in history. This trend is caused by:

- The availability of cheap computer hardware, which motivates system requirements
- More system solutions being provided by software rather than hardware
- Increased software complexity due to increased system complexity
- Acquirers who seek reliable and usable software systems
- Acquirers who want software capability and flexibility

As a result, software development costs are rising, and software takes much longer to produce. These extremely large and complex systems require technical system management and SE oversight. Without this SE approach, the following problems often result:

- Complex software systems become unmanageable.
- Costs are overrun and deadlines are missed.
- Unacceptable risks are sometimes taken, leading to greater risk exposure.
- Erroneous decisions are made early in the life cycle and prove very costly in the end.
- Independently developed subsystems and components do not integrate properly.
- Parts and components are not built or requirements are not met.
- The delivered system fails to work properly.
- Parts of the system must be reworked after delivery (maintenance).

SwSE is necessary to build the "new order" of computer-dependent systems now being sought by governments and industries.

### 3.4. What Is Software Engineering?

In contrast to SwSE, SwE is:

- The practical application of computer science, management, and other sciences to the analysis, design, construction, and maintenance of software and its associated documentation
- An engineering science that applies the concepts of analysis, design, coding, testing, documentation, and management to the successful completion of large, custom-built computer programs under time and budget constraints

The systematic application of methods, tools, and techniques to achieve a stated requirement or objective for an effective and efficient software system

Figure 2 illustrates the overlapping relationships between SE, SwSE, and SwE. Traditional SE does initial analysis and design and final system integration and testing. During the initial stage of software development, SwSE is responsible for software requirements analysis and architectural design. SwSE is also responsible for the final testing of the software system. SwE is responsible for what SE calls "component engineering," that is, software design, coding, and unit testing.

### 3.5. What Is Project Management?

The project management (PM) process involves assessing the software system's risks and costs, establishing a master schedule, integrating the various engineering specialties and design groups, maintaining configuration control, and continuously auditing the effort to ensure that cost and schedule are met and technical requirements objectives are satisfied.[7]

Figure 3 illustrates the relationships between PM, SwSE, and SwE. PM has overall management responsibility for the project and the authority to commit resources. SwSE has the responsibility for determining the technical approach, making technical decisions, interfacing with the technical acquirer, and approving and accepting the final software product. SwE is responsible for developing the software design, coding it, and developing software configuration items (subsystems).

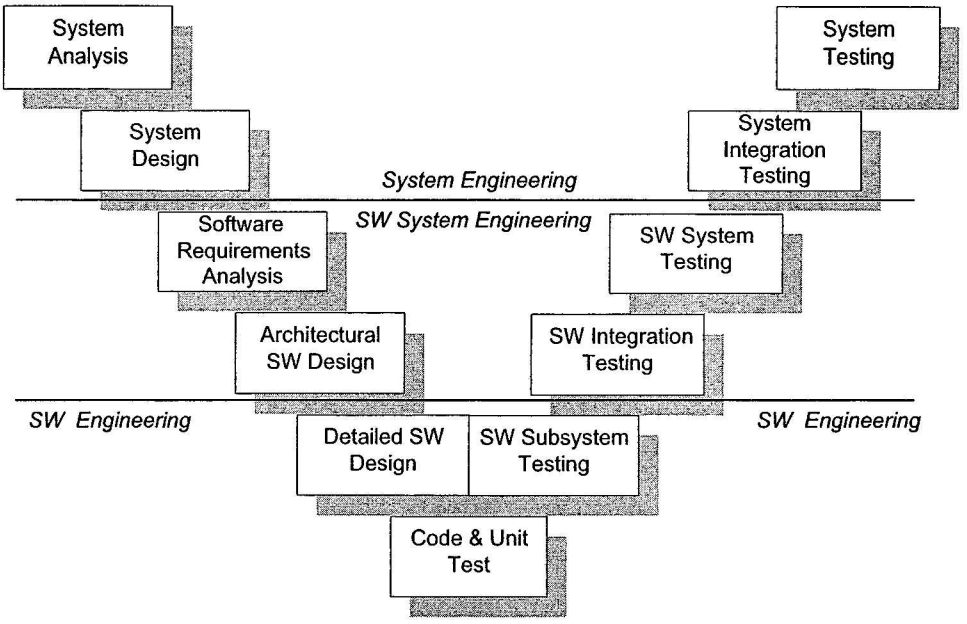


**Figure 2.** Engineering relationships.

### 3.6. What Are the Functions of Software System Engineering?

SwSE involves the following activities (note that the terms in parentheses are the typical names given to these functions in SE):

- *Requirements analysis (problem definition)* determines the needs and constraints of the software system by analyzing the system requirements that have been allocated to software.
- *Software design (solution analysis)* determines the set of possible ways to satisfy the software requirements and constraints, studies and analyzes the possible solutions, and selects the optimum one.
- *Process planning* determines the tasks to be done, the project size, the effort necessary to develop the product, the precedence between tasks, and the potential risks to the project.
- *Process control* determines the methods for controlling the project and the process, measures progress, reviews intermediate products, and takes corrective action when necessary.

