SYSTEMS ENGINEERING TRENDS

Systems engineering as a discipline has existed for about a half century; it is considered to have originated in the blending of the theoretical foundations of systems science, operations research, and the World War II production experience. In its early stages, the concerns of systems engineering were how to engineer, conceptualize, develop, and evaluate systems using operational tools and methods. Because of the necessity to adapt to advanced and rapidly changing technology, systems engineering has also started to include a managerial role. In this article, we will discuss the past approaches to systems engineering, the current focus of systems engineering, and the emerging technology that can be applied to systems engineering, particularly from an information systems technology perspective.

WHAT IS A SYSTEM?

What is a system? Ackoff (1) briefly defined a system as *a set of interrelated elements.* According to this definition, a system has more than one element and can be treated as one entity or set. The word interrelated implies that a system means more than a collection of elements. From the perspective of interrelatedness or mutual interaction, O'Connor (2) points out the differences between a system and a heap. For example, simply adding pieces to a system or cutting a system in half does not make a redoubled system or two smaller systems, whereas two divided smaller heaps have the same properties as the original heap. This is because of the properties of systems known as *emergent properties;* from the mutual interaction of the elements of a system there arise characteristics which cannot be found as characteristics of any of the individual elements. Examples of systems, moving from the least complex and smallest to most complex and largest, are as follows: cell, organ, person, community, state, nation, world, solar system, galaxy, and universe.

Classes of Systems

There are many ways to classify systems. We discuss two of the most typical and useful classifications here. The first deals with the categories of closed and open systems. The second deals with the categories of systems based on the theory of evolution.

A closed system is a system that does not need to interact with its environment to continue to exist; that is, it tends to be self-contained. As a result, the interactions of elements tend to be more stable and predictable. Examples are most mechanical systems. It is important to note here that it is an accepted principle that no system can continue to operate well without interacting with its environment. However, it is possible to treat a system as a closed system for study or design when there is a situation where inputs and outputs are known, defined, and predictable. Open systems, on the other hand, are organic and must interact with their environment in order to maintain their existence. In an open system, both internal and external elements are of concern. Examples of open systems are business organizations. An open system is adaptive and self-organizing in that it can change its internal organization in response to changing conditions. (A system that fails to do so is a malfunctioning system—these are beyond the scope of this article.) In other words, it is one of the purposes of systems engineering to guide a system to work properly.

Bellinger (3) provides another classification of systems based on the theory of evolutionary hierarchy. These are the parasitic system, prey–predator system, threat system, exchange system, integrative system, and generative system. In a parasitic system, an element that positively influences another is in turn influenced negatively by the second. In a prey–predator system, the elements are essentially dependent on each other from the perspective that the existence of one element determines the existence of the other element. If all preys die out, the predators will also die. In a threat system, for example, the United States–Soviet Union Arms Race, one element's actions are contingent on the actions of the other. The threat of action by each element is an essential deterent to the action of the other element. An exchange system is a system in which elements of the system provide goods and services to other elements in exchange for money or other goods and services. An integrative system is a system in which elements work together to accomplish some common desired objective or goal. A generative system is a system where, for example, people come together to create something neither of them had any idea about when they began.

Complex System

We often refer to a large system as a complex system because it has many elements and interactions which increase in an exponential manner as the number of elements increases. There are some important points that are needed to clearly describe a complex system, so that they can serve as a basis for any investigation of a complex system. First, a complex system consists of many independent components. Second, these components interact locally in the system. Only two components interact with each other at a time. However, this does not exclude the possibility of interaction with a third (or more) component within very short time frames. Third, the overall behavior is independent of the internal structure of the components. It is possible to have multiple systems that perform equally regardless of its internal structure. That is, two different systems with the same emergent properties can exist. Thus, we are able to observe a complex system without the knowledge of individual components. (As we shall see later, these characteristics of complex systems map easily onto the object-oriented problem-solving approach.)

Systems Approaches

The systems approach is one of the forms of methodological knowledge and is essentially an interdisciplinary approach. Three approaches that had a profound influence on systems theory are the approach of general systems theory founded by Ludwig von Bertalanffy (4), the cybernetics approach founded by Norbert Wiener (5), and the systems dynamics approach founded by Jay W. Forrester (6). Among these, general systems theory and cybernetics are the main streams that have arguably influenced systems research and development the most.

First, general systems theory (or systems science) (7) is the "transdisciplinary study of the abstract organization of phenomena, independent of their substance, type, or spatial or temporal scale of existence." Bertalanffy emphasized openness of system, interaction with the environment, and evolution resulting from emergent properties. He considered systems as mathematical entities and used mainly mathematical methods to describe and classify systems. Thus, systems theory is often criticized because of its abstractness, but its direction toward interdisciplinary study and unity of science is considered to be one of the important aims to the scientists in many areas related to engineering.

The second approach, cybernetics, is defined by Wiener (5) as "the study of control and communication in the animal and the machine." Wiener focused on the importance of maintenance of system parameters dealing with control and communication (information systems). In particular, homeostasis or adaptation and interaction with system and environment was his concern. In fact, cybernetics and systems theory deals with the same problem—that is, the system as a whole instead of as a collection of parts. The major difference is that cybernetics focuses more on the functional side of system characteristics (mostly on self-regulation), whereas systems theory focuses more on the structural side of system characteristics (mostly on relations between parts). While these systems approaches produced theoretical constructions mainly in terms of mathematical advances, systems engineering arose as a result of practical applications of systems approaches. Eventually, scientific knowledge obtained from these systems approaches contributed to the theoretical basis of systems engineering, which can be considered a technology rather than a science.

SYSTEMS ENGINEERING

What is Systems Engineering?

As systems became larger and more complex, the responsibility for systems design could not be conferred on one person or a few people in a group. The principle that was applied to solve this was division of labor; this principle was applied to systems design by decomposing a large system into smaller subsystems or components, if necessary. After the subsystems were designed, they were combined together to make a complete system. Such efforts have been successful in many ways. New systems pursuing various goals have been developed, with one of the major successes being the impressive systems development project of trips to the moon.

Dealing with the process related to decomposition and combination toward efficiency and effectiveness was the systems engineer's job. Yet, there was another problem that had to be considered. Simply connecting together individual subsystems does make a system, but this possibly haphazard system cannot guarantee the working system and sometimes results in the system exhibiting counterproductive behavior with respect to the goal. Chase (8) pointed out this aspect very clearly: "Systems engineering deals with the process of selecting and synthesizing the application of the appropriate scientific and technological knowledge in order to demonstrate that they can be effectively employed as a coherent whole to achieve some stated goal or purpose." Sage (9,10) also notes that systems engineering is a management technology that emphasizes the interaction between science, the organization, and its environment, with information serving as a catalyst that facilitates the interactions.

In order to deal with knowledge concerning systems engineering, we need to understand the three perspectives of systems engineering. As is often the case, we may define systems engineering according to structure, function, or purpose. In Table 1, we adapt several pertinent definitions of systems engineering from Refs. 9 and 10.

Throughout this article, we will use three hierarchical levels of systems engineering, those that can be derived from functional definitions, structural definitions, and purposeful definitions, respectively; these in turn give rise to systems engineering methods and tools, systems methodology, and systems management.

Three Levels of Systems Engineering

The first level, systems engineering methods and tools, can be considered to be the lowest level of systems engineering. At this level, product-oriented systems engineering approaches are used. Most of the operational methods and tools at this level were developed throughout the early stage of systems engineering within operations research and systems science. The combination of these methods and tools contributed to the development of systems of high quality and effective costs. This level supports the process level of systems engineering, systems methodology, which refers to process level and in turn supports systems management, which refers to the strategic level. Nowadays, due to the effect of computers and information technology, systems engineering faces the new challenge of integration of its operational tools and methods with automated tools such as computer-aided software engineering (CASE), computer-aided design (CAD), and computer-aided engineering (CAE).

