The management of software engineering Part II: Software engineering program

by D. O'Neill

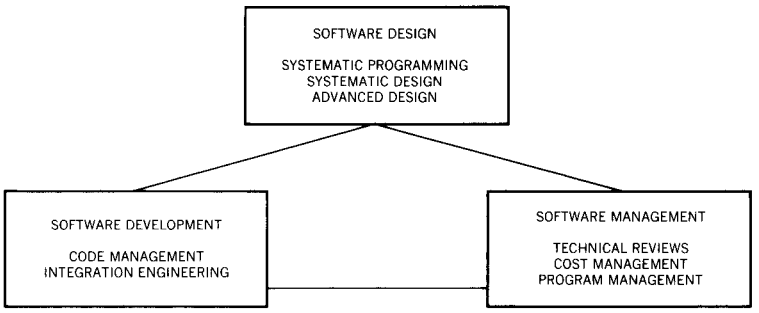
The breadth of applications in industry today stresses software development and has resulted in a diversity of design technologies, computer products, programming languages, support software tools, and documentation requirements. Moving from one application area to another can require major adjustments by both technical people and management. These time- and energy-consuming adjustments introduce more diversity and further complicate an already complex process.

It is well known that software costs associated with computer system developments have been increasing and are becoming a critical cost element in these developments.[17,18] Budget data indicate that software costs may become ninety percent of system development costs by 1985. Yet the development of reliable software on schedule within cost has been and remains a significant management challenge.[19,20] At the same time, hardware manufacturing costs are being reduced by orders of magnitude. These trends are introducing new levels of complexity into software developments as demands for system performance and reliability are requiring greater precision in software design and development. Potential solutions to these problems have been surveyed by the IBM Federal Systems Division, with particular attention to recent developments in the academic and professional communities.[7,13,21-23]

The result of our research into these new developments has been to make these developments teachable and practical in the discipline of *software engineering*. Software engineering has been defined as the systematic design and development of software products and the management of the software process. The software engineering discipline combines design topics resulting from university influences with the software development and management expertise of industry. Both perspectives are necessary to support a software engineering program that blends technology advances with practical innovations.

The software engineering program of the IBM Federal Systems Division demonstrates a commitment to the improving of the software development process beyond the software technology innovations of structured programming, top-down development,

Figure 1 Software engineering practices organization



and chief programmer teams. Following the advances in hardware technology, this program is designed to teach the practice of higher levels of precision in software design and development.[1,4,24,25] The program addresses current trends in the business environment that demand software product quality and reduced software costs.

The software engineering program combines development of comprehensive practices, education in such practices, and development of necessary support tools. The primary thrust of the program is the preparation of uniform *software engineering practices* that apply modern design, practical development, and proven management methods. Software engineering practices are introduced through formal education to provide a broad base of professional programmers who are able to produce software systems utilizing these disciplines. Uniform tools are provided to support the uniform practices. Process assessment has also been introduced to ensure that this program is being followed and to measure its effectiveness.

A comprehensive collection of technical and managerial practices is emerging from the combination of successful experience and university research. These represent the best of the current understanding of the software engineering process and a proven way of designing, developing, and managing software. Each practice defines specific work products that serve as visible intermediate steps in the process. The application of the design techniques produces modular designs and structured programs that are reliable and efficient as well as adaptable to change. Software development utilizes high-level languages and programming support library hierarchies to manage code produced in a natural sequence of phased increments. Management methodology provides plans and controls that ensure cost and schedule visibility of the process, as well as technical performance measurements of the emerging product. The relationships among software engineering design, development, and management practices are shown in Figure 1 and are discussed later in Parts III to V.

The software design practices introduce advanced software technology including systematic programming, systematic design, and advanced design. Systematic programming practices involve logical and program expression for recording designs, program design through stepwise refinement, and program verification using formal proof of correctness, as well as less formal methods. Systematic design practices cover data design with data types and structures, and modular design using state machines and modules. Advanced design practices cover concurrent design including synchronization and real-time considerations, and software system specification using state machine methodology.