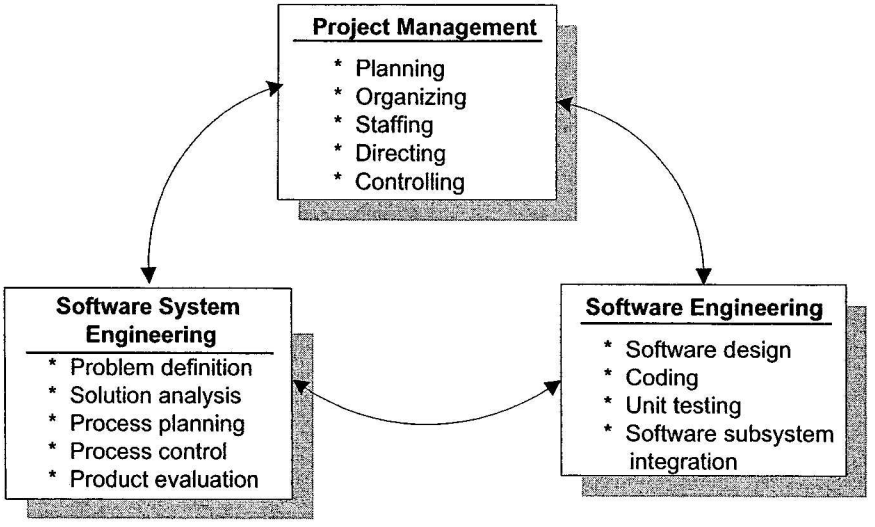


**Figure 3.** Management relationships.

- *Verification, validation, and testing (product evaluation)* evaluates the final product and documentation through testing, demonstrations, analysis, and inspections.

SwSE is the overall technical management of a system development project.

## 4. WHAT IS SOFTWARE REQUIREMENTS ANALYSIS (PROBLEM DEFINITION)?

Software requirements are:

1. A software capability needed by a user to solve a problem or achieve an objective
2. A system capability that must be met or possessed by a software system or software system component to satisfy a contract, standard, specification, or other formally imposed document[8]

Software requirements analysis establish the[9]

- *Functional requirement,* which specifies functions that a system or system component must be able to perform
- *Performance requirement,* which specifies performance characteristics (such as speed, accuracy, or frequency ) that a system or system component must possess
- *External interface requirement,* which specifies hardware, software, or database elements with which a system or system component must interface, or sets forth constraints on formats, timing, or other factors caused by such an interface
- *Design constraint,* which affects or constrains the design of a software system or software system component (for example, language requirements, physical hardware requirements, software development standards, and software quality assurance standards)
- *Quality attribute,* which specifies the degree to which software possesses attributes that affect quality, such as correctness, reliability, maintainability, and portability

The first step in any software development activity is to investigate, determine, and document the system requirements in a system requirements specification (SRS) and/or software requirements in a software requirements specification.

Software requirements analysis is *initiated* when:

- Acquirer and user system requirements have been properly identified
- A top-level system design (sometimes called system architecture) is complete
- A set of software subsystems is identified
- All user requirements have been properly allocated (assigned) to one or more of these subsystems

Requirements analysis is *complete* when:

- A concept of operations (ConOps) document is complete (if not already furnished by the user)
- All (or as many as possible) of the software system requirements are identified
- An SRS that identifies "what" the system must do (without assuming "how" it will be accomplished) is written
- The SRS is verified
- A requirements tracing is complete
- A preliminary user manual is written
- A test compliance matrix is finished
- A preliminary test specification is finished
- The software specification review (SSR) is complete
- A requirements (sometimes called allocated) baseline is established

### 4.1. Concept of Analysis Process and Concept of Operations (ConOps) Document

The *concept of analysis process* identifies the overall system characteristics from an operational (user's) viewpoint. Concept analysis helps users clarify their operational needs, thereby easing the problem of communication among users, acquirers, and

suppliers.[10] A *concept of operations (ConOps) document,* which records the results of concept analysis, provides a bridge from user needs to the system and software development process. Ideally, concept analysis and development of the ConOps document are the first steps in the development process.

One of the most useful aspects of the ConOps document is the scenario included in a fully developed ConOps. A scenario describes, from the users' point of view, a series of events that could occur simultaneously or sequentially. Scenarios are extremely valuable in helping the software system's prospective user to describe the requirements of the new software system. More recently, the scenario has been replaced with the *use case,*[11] a new *requirements elicitation* technique.

### 4.2. Software Requirements Analysis and Specifications

Software requirements analysis is the process of studying the acquirer's system needs and operational environment to arrive at a definition of system or software requirements. Over the past 25 years, many tools and techniques that can be used to analyze and represent the requirements have been developed. A few of the more prominent examples are listed below. A detailed analysis of these tools is beyond the scope of this article.

1. *Functional flow block diagrams* (FFBDs) "rough out" the major functions and their interfaces.
2. *Structured analysis and dataflow diagrams* determine and represent system functions.
3. *Object-oriented analysis and object diagrams* determine and represent system objects and classes.
4. *Use cases* enable the functional requirements of a business process to be written from the users' point of view.
5. *N-squared ($N^2$) charts* help establish and represent system and subsystem interfaces.
6. Timeline analyses depict the concurrence, overlap, and sequential relationship of time-critical functions and related tasks.