The second level, systems methodology, is the process-oriented level. In this perspective, a system is often achieved through appropriate systems development life cycles, which will be discussed later. So far the dominant methodologies for system analysis and design have ranged from traditional systems analysis and design during the 1960s, to structured analysis and design in the mid-1970s, to information engi-

Table 1. Definitions Used in Systems Engineering

Structural:	Systems engineering is management technology that can be used to assist clients through the formulation, analysis, and interpretation of the impacts of proposed policies, controls, or complete systems, based upon the perceived needs, values, and institutional transactions of stakeholders.
Functional:	Systems engineering is an appropriate combination of theories and tools, carried out through the use of a suitable methodology and the set of systems management procedures, in a useful setting appropriate for the resolution of real-world problems that are often of large scale and scope.
Purposeful:	The purpose of systems engineering is information and knowledge organization that will assist clients who desire to develop policies for management, direction, control, and regulation activities relative to forecasting planning, development, production, and operation of total systems to maintain overall integrity and integration as related to performance and reliability.

Adapted from Sage (9).

neering in the 1980s, and to object-oriented analysis and design from the mid- to late 1990s. In addition to systems development life-cycle methodology, other important aspects of the systems engineering process at this level are quality assurance, configuration control, and structural economic analysis.

The highest level, systems management, is at the organizational or strategic level. By definition, systems management provides products or processes with technical and administrative direction and control (10). The functions of systems management includes planning, allocating resources, organizing, directing, and controlling work. Thus, systems management comprises the technical direction and the efforts needed for management of systems definition, development, and deployment. Organizational environment, organizational cultures, strategic quality, strategic cost and effectiveness, process reengineering, and process maturity are some of the concerns of systems management. The pursuit of speed and betterment and the quest for implementing projects inexpensively are some of the issues that lead systems engineering to focus on organizational and strategic levels based on the process-oriented view of system.

Systems Engineering Life Cycles

Similar to plants, animals, and humans, systems have a life cycle, and they evolve to the next generation by changing over time and adapting to their environment. From a systems engineering point of view, life cycle is defined as “the scope of the system or product evolution beginning with the identification of a perceived customer need, addressing development, test, manufacturing, operation, support and training activities, continuing through various upgrades or evolutions, until the product and its related process are disposed of” (11).

The use of life cycles may also be considered as an adoption of functional decomposition to systems engineering tasks in order to identify a systems engineering process easily. A life cycle often requires clear understanding of what a systems engineering product is, as well as how efficiently it can be produced. A systems engineering product, often called end product, is not confined to hardware or software. For example, it can refer to personnel, services, or even processes themselves. A pilot may be produced in an airforce academy, a new telephone service in a telephone company, or a new filtering process in an oil company. Depending on stakeholders' needs, the end product of a system can vary drastically; however, the life cycle of systems engineering tends to be similar in any system. The following basic steps and phases used in systems engineering life cycle will explain this clearly.

There are three fundamental steps (9) needed for problem resolution: issue formulation, issue analysis, and issue interpretation. Issue formulation is an effort to identify the needs to be fulfilled and the requirements to be satisfied, constraints that affect issue resolution, and generation of potential alternative courses of action. Issue analysis is performed to determine the impacts of the identified alternatives. At this step, by comparing alternatives, we are able to select one alternative for implementation or further study. These three steps are the ingredients for the phases of systems engineering life cycle. Many life-cycle models have, in general, at least five phases—for example, initiation, design, development, implementation, and operation management. Sage (9) identified three basic phases for simplicity, which are systems definition, systems development, and systems deployment. The systems definition phase entails requirements, specifications, and a conceptual design so that systems development can begin. At the stage of systems development, a logical and detailed design of the system, along with operational implementation and testing are involved. The next is the system deployment phase, at which time the product is fielded and implemented in an operational setting for evaluation and modification and maintenance. This phase continues until another new system development initiative appears to substitute for the existing system.

Software Development Life-Cycle Models

Many software development life-cycle models are utilized in order to acquire and develop trustworthy software. Because the systems engineering discipline has had a close relationship with computer and information technology, many of the software systems engineering life-cycle models are based on systems engineering lifecycle. Conversely, the life-cycle models in software systems engineering are applicable to the general systems engineering life-cycle model.

The first systems-engineering-based life-cycle model for software development was the one introduced by Royce (12), known as the waterfall model. Royce described the pattern of downward flow of information and development effort. Based on this model, many modified waterfall life-cycle models have appeared. Boehm (13) also defined seven phases for his waterfall life-cycle model as shown in Fig. 1(a). The Department of Defense (14) extended the approaches, as shown in Fig. 1(b), to split the systems development effort in two ways: hardware effort and software effort. These life-cycle models are typical examples of traditional waterfall models. The major advantage is the capability to manage complexity of system development by splitting it into activities or phases. Although the waterfall model is criticized as being slow, inflexible, and costly, later models do not seem to throw away the advantage of manageability, and we see that most of these also follow similar phases, as shown in Fig. 1.

Sequential development, the method used in the waterfall model, is often not possible to carry out, especially when one or more phases needs to be repeated due to the possible omissions in each phases. In order to handle such cases, Boehm (15) created a somewhat different life-cycle model (called the spiral life cycle) that emphasizes the need for iterative development. Figure 2 depicts the comprehensive spiral model, and it shows how formulation, analysis, interpretation 1, and interpretation 2 steps in each quadrant repeat until the final product, software, is developed (10). This representation of the life-cycle model contains the three steps and three phases in the systems life cycle discussed above. The advantage of this is that instead of having one step for interpretation, the spiral model dissociates interpretation of the software engineering plan for the next phase from the first interpretation by emphasizing the iterative and evolutionary characteristic of the life-cycle model.

SYSTEMS DEVELOPMENT

In a broad sense, systems development encompasses all tools and techniques, as well as efforts to manage them, in order

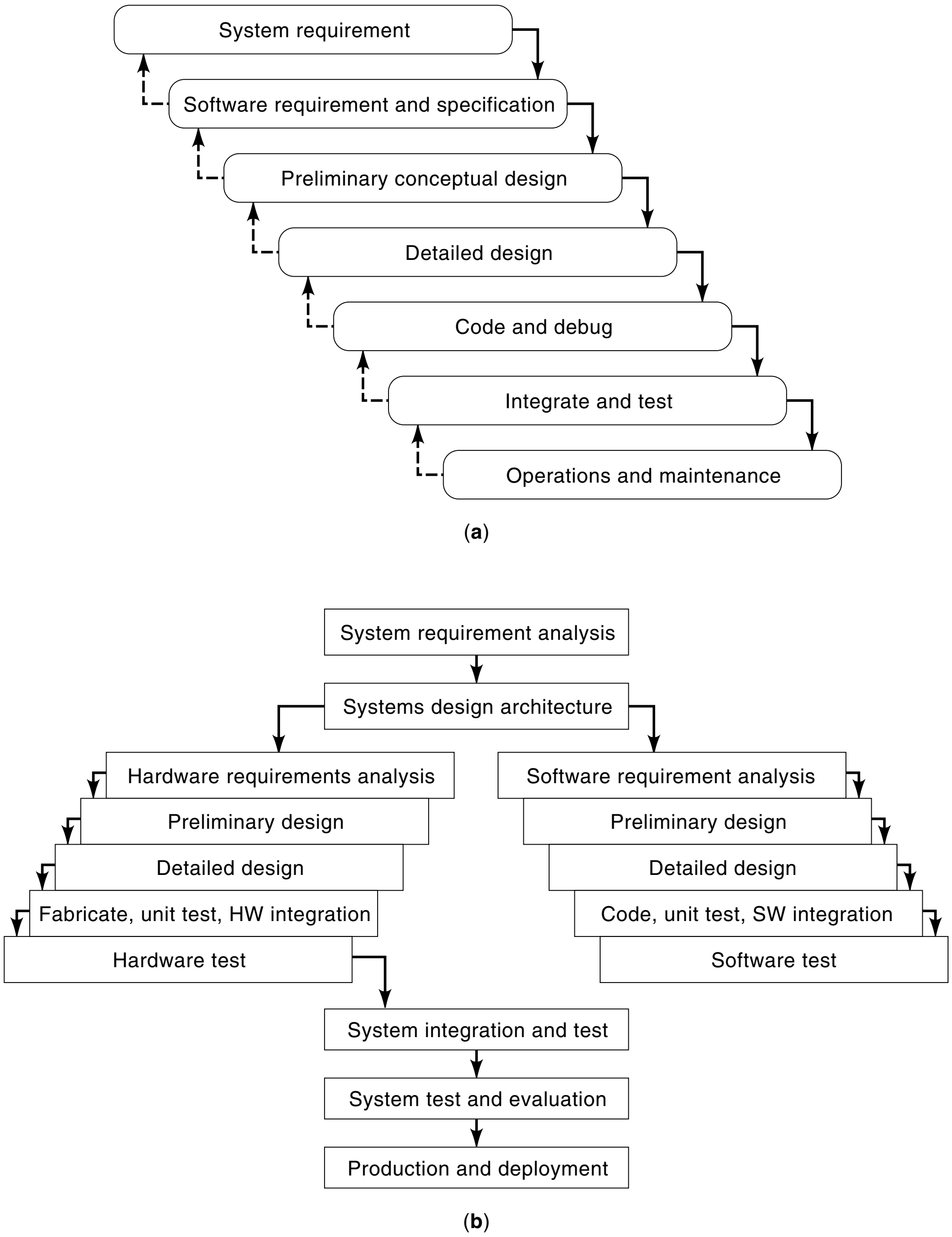

Figure 1. (a) Waterfall software development life-cycle process model of Boehm. (b) Software development life cycle by US Department of Defense. [From Sage (10).]