Software development practices include code management and integration engineering. Code management ensures that software is uniform with respect to programming language usage, coding standards, and conventions. This also includes software system-building procedures and covers computer product support software and the software development environment. Integration engineering introduces procedures for software integration, incremental software development, and interface specification management. It also covers simulation and performance measurement software.

Software management practices include technical reviews, cost management, and program management. The technical reviews are product based for the completion of each work component of the software development life cycle. Cost management includes the practices for process and design-to-cost methodology, and program management is designed to improve the visibility of the software process through more effective plans and controls.

Software development life cycle

A set of *activities* has been defined that describes the software process from system definition through operational support. The activities that make up the software process are further defined in terms of work components that identify the tasks to be performed, as shown in Table 1. These activities portray software in the enlarged perspective of the full life cycle as viewed by the customer, and provide the basis for effective management. The activities may overlap in time, but each must be scheduled for completion prior to subsequent dependent activities. Many developments call for performance in all activities of the life cycle, whereas others may involve only certain ones. Work components have specific completion criteria in the form of *work products* that are subject to technical review.

Table 1 Software life-cycle activities

Activity	*Work components*
System definition	Software requirements definition Software system description Software development planning
Software design	Functional design Program design Test design Software tools Design evaluation
Software development	Module development Development testing
Software system test	Software system test procedures Software integration and test
System and acceptance test	System test support Acceptance test support
Operational support	System operation support Training Site deployment support

In many projects, support activities that are required to produce a work product may be collected together from an organizational or cost accounting viewpoint into a general support function. Typically, this includes project management, software configuration management, software quality assurance, software cost engineering, administrative centers, technical publication,[22] financial management services, and data management. Software engineering practices apply across the full life cycle. The correspondence between software engineering practices and the life-cycle activities is shown in Table 2.

System definition includes definition and analysis of the software system requirements, establishment of a software system description, and initiation of the software development planning necessary to proceed with further development of the software system. The software system requirements are a record of the complete system capability, including both the software and the environment in which the system is to operate. Because requirements documentation is expressed in natural language and may lack precision, a description that is produced for the software system only is prepared in a precise, detailed, succinct, and sufficient manner using prescribed methods of expression. The software system description must be traceable to the requirements document and must maintain semantic correspondence with that document. The initial software development plan is prepared on the basis of the software system description and includes cost management planning, schedules and external dependencies, and resources of both people and machines.

Table 2 Practices and activities relationships

Life cycle activities	*Design*			*Development*		*Management*		
	Advanced design	*Systematic design*	*Systematic programming*	*Code management*	*Integration engineering*	*Technical reviews*	*Cost management*	*Program management*
System definition	•					•	•	•
Software design		•	•	•	•	•	•	•
Software development			•	•	•	•	•	•
Software system test				•	•	•	•	•
System and acceptance test				•		•	•	•
Operational support				•			•	•

Software design includes conversion of the software system description into a design, design evaluation, preparation of test designs, and production of software tools. Functional designs are composed of module designs produced according to systematic design practices, and program designs are composed of structured programs produced from the module designs according to systematic programming practices. The preparation of test designs is performed using integration engineering practices.

The *software development* activity includes module development and development testing. Module development is the final elaboration of design details according to systematic programming practices and the preparation of source language statements that can be translated into executing code. These statements comply with their program and module designs and are produced in accordance with code management practices. Module testing includes test procedure executions to ensure that an implemented module complies with the specification of the software system and is conducted according to integration engineering practices.

Software system testing includes the preparation of testing procedures followed by software integration and testing as specified in the integration engineering practices. This is to ensure that the implemented software system complies with specification of the software system and the code management practices.