### 4.3. Requirements Tracing

*Software requirements tracing* is defined as the step-by-step flowdown of a requirement from its inception as a system or ConOps requirement, to its allocation in an SRS, to its implementation in a design and code, and, finally, to its test criteria in a test specification.

### 4.4. Preliminary User's Manual

The *user's manual* (UM) is prepared by the software system supplier and used to provide the user and/or user personnel with the necessary instructions regarding how to use and operate the software system.[12] The manual's content and format are specifically designed to meet the needs of the intended users.

The preliminary user's manual is developed early in the software development process to allow verification of software requirements. The user's manual is based on the SRS because the requirements specifications describe the finished system. If the SRS contains errors of omission, incompleteness, or incorrectness, users will likely discover these errors during their review of the user's manual.[13]

### 4.5. Preliminary Test Specification

A *preliminary test specification* details the test approach for a software feature or combination of software features and identifies the associated tests.[14]

The preliminary test documents are developed early in the software development process to allow for verification of software requirements. The test documents are based on the requirements specifications because these specifications describe the finished system. The test suppliers will likely discover any errors of omission, incompleteness, or incorrectness in the software requirements specification early in the life cycle. No requirement is complete until its testability has been demonstrated in a test-planning document.

### 4.6. Verify Compliance Matrix

A *compliance matrix* is a representation method for determining which methods will be used to verify (test) each software requirement. A requirement that cannot be verified needs to be dropped or modified so that it can be verified.

### 4.7. Software Specification Review (SSR)

The SSR allows acquirers, users, and management to analyze products and specifications in order to assess progress.[15]

An SSR is a *joint acquirer–developer review* (also known as a milestone review) conducted to finalize software requirements so the software supplier can initiate a top-level software design. The SSR is conducted when software requirements have been sufficiently defined to evaluate the supplier's responsiveness to and interpretation of the system- or segments-level technical requirements. A successful SSR is predicated upon the acquirer and/or supplier's determination that the SRS and interface specifications form a satisfactory basis for proceeding into the top-level design phase.[15]

### 4.8. Software Requirements Verification

*Software requirements verification* involves verifying software requirements specifications against the system requirements and architectural design.[17] The verification function looks for bad software requirements (incorrect, incomplete, ambiguous, or untestable) and noncompliance with the system functional and nonfunctional requirements.

Some of the tools and procedures used by validation and verification (V&V) in this phase are requirements traceability analysis, requirements evaluation, requirements interface analysis, and test planning.[18] Additional tools include control flow analysis, dataflow analysis, algorithm analysis, simulation analysis, in-process audits, and requirements walk-throughs.

### 4.9. Requirements Baseline

The *requirements baseline* is the formal description of a system that has been accepted and agreed on by all concerned parties.[19]

The *software requirements baseline* is the initial software configuration identification established at the end of the software requirements generation phase. This baseline is established by an approved software requirements and interface specification and placed under formal configuration control by joint agreement between the supplier and the user. Department of Defense suppliers would call this the *software allocated baseline.*

## 5. WHAT IS SOFTWARE DESIGN (SOLUTION ANALYSIS)?

*Software design* is the process of selecting and documenting the most effective and efficient system elements that together will implement the software system requirements.[20>] Software design is traditionally partitioned into two components: architectural design and detailed design.

*Architectural design* is equivalent to system design. During architectural design, the system structure is selected and the software requirements are allocated to components of the structure. Architectural design (also called *top-level design* or *preliminary design*) typically includes definition and structuring of computer program components and data, definition of the interfaces, and preparation of timing and sizing estimates. *Detailed design* is equivalent to what SE calls "component engineering," and is considered part of the software engineering phase. The components in this case are independent software modules and artifacts. This article allocates architectural design to SwSE and detailed design to SwE.

Software architectural design establishes the

- Top-level system design
- General system structure (architecture)
- System components (subsystems) that populate the architecture
- General description of each subsystem
- Source of each subsystem (build, buy, reuse)
- Internal interfaces between subsystems
- External interfaces with outside systems
- Allocation of the functional requirements to the subsystems
- Allocation of nonfunctional requirements to each subsystem
- Verification that the architectural subsystem implements the requirements specifications

The design represents a logical satisfaction of the requirements by a specific approach.

The software design is *initiated* when:

- The software requirements have been properly identified and documented
- The draft user's manual and draft test documents have been developed

Software system design is *completed* when:

- The "how" question is answered (conceptual design)
- The architectural description is complete
- An architectural design review (sometimes called preliminary design review) is complete and the stakeholders have accepted the design description
- The preliminary operator's and maintenance manuals are written
- The design tracing is complete
- The architectural design is reviewed
- The architectural design is verified
- The product baseline is established

### 5.1. Architectural Design

Architectural design is the part of SwSE that can illustrate the final product for users and customers. Similar to a building architect's drawing, the software architectural drawings represent what is to be built to the user, software engineer, and programmer. Instructions for building the component and necessary quality attributes are also included in the architectural drawings. Architectural design tools are similar to those used in the requirements analysis process:

1. *Functional flow block diagrams* (FFBDs) "rough out" major system functions and their interfaces.
2. *Structured charts* are treelike representations of an architectural structure. Charts represent the organization (usually hierarchical) of program components (modules), the hierarchy of control, and the flow of data and control.
3. *Dataflow diagrams* determine and represent system functions.
4. *Object-oriented design and object diagrams* determine and represent system objects and classes.
5. *Use cases* represent the functional requirements of a software system.