to achieve an effective and efficient system. Systems development is related to the diverse areas of human systems, economic systems, and business systems. In most modern systems, information is the critical factor that allows the interactions among the various areas. In this section we focus on information systems in an organization.

As we discuss systems development, we will see many systems development methodologies that have evolved. Research done by Mahmood (16), which compared the traditional systems life-cycle approach to the prototyping approach, showed that neither of the methods is preferred unanimously by both the system designer and system user. Some methods performed better in some areas than in others, thus implying that methods should be selected depending on project, environment, and decision characteristics.

In a comprehensive article, Wetherbe and Vitalari (17) discussed key issues governing systems development. They conducted a survey about key issues that included philosophy, project management, and tools and techniques. Forty-two directors of systems development were polled. Most responded that they were strongly moving toward the new trends in those issues (presented in Table 2).

These trends in systems development philosophy reflect the characteristics of strategic rather than operational, process rather than product, and distributed rather than central. In the area of tools and techniques, there is much emphasis on new methodologies such as objected-oriented methods, joint application development (or joint application design), and rapid application development. Each methodology gives us general guidelines about which steps to take and how to model—that is, provide the overall framework that enables the system designer to organize problem-solving work. Thus, methodology is critical when designing a large and complex system.

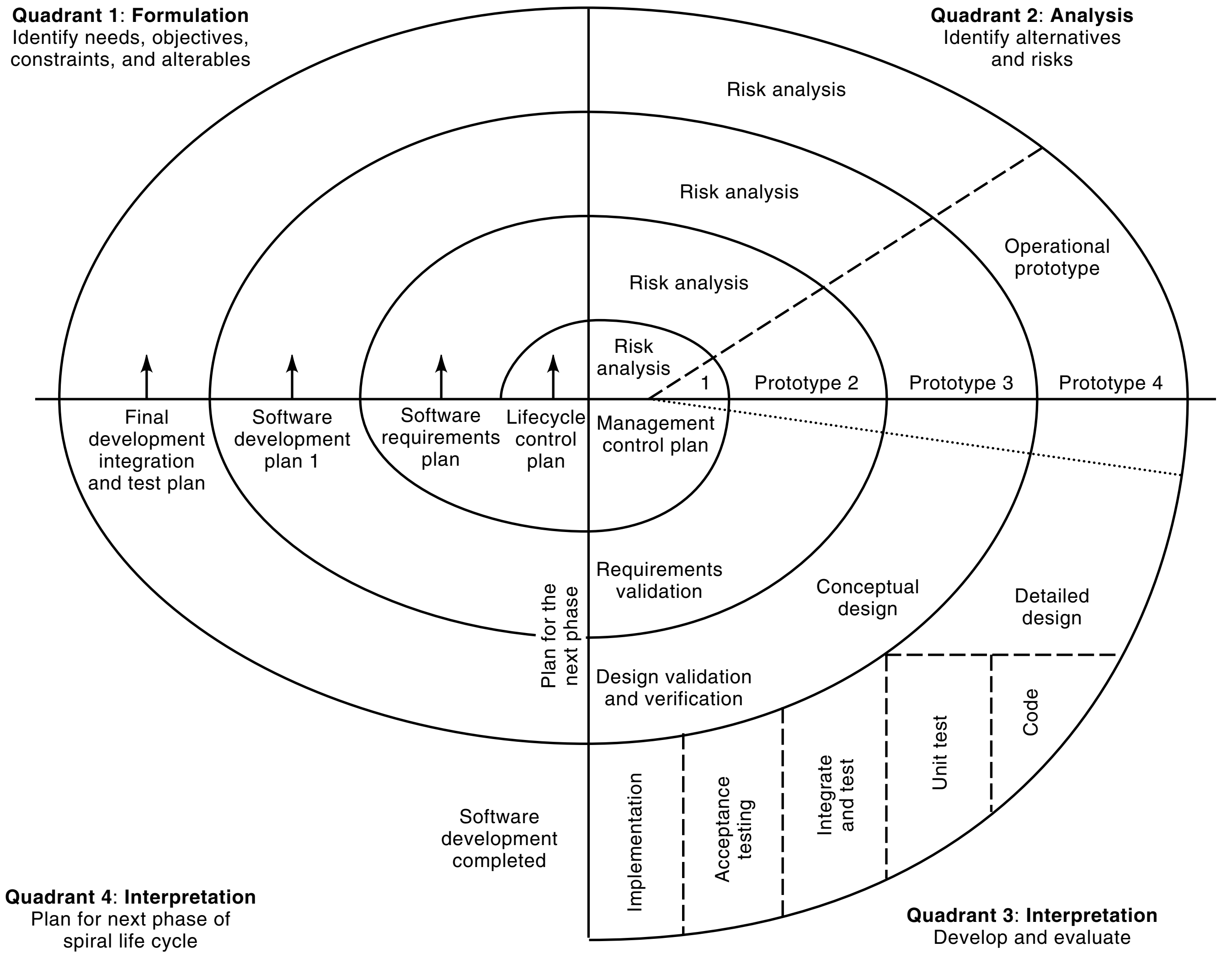

Figure 2. Spiral model for software systems engineering. [From Sage (10).]

Table 2. Key Issues in Systems Development

	The Old	The New
Systems development philosophy	Operational efficiency	Strategic/customer-driven
	Automation	Business reengineering
	Quality control	Total quality
	Cost justification	Value added
	Charge out	Allocation
	FIFO/bottom-up	Information architecture
	Mainframes	Multiple platforms/networking
	User/IS dependency	End-user self-sufficiency
Project management	Multiyear project	Time box
	Matrix management	Small self-directed teams
	Project control	General contractor model
Tools and techniques	Interviews	Cross-functional JADs
	Life-cycle methodologies	Projection prototyping
	CASE	I-CASE
	Data dictionaries	Repositories
	Code-level maintenance	Model-level maintenance
	Code redundancy	Reuse
	Contracting	Outsourcing

FIFO: first in, first out; JADs: joint application designs; CASE: computer-aided software engineering; I-CASE: integrated systems for computer-aided software engineering; IS: Information Systems.
Adapted from Wetherbe and Vitalari (17).

Whitten and Bentley (18) provide an in-depth definition of methodology incorporating various aspects that can occur in systems development. Methodology is the physical implementation of the logical life cycle that incorporates (1) step-by-step activities for each phase, (2) individual and group roles to be played in each activity, (3) deliverables and quality standards for each activity, and (4) tools and techniques to be used for each activity. Thus, it is a set of tools and techniques with a systematic description of the sequence of activities, and it also includes systems management activities such as configuration management and quality assurance issues in each phase of system development life cycle.

Most literature focuses on two approaches for systems development: traditional systems development and object-oriented systems development. For example, Dewitz (19) points out that traditional systems development focuses on what a systems does (i.e., on the verbs that describe the system), while object-oriented systems development focuses on what a system is made of (i.e., on the nouns that describe the system). Since objected-oriented development is an emerging trend, it will be covered later and we start with the traditional systems development.