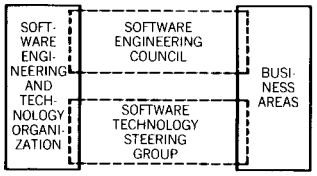
System and acceptance testing ensures that the software system complies with all project-deliverable objectives. This testing also verifies that all deliverable items exist and all reviews have been successfully completed. Code management practices apply during this activity.

Operational support includes system operation support, site deployment support, and training; code management practices apply here as well. The product of a project is typically delivered to a customer who operates it with minimal post-delivery assistance. Customer procedures for change control, system evaluation, and so forth apply during the operations stage of the life cycle.

Software engineering program implementation

The main thrust of the software engineering program is to improve product quality and reduce cost by implementing consistent practices. A successful operation of the program also requires education, tools, and measurements. Education in modern techniques gives personnel an understanding and appreciation of the methods defined in the practices. Tools that support these methods ensure their effective utilization and rapid adoption. The application of these methods and realization of their benefits require continuous assessment and feedback of results.

Figure 2 Software engineering program organization

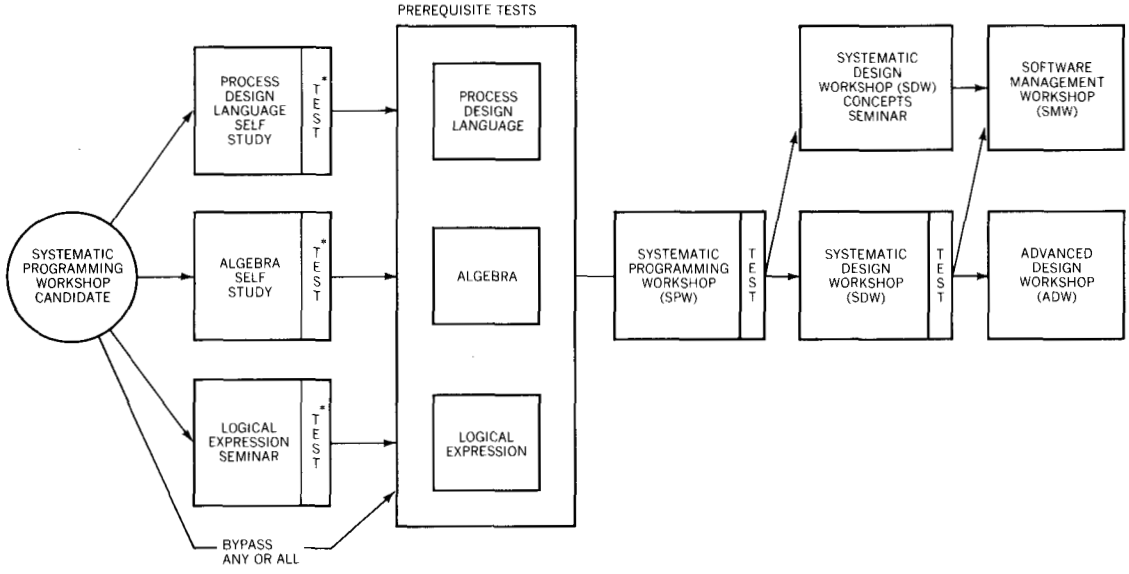


The software engineering program requires continuous communication between the business areas responsible for contract performance and the Software Engineering and Advanced Technology group, which is the divisional organization responsible for technology advances. Two special communication channels, shown in Figure 2, have been established for this purpose.

A Software Engineering Council provides policy guidance in software operations and is responsible for setting the direction of the software engineering program. The Council, which meets quarterly, is composed of senior software executives. A Software Technology Steering Group, which sponsors and reviews the software engineering practices, formulates software technology strategy. The latter group also generates the software investment programs. The steering group meets monthly and is composed of representatives from the business areas and the Software Engineering and Technology Group. They are responsible for formulating the software engineering practices to reflect a combination of the best current business usage and practical new ideas from the computer sciences. Participation of the business areas ensures their commitment to the program while at the same time providing the technologists with practical insights and project constraints.