### 5.2. Design Traceability

The requirements tracing system documents the forward and backward trace between the software requirements (the source) and the software design (the results). The results are documented in a requirements tracing report, which is presented in the architectural design review.

### 5.3. Preliminary Operator and Maintenance Manuals

The preliminary operator and maintenance manuals are initial versions of two of the three deliverable documents used to support a software system.

An *operator manual* enables system operators (in contrast to users) to support the users and operate the system. On many systems, particularly modern desktop computers, the operator, and the user are the same, resulting in combined operator and user manuals.

The *maintenance manual* is the legacy that the software suppliers leave future software engineers to aid them in maintaining the system. A maintenance manual (sometimes called a *software maintenance document*) is a SwE project deliverable document that describes the software system to software engineers and programmers who are responsible for maintaining the software in the operation and maintenance phases.

### 5.4. Architectural Design Review

The *architectural design description* and other software design products are reviewed by an appropriate review team composed of the suppliers, acquirers, and potential users of the system. This major milestone review is also called the *preliminary design review* (PDR). If the architectural design and other phase products are found acceptable, the suppliers can proceed to the detailed design phase.

*Special note.* The term *architectural design* has gradually replaced the early term *preliminary design* as being more appropriate for the activities involved. The acronym for the "appropriate review" (architectural design review, ADR) did not catch on, so we still use the term "preliminary design review" for the architectural design phase review.

### 5.5. Architectural Design Verification

The software design specifications are verified against the software requirements and the top-level system design. The verification function looks for incomplete or incorrect designs, inefficient and unmaintainable designs, poor user interfaces, and poor documentation.

The minimum V&V tasks are design traceability analysis, design evaluation, interface analysis, and updating of the V&V test plan and test design specifications for component testing, integration testing, system testing, and acceptance testing.[21] The V&V activity then reports the discrepancies found between these levels of life-cycle documents and other major problems.

### 5.6. Producing the Software Architectural Design Baseline

The *architectural design baseline* is the top-level product baseline. It is established by an approved software architectural design and interface specification and placed under formal configuration control by joint agreement between the supplier and user. Many projects do not formally establish this baseline.

## 6. WHAT IS PROCESS PLANNING?

*Planning* involves specifying the *goals* and *objectives* of a project, and the *strategies, policies, plans,* and *procedures* for achieving them. Planning is deciding in advance what to do, how to do it, when to do it, and who should do it.

Planning a SwE project consists of the SwSE management activities that lead to selecting, among alternatives, future courses of project action and a program for completing those actions.

The software process plan is initiated once the decision to develop a software system has been made. Planning is a continuous effort. Software system planning is completed when the project is finished.

### 6.1. Planning the Engineering Workload

Project planning is divided into two separate components: planning that is accomplished by PM, and planning that is done by SwSE.

There is an erroneous assumption that only PM accomplishes project planning. In reality, PM is only part of the project-planning effort. Most project planning is done by the SwSE function (which is not to say that project managers might not do both). The planning partitioning illustrated in Table 1 might exist in a software system project.

### 6.2. Determine Tasks to be Done

A *work-breakdown structure* (WBS) is a method of representing, hierarchically, the parts of a process or product. It can represent a process (for example, requirements analysis, design, coding, or testing), a product (for example, an applications program, a utility program, or system software), or both. The WBS is a major system-engineering tool that can be used by both SE and PM.

The WBS is used by SwSE to partition the software project into elementary tasks to be done. Figure 4 shows a generic example of a WBS.

### 6.3. Determine Project Effort and Schedule

SE determines the effort (normally in labor hours) and precedence relationships between tasks. PM will then assemble the tasks into a precedence activity network and determine a schedule and, in some cases, the critical path (see Figure 5). A critical path is an activity network highlighting the longest path through the network.

### 6.4. Determine Technical and Managerial Risks

*Risk* is a chance of something undesirable occurring, such as a schedule overrun, the project exceeding its budget, or the project delivering an unsuitable product. Risk is characterized by:

- Uncertainty (e.g., a probability between 0 and 1)
- An associated loss (life, property, reputation, and so on)
- The fact that it is at least partially unknown
- Change over time

A *problem* is a risk that has materialized.

## 7. PROCESS CONTROL

*Control* is the collection of management activities used to ensure that the project goes according to plan. Performance and results are measured against plans, deviations are noted, and corrective actions are taken to ensure conformance of plans and actual results.