Traditional systems development began with the framework that focuses on the functional decomposition emphasizing the process or functions that a system performs. The framework of waterfall systems development methodology is valid with minor modifications across most of the methodologies. For example, Whitten and Bentley (18) defined five phases for systems development: systems planning, systems analysis, systems design, systems implementation, and systems support. Two major general methodologies for business information systems are (a) structured analysis and design by DeMarco (20) and Yourdon (21) and (b) information engineering by Martin (22). Structured analysis and design methodology emphasizes processes in systems development and is often referred to as the data flow modeling methodology. In structured analysis and design methodology, models such as system flowchart and hierarchical input–process–output chart (HIPO) models, even before the use of CASE tools, clearly contributed to systems development since they played a role of replacing inadequate English texts with graphical views to identify system functions. In order to cope with large systems, modern structured analysis and design (22) has emerged with the models of data flow diagram, data-dictionary, entity-relationship diagram, structured English, and structure chart. Information engineering is often referred to as data modeling methodology because of the emphasis on data. In addition, information engineering is process-sensitive and has the characteristic that it can be extended to strategic plans. This characteristic has resulted in the popular use of this method in many business areas. This methodology consists of the four phases of information strategy planning, business area analysis, systems design, and construction.

Traditional waterfall systems development methodology based on life cycle, as shown in Fig. 1, has three major problems (16). First, systems development delays the delivery of systems to users until the last stages of system development. Second, it requires specified systems outputs at the outset because an ambiguous system output may result in redevelopment, which is costly and time-consuming. Third, it creates communications problems because system users are often involved only in the systems requirement phase and because communication between nonadjacent phases would be difficult in case those phases were performed by different functional departments or groups. The effort to solve these problems led to the approaches of joint application design, prototyping, and rapid application development.

Joint Application Design

Joint application design was originally developed at IBM to reduce communication problems between designers and users through the use of structured workshops, called JAD sessions. Additional benefits such as early detection of design problems, reduced time and effort, and user satisfaction made JAD applicable to many methodologies, and thus many versions of JAD are available to system designers. JAD includes complete specifications throughout five phases (23) of project definition, research, preparation, the JAD session and the final document. Table 3 shows the five phases, the steps in

Table 3. Five Phases for Joint Application Design

Phases	Step	Work Days	Resulting Output
Project definition	Interview management	1–3	Management definition guide
	Produce the management definition guide	1–3	
		1	
Research	Get familiar with the existing system	1–4	Work flow
	Document work flow	1–5	Preliminary specifications
	Research data elements, screens, and reports	2–4	JAD session agenda
	Prepare the session agenda	1	
Preparation	Prepare the working document	2	Working document
	Prepare the JAD session script	3–5	JAD session script
	Prepare overheads, flip charts, and magnetics	1	Overheads, flip charts, magnetics
The JAD session	Hold the session	3–5	Completed scribe forms
Final document	Produce the final document	3–10	JAD design document
	Participants review the document	2	Signed approval form
	Hold the review session	1	
	Update and distribute the final document	2	

JAD: joint application design.
Adapted from Wood and Silver (23).

each phase, the time required, and the resulting outputs. Since JAD can be used only at the initiation, analysis, and design phases in the systems development life cycle, it is often combined with other methods to fit all phases of the systems development life cycle.

The selection of the right people for the JAD team is the most important factor for the success of the JAD session. The JAD team includes an executive sponsor, JAD leader or facilitator, scribe(s), and full-time participants. The critical role of executive sponsor is to define the purpose, scope, and objective of the project. During the session, the executive sponsor should at least be accessible by phone to resolve possible cross-departmental conflicts that might delay the JAD session. The JAD leader should be an impartial business person or professional who has experiences in group facilitation, JAD methodology, group dynamics, and the products being developed. The JAD leader is responsible for making an agenda by gathering information on workflow and system requirements before the session. During the session, the role of JAD leader includes that of impartial referee, discussion leader, and negotiator. Once the session is over, the JAD leader's concern moves to creation, review, and distribution of final document. A scribe's role is critical to a successful JAD because the notes taken by the scribe evolve into the final document and the scribe is usually selected from management information systems (MIS) personnel. The development of convenient word processor and CASE tools made it easier for the scribe to apply notes directly to the final document. JAD participants include users and MIS people who are involved in making decisions about the system design. Users range from end-users to supervisors who can provide valuable input on design issues and prototypes, specify training needs and acceptance criteria, and ensure that the system captures critical success factors. As observers, MIS personnel attend the session hoping to obtain a clear understanding of user needs and systems requirements, which will be reflected in actual system development. JAD is appropriate for generative systems, since traditional waterfall methodology requires specific goals before the initiation of systems development.

Prototyping

Prototyping is another conceptual technique that can be applied to many systems development methodologies. Conceptually, prototyping approaches utilize two enhancements to traditional systems development. One is incremental development, and the other is evolutionary development. Incremental development as shown in Fig. 3(a) is similar to the spiral life cycle which tries to correct the problems of the traditional waterfall life-cycle model. In this approach the product is delivered at the end of each iteration with add-ons from previous products. The first product is made of only a kernel (10) that has minimal functions of a system, called a prototype; and as iteration goes on, the product advances toward full functionality. Evolutionary development as shown in Fig. 3(b) is similar to the incremental model, but the product in each cycle is a complete product in terms of functionality. Another difference is that fundamental changes in the product after each cycle are possible. This model is often implemented in many object-oriented systems development scenarios where object classes can be easily redesigned.

Rapid Application Development

Rapid application development (RAD) is a variation of incremental and evolutionary development models that use the prototype approach. RAD (24) is defined as a systems development methodology that employs evolutionary prototyping and incremental prototyping techniques to deliver limited functionality in a short timeframe. The timeframe, called a timebox, is usually 3 to 6 months, and is nonextendable. Incremental development models are applied within the timebox. The major difference between JAD and RAD is that RAD covers all the phases in systems development. Through JAD sessions, RAD balances the use of prototyping approaches with modeling and continual feedback and determines what will be achieved for each timebox. Similar to prototyping, RAD focuses on the most important system functions at the beginning of an iteration. The philosophy of the "timebox" approach is that it is better to have a working system of limited functionality quickly than to wait years for a more comprehensive system (25).

The goals of systems development—fast, better, and cheaper development—are realized in RAD. Since the measure of quality often depends on the system user, the request for change by users in RAD activities always shapes a high-quality system. In order to achieve cost goals, many software tools are heavily used throughout development stages. Examples are the use of graphical user interface (GUI), frontware, fourth-generation languages (4GLs) including relational database management system (RDBMS), and various CASE tools.

Methods, Models, and Tools

The terms methodology, method, model, tool, and technique are considerably interrelated. Methodology in systems development, as we defined earlier, is the most overarching concept, but tends to be somewhat conceptual. It often comes with various models, tools, and techniques in order to solve problems arising in systems development. In general, methodology refers to a general principle or theory to be used in problem solving regarding a specific area of study. Hence, we may refer to traditional (waterfall) methodology, structured methodology, information engineering methodology, and so on. In our definition, a method refers to a specific instance of a methodology. In that sense, many methods may exist in one methodology, or a methodology may be composed of many methods with a common principle. A model, a representation of the real world, or a system in the process of development can be a method bundled together with tools or techniques. Again, this method (or model) can be an exemplar of a certain methodology, and, in turn, from this methodology many revised methods can appear later. The distinction between tool and technique is not well defined due to their close relationship with each other; however, following the definition of Hackathorn and Karimi (26), we refer to the term technique as a procedure for accomplishing a desired outcome and the term tool as an instrument for performing a procedure. A data flow diagram is a good example of a technique, and the software to draw a data flow diagram can be considered as a tool. CASE tools, for example, were originally considered as tools. However, nowadays CASE tools tend to be called method or methodology as they grow to handle integrated system development as well.