A software engineering curriculum has been designed around seminars and workshops, as illustrated in Figure 3. Three prerequisite self-study courses and four instructor-taught courses are available in this curriculum, which was begun in October 1977 and is to be completed in 1981. Their common objective is to

Figure 3 Software engineering education curriculum



significantly improve the predictability of the software process and the quality of the resulting product. The underlying technical objective is to increase visibility and intellectual control over a developing software product by the process of stepwise refinement using standard conceptual models and a limited number of basic control and data structures. The process calls for the successive replacement of abstract designs with increasingly more detailed designs that are known to be equivalent. The underlying management objective is to provide complementary development strategies, feedback mechanisms, and control techniques. Admission to these courses is arranged through Software Engineering Program Coordinators in each business area. Table 3 summarizes information on the expected audience, prerequisites, and duration of each course.

The *three prerequisite self-studies* prepare students to read the professional literature and to communicate in the workshops using the language of mathematics and a software design language. Prerequisite tests are administered by business area coordinators in algebra, Process Design Language (PDL), and logical expression (basic concepts and the notations of set theory and symbolic logic).

A *Systematic Programming Workshop* (*SPW*) advocates a particular discipline for the design of sequential programs modeled on mathematical functions. Designs are expanded from abstract statements of a program's intended function, using PDL, which is a programming-like design language. At each step, a design state-

Table 3 Software education summary

Course	*Audience*	*Prerequisites*	*Duration*
Self-study	All programmers and analysts	Programming experience	Logical expression 10–15 hours Algebra 1/2–1 hour Process Design Language 6–10 hours
Systematic Programming Workshop (SPW)	All programmers and analysts and others designated by management	Algebra, Process Design Language Logical expression	8 1/2 days
Systematic Design Workshop (SDW)	Key programmers and analysts and others designated by management	Satisfactory completion of Systematic Programming Workshop	5 days
Advanced Design Workshop (ADW)	Software designers and architects designated by management	Satisfactory completion of Systematic Design Workshop	3 days
Software Management Workshop (SMW)	All software managers and selected technical personnel	Satisfactory completion of SDW or SDW concepts seminar	5 days

ment is replaced by a simple function-equivalent program whose components are simpler intended functions. By restricting these replacement programs to a limited set of program structures, the equivalence of successive versions of the design is more readily verified.

A *Systematic Design Workshop* (*SDW*) extends the stepwise-refinement discipline of SPW to include the design of sequential programs with retained data, modeled on finite-state machines. State machines provide for encapsulating collections of data, with access limited to a fixed set of related programs. Concepts are introduced that allow the designer to relate initial abstract representations of state data to later, more specific representations.

An *Advanced Design Workshop/Seminar* (*ADW/S*) extends the design concepts of SDW to include the design of concurrent systems, modeled on networks of communicating state machines. The designer is asked to view the overall system as a state machine, and then to partition state data to define a network of state machines. Each input is identified with a path through the network. Several options for introducing and controlling concurrency within this framework are discussed. The approach is applicable to a wide variety of hardware configurations.

The *Software Management Workshop* (*SMW*) includes a review of concepts of FSD's functional organization, and the specific role of the software engineering function. The primary emphasis is on the principles underlying FSD's software standards and practices, in the context of management of the software product, the technical methodology, the organization, the development environment, and the customer.

Education has been carried out by a small group of instructors drawn from the professional programming cadre in FSD. This was a departure from the traditional method of using full-time educators who have less actual programming experience. The courses themselves represented new offerings, much more technical than most internal training programs. The level of difficulty was also high, for job-related required courses. Nevertheless, student achievement has been excellent, and course evaluations by the students have been highly favorable.

A principal software investment priority is the development of the tools needed to reinforce software engineering and establish an environment of modern methods. Current emphasis is on two tools, a *system development laboratory* and a *programming support library*, because of their applicability to multiple facets of the program.