**Table 1.** Process planning versus project planning

| Software system engineering determines: | Project management determines: |
|---|---|
| • The tasks to be done | • The skills necessary to do the task |
| • The order of precedence between tasks | • The schedule for completing the project |
| • The size of the effort (in staff time) | • The cost of the effort |
| • The technical approach to solving the problem | • The managerial approach to monitoring the project status |
| • The analysis and design tools to use | • The planning tools to use |
| • The technical risks | • The management risks |
| • The process model to be used | • The process model to be used |
| • Updates to the plans when requirements or development environments change | • Updates to the plans when managerial conditions and environments change |

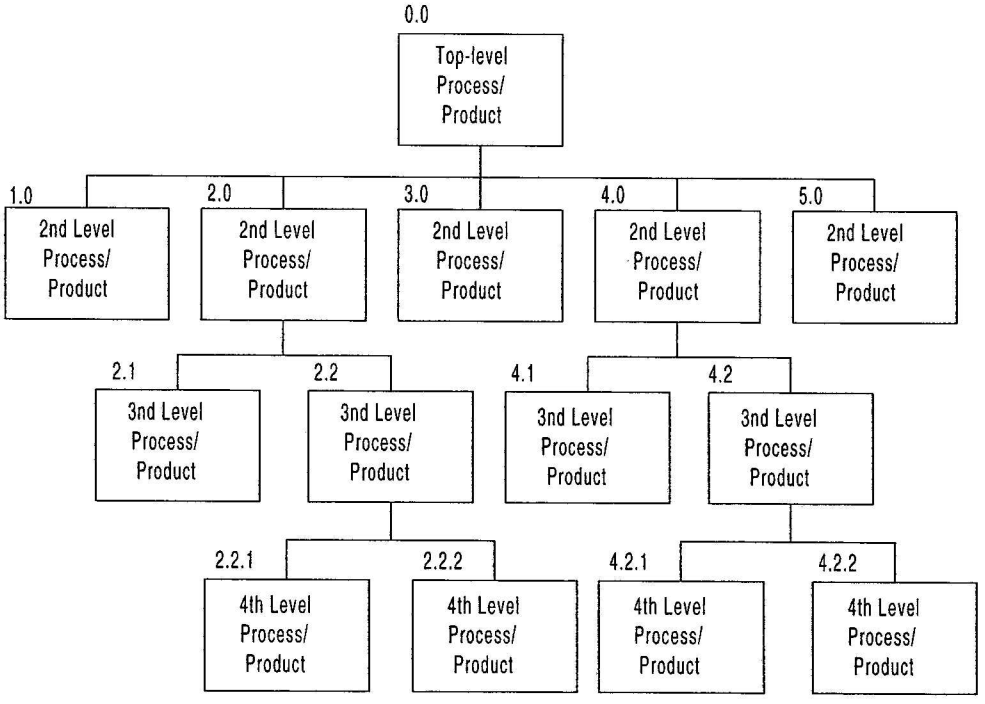


**Figure 4.** A generic work-breakdown structure (WBS).

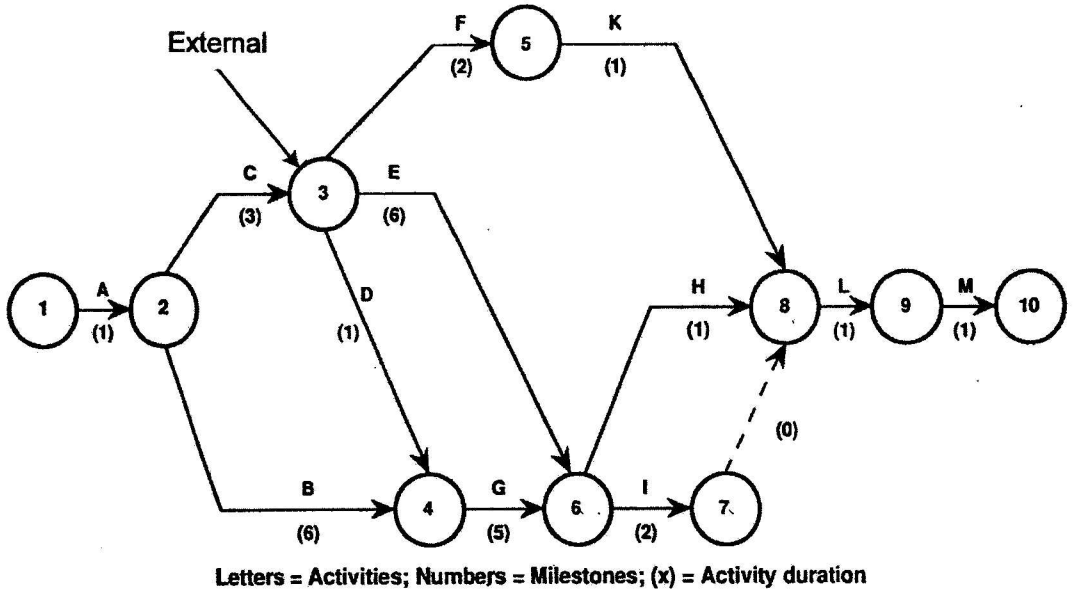


**Figure 5.** An activity network.

*Process control* is a feedback system that provides information on how well the project is going. Process control asks these questions:

- Are there potential problems that will cause delays in meeting the requirement within the budget and schedule?
- Have any of the risks turned to problems?
- Is the design approach still doable?

Control must lead to corrective action—either to return the actual status to plan, to change the plan, or to terminate the project.

### 7.1. Control the Engineering Workload

Project control is also sectioned into two separate components: control that is accomplished by PM, and control that is done by SwSE. The control sections in Table 2 might exist in a software system project.