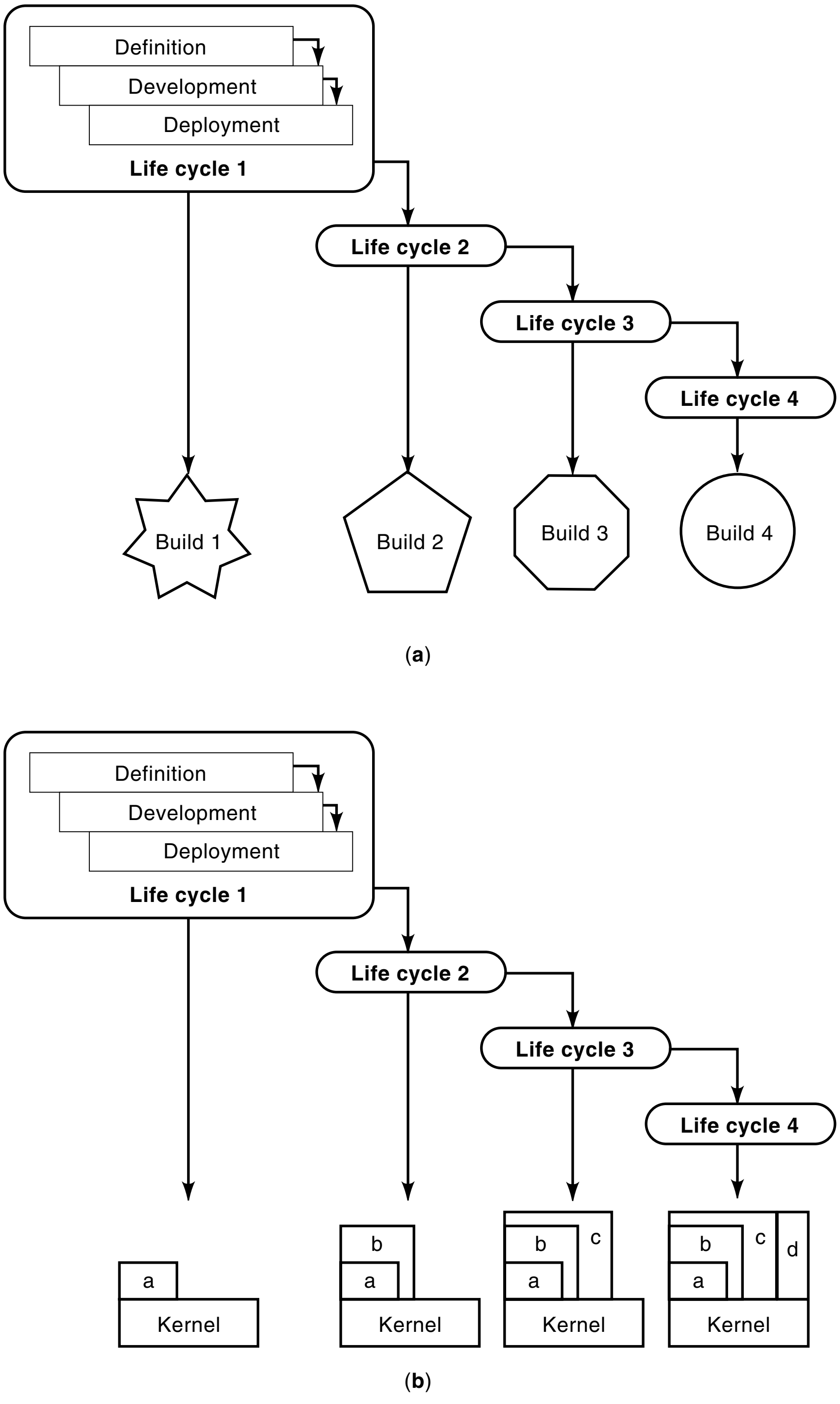

Figure 3. (a) Evolutionary life-cycle model and (b) iterative life-cycle model. [Adapted from Sage (10).]

Hackathorn and Karimi (26) conducted a comprehensive research survey comparing many information systems development methods. (They used the term method referring to CASE tools.) Even though their research covers only traditional systems development methods developed up to 1986, it provides a framework for comparing current systems development methods and shows the trends of methods, so that we can select appropriate method in real practice. Two dimensions, breadth and depth, were utilized for analysis, and twenty-six systems development methods were located on two-dimensional space. For the breadth dimension, they used five phases of systems development, and for the depth dimension they used the terms tool, technique, and methodology, depending on how practical or conceptual the method was. Not surprisingly, their conclusion was that no method was perfect for the whole range of breadth and depth, hence employment of a set of methods to cover the whole life-cycle phase was recommended. Further, they described the three stages of method evolution. The first consists of tools and techniques for application development that corresponds to lower or back-end CASE tools focused on systems implementation such as code generator or application generator. The second stage shows two trends—broader systems development techniques and emergence of methodologies for organizational analysis that correspond to upper or front-end CASE tools. The third stage shows the emergence of information engineering methodology to link the first two stages. The trends tend to merge lower CASE with upper CASE.

CONFIGURATION MANAGEMENT, METRICS, AND QUALITY ASSURANCE

So far, we have dealt with the information systems development aspect of systems engineering from the viewpoint of the life-cycle approach. This overall framework is of direct use in developing systems, but other issues such as quality assurance, configuration management, and metrics throughout the life cycle are important in supporting life cycles. Quality assurance, configuration management, and metrics are closely related to each other. Quality assurance is defined as a planned and systematic means for assuring management that defined standards, practices, procedures, and methods of the process will be applied. We can differentiate quality assurance from quality control: Quality assurance occurs during all the phases of the systems development life cycle with a focus on system processes, whereas quality control occurs at the end of the systems development life cycle with the focus on the end-product or system. Hence, quality assurance needs and is often achieved through configuration management. Configuration management is defined (10) as a systems management activity that identifies needed functional characteristics and nonfunctional characteristics of systems or products early in the life cycle, controls changes to those characteristics in a planned manner, and documents system changes and implementation status.

Configuration Management

While expensive, large, and complex systems are developed through systems engineering, it is likely that changes in product and process take place at the same time. Configuration management started with the idea of dealing with many problems that come from these changes. Back in the 1950s during the arms race, the Department of Defense (DoD) had many supporting and associate contractors, yet had weak control and minimal documentation of changes. DoD found that only the original manufacturer could supply systems or components because of inaccurate information resulting from changes in product design. Starting with ANA (Army, Navy and Airforce) Bulletin No. 390, which gave industry uniform guidelines for proposing aircraft changes, many government organizations such as the Air Force, Army, Navy, and NASA published their own document for the techniques of configuration management. Later, DoD incorporated the guidelines into the standard MIL-STD-483 (27) defining configuration management procedures and policies.

An alternate standard available for configuration management is the IEEE software configuration management standard (28) which describes what activities are to be done, how they are to be done, who is responsible for specific activities, and what resources are required. Major activities in configuration management, both in the DoD standard and IEEE standard, are grouped into four functions: configuration identification, configuration control, status accounting, and configuration audits and reviews.

Configuration identification activities identify names and describe physical and functional characteristics of the configuration items (CIs) to be controlled throughout the life cycle. A configuration item can be an element of the support environment as well as intermediate subsystem or the final deliverable. For example, operating systems used in systems development can be a configuration item in software configuration management. Naming methods of CIs include serialization, labeling, version marking, and so forth. Configuration control activities request, evaluate, approve or disapprove, and implement changes to CIs serving as a primary driver in configuration management. Configuration control does not require really new disciplines; instead, existing practices should be extended and systemized within the given policies and objectives of configuration management. For example, established committees like the configuration control boards (CCB) with the adequate level of authority to approve or disapprove change requests are usually recommended. Multiple levels of CCBs may be specified depending on the system or project complexity. For the projects that are not complex in structure, a central CCB may be installed and assume the responsibilities over several projects. Configuration status accounting activities record and report the status of CIs to ensure traceability and tractability of the configuration baseline. Baseline is defined as an approved reference point for control of future changes to a product's performance, construction, and design. Thus, configuration accounting activities should include information on what data elements in CIs are to be tracked and reported, what types of reports are generated, when those reports are generated, how the information is processed and reported, and how access to the status data is controlled. Configuration audits and reviews validate achievement of overall system or product requirements. Thus, configuration audits and reviews enable the integrity of the various baselines by determining to what extent the actual CI reflects the required physical and functional characteristics. In addition to four basic functions, IEEE software configuration management standard addresses requirements for interface control and subcontractor–vendor control. They are intended to support four functions by reducing the risk associated with items outside the scope of configuration management plans and items developed outside the project environment by contract.