The software development laboratory establishes a more complete and uniform programming environment. The laboratory includes interactive, batch, and dedicated development facilities for design creation, program generation, simulation, and target machine execution. The implementation of this discipline uses proven off-the-shelf development software tools that are integrated to address the software development process. This approach isolates validation of system design to the software design activity, implementation to the software development activity, and software-hardware interaction to the system test activity. The laboratory approach provides the system developer with a single interface for the entire software development process.

The integrated development discipline uses commercially supported large-scale operating systems to aid in the reduction of life-cycle costs. With time-shared resources, small projects and large projects alike can use the total technology without incurring the expense of a large dedicated resource.

The programming support library helps organize and control a programming project. It serves as the means of communication among development personnel and forms a standard interface between programmers and the system development laboratory. The programming support library is designed to provide a complete hierarchical library facility. As such the library supports the code

management and machinability needs associated with process design language during design, and it supports programming languages during module development, software integration, and product release. Thus this library is a key element in the application of software engineering technology. It is a collection of computer programs, disk-resident libraries, and operating procedures that provide facilities for programmers to store, edit, compile, and execute programs under development. Typically, the library produces summary statistics and analyses of such activities as storing, compiling, and executing programs.

Software engineering process assessment

One of the best ways of estimating future software development efforts and schedules is to rely on past experience. Yet few managers have access to recorded development data. A data base and retrieval system that can be used by line management to retrieve single-project development data as well as composite reports of selected categories of software makes possible a comparison of one manager's experience with that of managers of comparable efforts. The data base is also a source of information on software development.[3,4] In this role, it can support evaluations of the effectiveness of the software engineering program.

Individual projects may involve several categories of software, such as application, diagnostic, and support software. Application software can be further broken down by such characteristics as real-time signal processing, process control, and on-line graphics.

Data are associated with specific life-cycle activities, each of which is evaluated to determine key data parameters to be collected. Quantitative data and, more subjectively, project success judgments are included. The criteria for determining the pertinence of data are useful to software managers for characterizing and estimating the size of software projects.

Concluding remarks

Recent advances in software technology have necessitated a coordinated program involving people, tools, and practices. Education through a rich curriculum of software engineering courses is well underway. The development of tools that establish the proper environment for good programming is in progress, and software engineering practices are assembled and ready for use.

This program reflects an understanding of the software development process. With a realistic assessment of traditional methods,

software engineering includes a comprehensive collection of technical and management practices that can be applied today. Beginning with software system requirements recorded in source libraries, advanced techniques are now available to permit the conceptual abstraction, modularization, and structuring of designs that reduce complexity to manageable proportions and improve the completeness and correctness of the resulting software product. Modern development tools and uniform code management practices support orderly code generation in high-level languages, with storage in hierarchical libraries from which patch-free, quality source code products are delivered. Integration engineering is emerging as a distinct organizational function with a foundation of advanced technology. Management practices provide the methodology for balancing cost, schedule, performance, and quality perspectives.

As a result, software product quality can be dramatically improved by the routine application of modern design practices that contribute directly to program correctness and error avoidance through simplified and understandable designs. The manageability of software can be improved through uniform practices that govern plans, controls, and cost management, and through technology innovations that greatly improve the visibility of the software product for more effective management and control. Productivity improvements result primarily from the improvements in software product quality and the elimination or reduction of software errors. This reduces error detection and correction during testing. In addition to simplified designs, broader usage of higher-level programming languages and improved support tools also contribute to productivity gains.

Taken together, the software engineering discipline is a broad attack on the problems faced by the software community.[26] By emphasizing methodology and theoretical foundations, FSD has attempted to establish a common level on which each individual can build to the full extent of one's own creative ability. The benefit has been steady improvement in measured results over a sustained period of time.

The author is located at the IBM Federal Systems Division, 18100 Frederick Drive, Gaithersburg, MD 20780.