### 7.2. Develop Standards of Performance

Goals must be set. These goals will be achieved when tasks are correctly accomplished. A *standard* is a documented set of criteria that specifies and determines the adequacy of an action or object. A *SwE standard* is a set of procedures that define the process for developing a software product and/or specify the quality of a software product.

**Table 2.** Process control versus project control

| Software system engineering: | Project management: |
|---|---|
| • Determines the requirements to be met<br>• Selects technical standards to be followed (for example, IEEE Standard 830)<br>• Establishes technical metrics to control progress (for example, requirements growth, errors reported, rework)<br>• Uses peer reviews, in-process reviews, SQA, V&V, and audits to determine adherence to requirements and design<br>• Reengineers the software requirements when necessary | • Determines the project plan to be followed<br>• Selects managerial standards to be followed (for example, IEEE Standard 1058)<br>• Establishes management metrics to control progress (such as cost growth, schedule slippage, staffing shortage)<br>• Uses joint acquirer-developer (milestone) reviews and SCM to determine adherence to cost, schedule, and progress<br>• Restructures the project plan when necessary |

### 7.3. Establish Monitoring and Reporting Systems

*Monitoring and reporting systems* must be specified in order to determine project status. These systems determine necessary data: who will receive the status report, when they will receive it, and what they will do with it to control the project. SE and project managers need feedback on project progress and product quality to ensure that everything is going according to plan. The type, frequency, originator, and recipient of project reports must be specified. Status-reporting tools that provide progress visibility, and not just resources used or time passed, must be implemented. A few of the more prominent monitoring and reporting systems are listed below.

1. *Software quality assurance* (SQA) is "a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements."[22] SQA includes the development process, management methods (requirements and design), standards, configuration management methods, review procedures, documentation standards, V&V, and testing specifications and procedures. SQA is one of the major control techniques available to the project manager.
2. *Software configuration management* (SCM) is a method of controlling and reporting software status. SCM is the discipline of identifying system configuration at discrete points in time. This is done to systematically control changes to the configuration and maintain the integrity and traceability throughout the system life cycle.[23]

### 7.4. Measure Results

SE and PM are responsible for measuring project results both during and at the end of the project. For instance, actual phase deliverables should be measured against planned phase deliverables. The measured results can be management (process) results and/or technical (product) results. For example, the project schedule status is a process result, and the degree to which the design specifications correctly interpret the requirements specifications is a product result. A few of the more prominent monitoring and reporting systems are listed below.

1. *Binary tracking and work product specifications.* A work product specification describes the work to be accomplished in completing a function, activity, or task. A *work package* specifies *work objectives:* staffing, expected duration, resources, results, and any other special considerations for the work. Work packages are normally small tasks that can be assigned to two to three individuals to be completed in two to three weeks. A series of work packages comprises a software project.

   Binary tracking classifies a work package as either finished or not finished (that is, it assigns a numeric "1" or "0"). Binary tracking of work packages is a reasonably accurate means of tracking the completion of a software project. For example, a project that has 880 work packages with 440 finished is probably 50% complete.
2. *Walk-throughs and inspections. Walk-throughs* and *inspections* are reviews of a software product (design specifications, code, test procedures, and so on) conducted by peers of the group being reviewed. Walk-throughs are critiques of a software product by the producer's peers solely to finding errors. The inspection system is another peer review developed by Michael Fagan of IBM in 1976.[24] Inspections are typically more structured than walk-throughs.
3. *Independent auditing.* The *software project audit* is an independent review to determine a software project's compliance with software requirements, specifications, baselines, standards, policies, and SQA plans.

### 7.5. Take Corrective Action if Process Fails to Meet Plan or Standard

Corrective action means bringing requirements and plans into conformance with actual project status. This might involve requiring a larger budget, more people, or more checkout time on the development computer. It also might require reducing the standards (and indirectly the quality) by reducing the number of walk-throughs or by reviewing only the critical software modules instead of all of them.

An example of changing the requirements involves delivering software that does not completely meet all the functional requirements that were laid out in the original SRS.

## 8. VERIFICATION, VALIDATION, AND TESTING (VV&T) (PRODUCT EVALUATION)

The purpose of the *verification, validation, and testing* (VV&T) effort is to determine that the engineering process is correct and the products are in compliance with their requirements.[25]

The following critical definitions apply to VV&T:

- *Verification.* The process of determining whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. Verification answers the question, "Am I building the product right?"
- *Validation.* The process of determining the correctness of the final program or software produced from a development project with respect to the user's needs and requirements. Validation answers the question, "Am I building the right product?"
- *Testing.* The execution of a program or partial program with known inputs and outputs that are both predicted and observed for the purpose of finding errors. Testing is frequently considered part of validation.

V&V is a continuous process of monitoring the activities of SE, SwSE, SwE, and PM to determine that the technical and managerial plans, specifications, standards, and procedures are being followed. V&V also evaluates the interim and final products of the SwE project. Interim products are requirements specifications, design descriptions, test plans, review results, and so on. Final products include software, user manuals, and training manuals.

Any individual (or functions) within a software development project can perform V&V. SwSE uses V&V techniques and tools to evaluate requirements specifications, design descriptions, and other interim products of the SwSE process. SwSE uses testing to determine if the final product meets the project requirements specifications.