Configuration management will continue to be a major factor in the definition, development, and deployment of a system as long as changes are inevitable during system life cycle. In particular, use of configuration management is becoming an essential activity when developing a software system where changes are more extensive and change faster than any other hardware systems.

Metrics

Metrics are the instruments needed for quality assurance and configuration control since without the measurement of product or process, no one can say an improvement has been achieved. Metrics are so important that a whole system is subject to failure if metrics are not able to measure system quality or cost. It is important to identify where metrics should be used, the appropriate time for them to be applied,

Table 4. Four Types of Measurement

Inactive:	This denotes an organization that does not use metrics or does not measure at all except perhaps in an intuitive and qualitative manner.
Reactive:	This denotes an organization that will perform an outcome assessment and after it has detected a problem or failure will diagnose the cause of the problem and often will get rid of the symptoms that produce the problem.
Interactive:	This denotes an organization that will measure an evolving product as it moves through various phases of the life-cycle process in order to detect problems as soon as they occur, diagnose their causes, and correct the difficulty through recycling, feedback, and retrofit to and through that portion of the life-cycle process in which the problem occurred.
Proactive:	These measurements are designed to predict the potential for errors and synthesis of an appropriate life-cycle process that is sufficiently mature such that the potential for errors is minimized.

Adapted from Sage (9).

and their purpose. Determining the object and moment to be measured is included in the task of measurement. Misuses of measure are often encountered, for example, confusing the scale (27), which can be classified into nominal scale (determination of equality), ordinal scale (determination of greater or less), interval scale (determination of equality of intervals or differences), and ratio scale (determination of equality of ratios). For example, matching the number of defects directly with the quality of a production system may not be appropriate when the interval or ratio scale is needed.

Measurements are often classified into four categories—inactive, reactive, interactive, and proactive (10)—as listed in Table 4. Measurement is needed at every systems engineering level: systems tools and methods, systems methodology, and systems management. Although all types of measurement are possible at each level, we can see a correspondence between the types of measurement at the levels of systems engineering. At the systems methods and tools level, product-oriented measurements—for example, metrics in inspections or quality control—are often reactive and metrics are used after the product appears. At the systems methodology level, process-oriented measurements—for example, metrics in configuration management or operational quality assurance—have interactive characteristics and generally measure a system's functionality through the whole life cycle. At the highest level, systems or process-management-oriented measurement—for example, metrics for strategic quality assurance or process improvement—are often proactive so that process improvement will occur. It is true that the higher the level, the more difficult it is to get a clear metrics system, yet ISO9000 and the capability maturity model have done yeoman service in this area.

ISO9000 series standards are considered to be most comprehensive quality assurance and management standards. These have been adopted by the European Community and many companies worldwide. ISO9000 series—ISO9000, ISO9001, ISO9002, ISO9003, and ISO9004—have their own purposes. ISO9000 is an overview document of suggestions and guidelines. ISO9000-3 specifies quality management and quality assurance standards for software and provides 20 guidelines for the development, supply, and maintenance of software. For quality assurance for the general systems engineering life cycle, ISO9001 is comprised of design, development, production, installation, and servicing. ISO9002 explicitly focuses on production and installation of products and systems. ISO9003 deals with the standards for final inspection and test. Finally, in ISO9004, the detailed list of what should be done is presented instead of general suggestions or guidelines. Figure 4 illustrates the twenty requirements and their relationship to ISO9000 family.

Cost Metrics

A systems engineer's interest in metrics has two aspects. One is cost-related, and the other is quality-related. For the estimation of software cost, often dealing with estimations of effort and schedule, the most common metric is lines of code (LOC). There are many ways of measuring LOC. Some include comments and data definitions in the measurement, but others only include executable lines or only newly developed

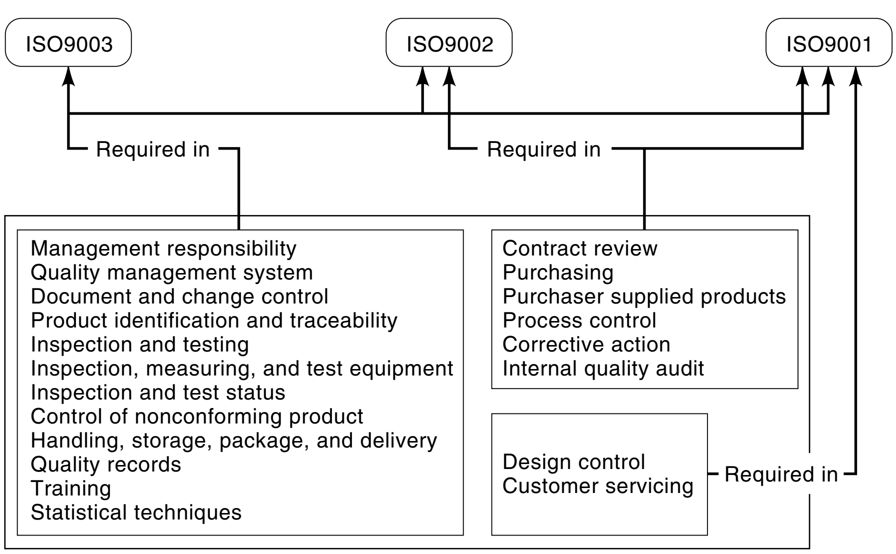

Figure 4. The 20 requirements and their relationship to ISO9001, ISO9002, and ISO9003. [Adapted from Sage (10).]

codes. Starting with the size of software, there are many factors to be measured. For example, Boehm (28) grouped many factors measured in various cost estimation models by size, program, computer, personnel, and project attributes. The effort to generalize the process of cost estimation has resulted in a flood of models, some of which examine only the software product (e.g., Wolverton's software cost–effort model), while others examine the process (e.g., COCOMO by Boehm).

Wolverton's software cost–effort model (29) is based on expert judgment on software complexity, the model estimates for types of software, and the development difficulty. The types are categorized as follows: control, I/O, pre/post-processor, algorithm, data management, time critical. The difficulty—determined by whether the problem is old (O) or new (N) and whether it is easy (E), moderate (M), or hard (H)—are categorized into OE, OM, OH, NE, NM, and NH. Thus, a 6 × 6 matrix for software type and software difficulty can be developed. To use this matrix, the software system is partitioned into modules i, where $i = 1, 2, \ldots, n$. Then, the cost of the kth module becomes

$$C_t(k) = S_k C_{t(k)d(k)}$$

where $C_{t(k)d(k)}$ represents the cost per line of code for that particular type and difficulty of software, and S_k denotes a size estimate of the software, which is measured in the number of lines of uncommented code for kth module. To get the total cost of producing the software system, we sum this cost over all modules to get

$$\text{Total cost} = \sum_{k=1}^{n} C_t(k) = \sum_{k=1}^{n} S_k C_{t(k)d(k)}$$

Wolverton's model can estimate the dollar-valued total cost based on the size, type, and difficulty of the software system. However, it is criticized because it has little consideration of process-related factors—for example, the effect of the position in software development life cycle on cost estimation. Too much reliance on one expert's experience could be another weakness of this model.

As opposed to the expert judgment approach, many empirical models based on regression have appeared for software cost estimation. Although it is a simple regression model, Nelson's cost estimation model (30) has been often utilized. It sets software system cost as a dependent variable and attributes or quantities of software cost as independent variables. An alternate nonlinear model has been proposed in many cost estimation studies. The Bailey–Basili model (31) is one of the examples. This model uses three basic estimator equations, which are

$$E = b + aS, \qquad E = aS^b, \qquad E = c + aS^b$$

where E denotes effort in man-months of programming and management time and S denotes number of developed lines of source code with comments. The model parameters were determined to minimize the standard error estimate (SEE). The SEEs are obtained by summing the squares of estimation error ratio (the difference between estimated effort and observed effort, divided by observed effort) over 18 projects. For example, The SEE for the third equation is

$$\text{SEE} = \sum_{i=1}^{N} \left[1 - \frac{(c + aS_i^b)}{E_i} \right]^2$$

Bailey and Basili attempt to improve on the estimator by calculating error range based on software complexity factors. They categorize complexity factors from 18 projects into three categories: total methodology (METH), cumulative complexity (CMPLX), and cumulative experience (EXP). At the next step, a least-squares estimation is used to calculate coefficients in the regression model that regress effort ratio (ratio between actual effort expended and the amount predicted by the background equation) on the three software complexity factors

$$\text{ER} = \alpha + \beta_1 \text{METH} + \beta_2 \text{CMPLX} + \beta_3 \text{EXP}$$

Based on ER value, the effort can now be adjusted to either

$$E_{\text{adj}} = (1 + ER_{\text{adj}})E$$

or

$$E_{\text{adj}} = \frac{E}{(1 + E_{R\text{adj}})}$$

according to the effect of software complexity. The adjusted effort can be interpreted as the effort for the life-cycle development.

Because the Bailey–Basili model is only based on the FORTRAN software product in NASA Goddard Space Center, the database is very homogeneous. Boehm (32) developed the constructive cost model (COCOMO) based on heterogeneous databases that include programs such as FORTRAN, COBOL, PL/1, Jovial, and assembly language ranging from 2000 to 1 million lines of code, exclusive of comments. His model has three estimates: basic, intermediate, and detailed. Each form of COCOMO model uses an estimate of the form

$$E = aS^{\delta} M(\boldsymbol{x})$$

where $M(\boldsymbol{x})$ represents an adjustment multiplier, which is a composite function of 15 cost drivers x_1 through x_{15}. In the basic COCOMO, $M(x) = 1$ for all x_i. For intermediate COCOMO, the adjustment multiplier is calculated as the products of individual cost drivers: $M(x) = m(x_1)m(x_2) \ldots m(x_{15})$. In detailed COCOMO, phase-sensitive effort multipliers are introduced to the model. Phase-sensitive effort components play an important role that reflect process-related costs. The parameter values are not obtained from least-squares regression as opposed to many other cost estimation models. They are obtained from subjective opinion and experience of software experts and from the results of other cost estimation models.

There is a different approach to cost estimation that tries to overcome the lack of standardization of LOC count which is used as a size measure. The function measure, originated by Albrecht (33), is more macro level than LOC, capturing information like the number of distinct input data items and the number of output screens or reports. The possibility of estimating cost early in the life cycle, along with the availability to nontechnical project managers, makes this model useful. In the initial function point developments by Albrecht (33), four elements—external inputs, external outputs, number of logical internal files, and number of external inquiries—were representative of the functionality of a software product. Soon, it is recognized that number of external interface types is to be added (10). To calculate the function count, the number of function points x_{ij} at each complexity level (low, average, high) is multiplied by appropriate weight w_{ij}, and is summed over function point type and complexity level. This results in

$$FC = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} x_{ij}$$

where x_{ij} represents the number of function points of type j and weight level i.

There are some problems with these models, as pointed out by Sage and Palmer (34). These are that it is difficult to build a model independent of either user or developer, to correctly measure cost factors, difficulty to incorporate the change in technology, tools, and methods, and difficulty to assign process-based factors to software cost.

Quality Metrics

Quality is so intangible that no one is able to measure it, whereas cost is expressed as a dollar value. However, by using quality metrics it is possible to say that "a system has high quality since it has not failed for 10 years." Thus, quality metrics such as defect rate enable us to determine quality. Just as LOC is the base metric for cost-related software metrics, quality metrics are also based on LOC. Kan (35) gives us three categories: end-product quality metrics, in-process quality metrics, and maintenance quality metrics. Examples and metrics are summarized in Table 5.

Information systems development methodology alone cannot satisfy an organization's goal of quality improvement. As we have seen, as a result of the importance of user satisfaction in development methodology that is incorporated into JAD and RAD, there is a big movement toward quality assurance that can be adapted to development methodology levels and ultimately to organizational levels. The Japanese philosophy about quality is characterized by the total quality control (TQC) system, which uses seven basic tools (36): checklist, Pareto diagram, histogram, scatter diagram, run chart, control chart, and cause-and-effect diagram.

A checklist is a paper form with items to be checked. It is used to gather and arrange data easily for later use. A Pareto diagram is a frequency bar chart in which the x axis and the y axis usually represent the causes and counts, respectively. A histogram is similar to a Pareto diagram, but in a histogram the x axis represents the unit interval of a parameter (e.g., time unit) by the order of which the frequency bars are shown. A scatter diagram shows the relationship between two variables by plotting points with respect to the x axis and the y axis values. It is useful when the linear relationship exists between two variables. A run chart tracks down the parameter of interest over time by using the x axis as time. It is very useful to capture the process characteristics of a variable. A control chart is similar to a run chart, but it is often used to control the outliers that are outside the upper control limit (UCL) or lower control limit (LCL). The cause-and-effect diagram, also known as the fishbone diagram because of its shape, is used to show the relationship between a quality characteristic and factors affecting that characteristic.

These statistical quality control tools often play a role in operational quality assurance, but strategic quality assurance seemed to be missing in the American implementations. Hence, the American view of the Japanese quality implementation produced the term total quality management (TQM), which identifies and adds factors hidden in Japanese culture

Table 5. Software Quality Metrics

Category	Example	Possible Metric
Product quality metrics	Mean time to failure	Amount of time before encountering crash
	Defect density	Number of bugs/KLOC (thousand lines of code)
	Customer-reported problems	PUM (problem per user month) = Total problems that customers reported for a time period/Total number of license-months of the software during the period
	Customer satisfaction	Percentage of very satisfied customers from a customer survey data via the five point scale: very satisfied, satisfied, neutral, dissatisfied, very dissatisfied
In-process quality metrics	Phase-based defect removal pattern	Bar graph of defects removal with the index of development phases
	Defect removal effectiveness (DRE)	DRE = (Defects removed during a development phase/Defects latent in the product) × 100
	Defect density during testing	(Number of bugs/KLOC) during testing
	Defect arrival pattern during testing	Weekly Plotted cumulative defect rate during test
Maintenance quality metrics	Fix backlog	BMI (backlog management index) = (Number of problems closed during the month/Number of problem arrivals during the month) × 100
	Fix response time	Mean time of all problems from open to closed
	Percent delinquent fixes	(Number of fixes that exceeded the fix response time criteria by severity level/Total number of fixed delivered in a specified time) × 100
	Defective fixes	Number of defective fixes

(e.g., human side of quality). As strategic use of TQM increased, TQM became an aspect of the methodology of systems development. TQM (37) is defined as a structured system for satisfying internal and external customers and suppliers by integrating the business environment, continuous improvement, and breakthroughs with development, improvement, and maintenance cycles while changing organizational culture.

SYSTEMS MANAGEMENT

Today's major concern in systems engineering can be said to be the concern about systems management issues, especially for process improvement. TQM is the one of the efforts to provide the guidelines for process improvement. Another effort is business process reengineering (BPR). Although the theme of these two is the same with regard to process management, differences between TQM and BPR (38) do exist and are presented in Table 6. In addition to the TQM and BPR efforts for process improvement, the capability maturity model (CMM) by the Software Engineering Institute (SEI) at Carnegie Mellon University is an example of an alternate initiative. In this section, we will discuss BPR further, focusing on what it is and how to implement it, and subsequently discuss details of the CMM.

Business Process Reengineering

Business process reengineering (BPR) is defined by Hammer and Champy (39) as the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance. BPR needs radical change. Thus, many BPR advocates argue that an organization should think about business processes as if it is starting a new business in order to identify processes that would result in dramatic improvement. BPR often deals with the entire organization and requires a large amount of time. Davenport and Short's (40) five steps in process redesign comprise the first methodological approach: (1) Develop business vision and process objectives, (2) identify processes to be redesigned, (3) understand and measure existing processes, (4) identify information technology levers, and (5) design and build a prototype of the process. Many consulting companies have developed BPR methodologies. While these are proprietary, Grover and Malhotra (41) provide a generic reengineering methodology, which is summarized in Table 7.