The last step of a software development activity is to validate and test the final software product against the SRS and to validate and test the final system product against the system requirements specifications.

Software VV&T is *initiated* when:

- The software is complete
- Unit (component) tests are complete

Software VV&T is *completed* when:

- Software integration testing is complete
- Software system testing is complete

### 8.1. Integration Testing

*Software integration testing* involves integrating the components of the software system and testing the integrated system to determine if the system works as required. Integration testing can be either incremental or nonincremental. *Incremental testing* involves testing a small part of the system and then incrementing the system configuration by adding one component at a time and testing after each increment. *Nonincremental testing* involves testing all system components at once. Hardware engineers call this the "smoke test," that is, "let's test it all at once, and see where the smoke rises." Since software failures don't smoke, it is often difficult to tell where a system has failed when all components are tested at one time. Incremental testing has proven to be a more successful approach.

There are two separate strategies to incremental testing: top-down testing and bottom-up testing (see Figure 6). *Top-down testing* integrates the system under test progressively from top to bottom, using simulations of low-level components (called *stubs*) during testing to complete the system. This is different from *bottom-up testing,* which increments the system under test progressively from bottom to top, using software drivers to simulate top-level components during testing.

Each of these strategies has a number of advantages and disadvantages, several of which are listed in Table 3.

At first glance, bottom-up testing appears to have the most advantages and the least disadvantages. However, advantage 3 under top-down testing in Table 3 and disadvantage 2 under bottom-up testing provide significant management advantages through increased user satisfaction and progress visibility. Therefore, top-down testing is often the preferred method.

### 8.2. System Testing

In this process, the integrated software system is tested against the software requirements. Tests are conducted in conformance with formal test documents and the test compliance matrix. As described earlier, *verification* determines whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. *Validation* ensures that what is built corresponds to what was actually required. It is concerned with the completeness, consistency, and correctness of the requirements.

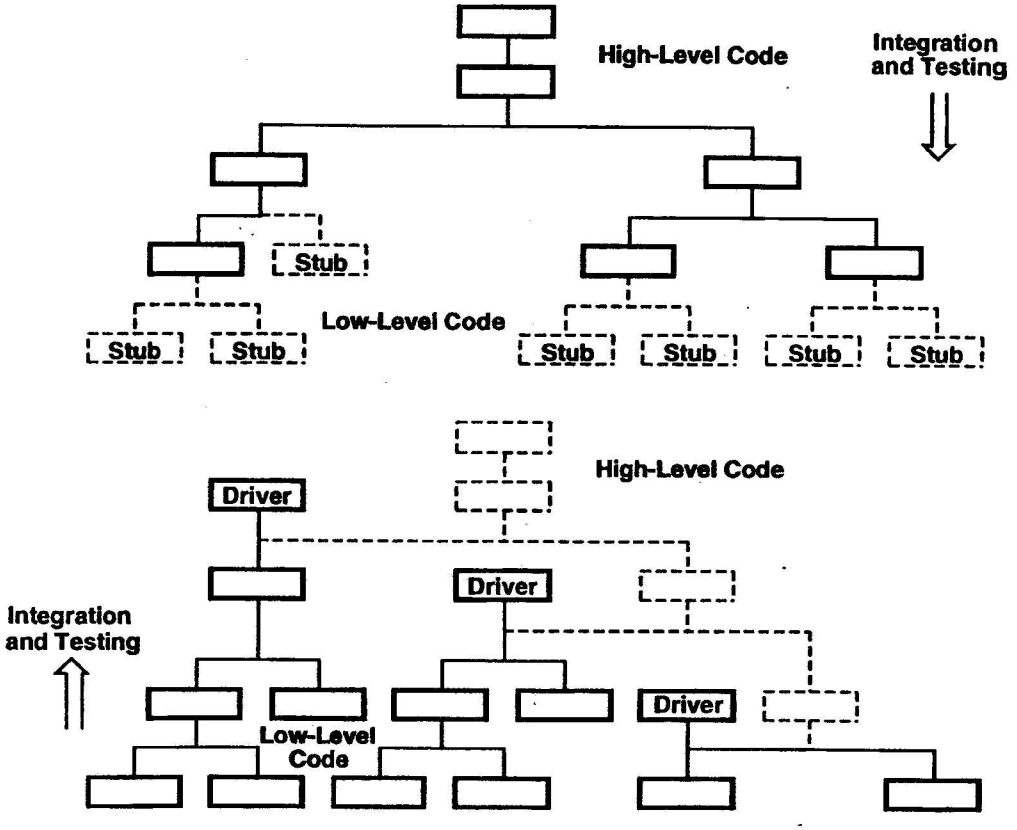


**Figure 6.** Top-down testing versus bottom-up testing.

During the software system testing phase, the software system is verified against the build to baseline to determine if the final program properly implements the design. At roughly the same time, the software system is tested and validated against the original system and software requirements specifications to see if the product was built correctly.

## 9. SUMMARY AND CONCLUSIONS

Conducting SwE without conducting SwSE puts a project in jeopardy of being incomplete or having components that do not work together and/or exceed the project's scheduled budget.

SE and SwSE are primarily disciplines used in the front end of the system life cycle for technical planning and at the latter

**Table 3.** Comparison of top-down versus bottom-up testing[26]

| Top-down testing | |
|---|---|
| Advantages: | Disadvantages: |
| 1. It is more productive if major flaws occur toward the top of the program. | 1. Stub modules must be produced. |
| 2. Representation of test cases is easier. | 2. Stub modules are often complicated. |
| 3. Earlier skeletal programs allow demonstration (and improve morale). | 3. The representation of test cases in stubs can be difficult. |
| | 4. Observation of test output is more difficult. |
| **Bottom-up testing** | |
| 1. It is more productive if major flaws occur toward the bottom of the program. | 1. Driver modules must be produced. |
| 2. Test conditions are easier to create. | 2. The program as an entity does not exist until the last module is added. |
| 3. Observation of test results is easier. | |

part of the life cycle to verify whether the plans have been met. A review of the emphasis in this article will show that much of the work of planning and SwSE is done during the top-level requirements analysis and top-level design phases. The other major activity of SwSE is the final validation and testing of the completed system.

SE principles, activities, tasks, and procedures can be applied to software development. This article has summarized, in broad steps, what is necessary to implement SwSE on either a hardware-software system (that is, primarily software) or on an almost entirely software-based system.

SwSE is not cheap, but it is cost effective.

## REFERENCES


1. W. W. Gibbs, "Software's Chronic Crisis," *Scientific American,* Sept. 1994, pp. 86–95.
2. W. W. Royce, "Current Problems," in *Aerospace Software Engineering: A Collection of Concepts,* C. Anderson and M. Dorfman, eds., American Institute of Aeronautics, Inc., Washington, DC, 1991.
3. *System Engineering Management Course Syllabus,* The Defense System Management College, Ft. Belvoir, VA, 1989
4. IEEE Standard 1220-1998, *IEEE Management of the System Engineering Process,* IEEE, New York, 1998.
5. W. W. Royce, "Software Systems Engineering," A seminar presented during the Management of Software Acquisition Course, Defense Systems Management College, Fort Belvoir, VA, 1981–1988.
6. R. Fujii, speaker, "IEEE Seminar in Software Verification and Validation," Logicon, Inc., two-day public seminar presented by the IEEE Standards Board, Munich (13–14 July 1989).
7. IEEE Standard 1058-1998, *IEEE Standard for Software Project Management Plans,* IEEE, New York, 1998.
8. IEEE Standard 610.12-1990, *IEEE Glossary of Software Engineering Terminology,* IEEE, New York, 1990.
9. IEEE Standard 830-1998, *IEEE Recommended Practice for Software Requirements Specifications,* IEEE, New York, 1998.
10. IEEE Standard 1362-1998, *IEEE Guide for Information Technology—System Design—Concept of Operations Document,* IEEE, New York, 1998.
11. G. Schneider and J. P. Winters, "Applying Use Cases," in *Software Engineering, Part 1, The Development Process,* R. H. Thayer and M. Dorfman, eds., IEEE Computer Society Press, Los Alamitos, CA, 2002.
12. IEEE Standard 1063-1987 (reaffirmed 1993), *IEEE Standard for Software User Documentation,* IEEE, New York, 1993.
13. N. R. Howes, "On Using the Users' Manual as the Requirements Specification II," in *Tutorial: System and Software Engineering Requirements,* R. H. Thayer and M. Dorfman, eds., IEEE Computer Society Press, Los Alamitos, CA, 1989.
14. IEEE Standard 829-1998, *IEEE Standard for Software Test Documentation,* IEEE, New York, 1998.
15. IEEE Standard 1028-1997, *IEEE Standard for Software Reviews,* IEEE, New York, 1997.
16. MIL-STD 1521B (USAF), *Technical Reviews and Audits for Systems, Equipment, and Computer Programs* (proposed revision), Joint Policy Coordination Group on Computer Resource Management, 1985.
17. IEEE Standard 1012-1998, *IEEE Standard for Software Verification and Validation,* IEEE, New York, 1998.
18. IEEE Standard 1012-1998, *IEEE Standard for Software Verification and Validation,* IEEE, New York, 1998.
19. IEEE Standard 828-1998, *IEEE Standard for Software Configuration Management Plans,* IEEE, New York, 1998.
20. IEEE Standard 1016-1998, *IEEE Recommended Practice for Software Design Descriptions,* IEEE, New York, 1998.
21. IEEE Standard 1012-1998, *IEEE Standard for Software Verification and Validation,* IEEE, New York, 1998.
22. IEEE Standard 730-1998, *IEEE Standard for Software Quality Assurance Plans,* IEEE, NY, 1998.
23. E. H. Bersoff, "Elements of Software Configuration Management," *IEEE Transactions on Software Engineering, SE-10, 1,* Jan. 1984, pp. 79–87. Reprinted in *Tutorial: Software Engineering Project Management,* R. H. Thayer, ed., IEEE Computer Society Press, Los Alamitos, CA, 1988.
24. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal, 15,* 3, 1976.
25. IEEE Standard 1012-1998, *IEEE Standard for Software Verification and Validation,* IEEE, New York, 1998.
26. G. J. Myers, *The Art of Software Testing, Wiley, 1979.*