Table 6. Total Quality Management Versus Business Process Reengineering

	Total Quality Management	Business Process Reengineering
Level of change	Incremental	Radical
Starting point	Existing process	Clean slate
Frequency of change	One-time/continuous	One-time
Time required	Short	Long
Participation	Bottom-up	Top-down
Typical scope	Narrow, within functions	Broad, cross-functional
Risk	Moderate	High
Primary enabler	Statistical control	Information technology
Type of change	Cultural	Cultural/structural

The success of CIGNA corporation (42)—saving more than $100 million, $2 to $3 returned benefits resulted from each $1 invested in reengineering, operating expenses reduced by 42%, cycle times improved by 100%, customer satisfaction up by 50%, and quality improvements of 75%—has increased the popularity of BPR. Most companies to date have thought about a BPR exercise. However, there are often innumerable obstacles—such as unrealistic scope and expectation, lack of management support, and resistance to change—that have caused a great number of BPR projects to fail. Such failures led to the modification of the BPR concept that focuses only on radical redesign to ensure dramatic improvement. The new term, business process change (BPC), reflects this future trend of BPR by keeping the focus on importance of process but weakening the necessity for radical change. Although it is still a question whether BPR should be incremental or radical, BPR and TQM seemed to be converging and assisting each other.

Capability Maturity Model

The capability maturity model (CMM) is concerned with the systems development process. As opposed to BPR, which is often fit to mature organizations, the CMM provides guidelines about how to manage processes depending on the maturity level of organization. The CMM is not a competitive model because it encompasses all approaches such as TQM's ISO9000 based on the level of maturity. The CMM's underlying assumption is that certain process models can perform better for certain types under certain environments. The CMM begins with defining the five capability levels, and at each level the CMM suggests (a) common features to assess the current situation of an organization and (b) key process areas to follow in order to evolve to higher levels. The key process areas (43) are presented in Table 8. At level 1—the initial level—the software process is ad hoc, even chaotic. Hence, few processes are defined due to unpredictable cost, schedule, and quality performance. Success depends on having an exceptional manager or effective software team. At level 2—the repeatable level—project management processes are established, but are so basic and variable because planning and managing new projects is based on experience with prior projects. At level 3—the defined level—the software process for both management and engineering activities is documented, standardized, and integrated. Reliable cost and schedule is achieved, but quality measure is still qualitative. At level 4—the managed level—both product and process quality are measured quantitatively. Statistical quality control is often used to manage quality. At level 5—the optimizing level—the entire organization is focused on continuous process improvement.

Process improvement is essential to reduce cost and enhance quality. The CMM's assumption that lower level of maturity should be experienced to proceed to higher level is somewhat contradictory because it might imply that organization should be in level 4 or 5 for continuous process improvement. However, many successful stories of software companies (e.g., in Refs. 44 and 45) that applied key CMM practices support the CMM's validity. In addition, in the CMM version for systems engineering (46), SEI provides classified do-

Table 7. A Generic Reengineering Methodology

	Key Activities	Types of Tools/Techniques
Preparation	Evaluate organization and environment, recognize need, set corporate and reengineering goals, identify and motivate team, train team on reengineering concepts, develop a change plan; develop project scope, components, and approximate time frames.	Planning Team building Goal seeking Motivation Change management Project management
Process-think	Model processes, model customers and suppliers, define and measure performance, define entities or “things” that require information collection, identify activities, map organization, map resources, prioritize processes.	Customer modeling Performance measurement Cycle time analysis Cost analysis Process modeling Process value analysis Value chain analysis Workflow analysis Organizational mapping Activity-based cost accounting
Creation	Understand process structure, understand process flow, identify value-adding activities, identify benchmark performance, brainstorm information technology possibilities, estimate opportunity, envision the ideal process, integrate visions, define components of visions.	Work flow analysis Process value analysis Benchmarking Cycle time analysis Brainstorming Visioning Documentation
Technical design	Examine process linkages, model entity relationships, develop performance metrics, consolidate interfaces, consolidate information, design technical systems, modularize, plan implementation.	Information engineering Work flow analysis Performance measurement Process modeling Project management
Social design	Empower customer contact personnel, identify job clusters, define jobs/teams, define skills/staffing, specify organizational structures, design transitional organization, design incentives, manage change, plan implementation.	Employee empowerment Skill matrices Team building Self-managed work teams Case managers Organizational restructuring Change management Incentive systems Project management
Implementation	Develop test and rollout plans, construct system, monitor progress, evaluate personnel, train staff, pilot new process, refine, implement full rollout, ensure continuous improvement.	Process modeling Information engineering Skill matrices Performance measurement Just-in-time training Project management

Adapted from Grover and Malhotra (41).

mains—project, engineering, and organization—depending on the responsibility of key practice areas. For example, it assigns allocation of requirements to the engineering domain, assigns quality assurance to the project domain, and provides ongoing knowledge and skills to the organization domain. This kind of specification of the role of each domain at each maturity level allows the systems engineering domain to advance to the next higher level.

OBJECT-ORIENTED PARADIGM

The object-oriented paradigm is not new if we recall the history of programming languages. Simula in the 1960s was the first language that implemented the object-oriented concept. However, the systems engineering community became interested in the object-oreiented approach only recently. One reason was the popularity of other approaches, such as traditional waterfall methodology, a structured methodology that has been successful for a good deal of time, so that systems engineering did not need object-oriented methodology. The data-oriented thinking might have prevented object thinking, and it made us consider object-oriented thinking as difficult. It is only recently that attention has been given to the new approach since systems have become larger and more complex with increasing difficulty of problem-solving.

Object-oriented system methodology has some useful characteristics that can be utilized in systems development. Eight key characteristics (25) that help in systems development are (1) common methods of organization, (2) abstraction, (3) encapsulation, (4) inheritance, (5) polymorphism, (6) message communication, (7) association, and (8) reuse. Common meth-

Table 8. The Key Process Areas in Capability Maturity Model

Level	Key Process Areas
1. Initial	No specific areas
2. Repeatable	Software configuration management
	Software quality assurance
	Software subcontract management
	Software project tracking and oversight
	Software project planning
	Requirement management
3. Defined	Peer reviews
	Intergroup coordination
	Software product engineering
	Integrated software engineering
	Training program
	Organization process definition
	Organization process focus
4. Managed	Software quality management
	Quantitative process management
5. Optimizing	Process change management
	Technology change management
	Defect prevention

Adapted from Paulk et al. (43).

ods imply information systems can be developed similar ways. The concepts such as objects, attributes, class, and message used in object-oriented methodology can be applied in the design of a system. For example, anything can be an object in the object-oriented paradigm, hence both product-oriented and process-oriented design can be carried out by simply varying the object focus. Abstraction characteristics, encapsulation, and information hiding help the system developer to concentrate more on current issues by removing unnecessary details. Inheritance characteristics have two implications: generalization or specialization. When classes have some common attributes, we can generalize them to make a superclass, whereas specialization is possible when we need subclasses. Polymorphism, which means "many forms," is an advantage that can only be implemented in the object-oriented approach. Implementing polymorphism is simple in the object-oriented approach due to class hierarchy. An instruction such as displaying the defect rate of product A and product B which are not in the same subclass is conducted by searching the procedure (code) from the lowest level to higher levels (superclasses). Message communication deals with communication between objects. For example, a customer's request to display an order number n is a message to the order object, and the order object fulfills this request by telling itself (calling its member function) to display order number n. Association is the procedure of setting relationships between objects after the identification of all objects. Well-designed associations result in well-designed processes and enhance reusability. Reusability is an object-oriented approach that is more advanced than the module subroutine used in structured methodologies, both in terms of reliability and contribution to rapid development. Under the assumption of correct implementation of other characteristics, object-oriented approaches ensure high-quality systems with less cost. Applying object concepts is a challenging job to a system developer, but once established, systems become faster, better, and cheaper and systems development becomes a routine task.

Object-oriented methodology is still in a relatively immature stage. Most of the methodologies focus on systems analysis—up to the logical design phase in the whole life-cycle phase, adding information engineering methodology at the systems design and implementation phases. Coad and Yourdon's object-oriented analysis (OOA) methodology (47) consists of a five-step procedure: (1) Define objects and classes, (2) define structures, (3) define subject areas, (4) define attributes, and (5) define services. Major tools utilized here are class and object diagram, object-state diagram, and service chart. A class and object diagram is a diagram consisting of five layers: (1) class and object layer, which shows classes and objects, (2) structures layer, which connects classes and objects with arcs to show generalization–specialization and whole-part inheritance relationships, (3) subjects layer, which groups closely related classes by adding borders, (4) attributes layer, which adds a list of attributes, and (5) service layer, which adds a list of services inside the class and object boxes and provides arcs showing message connections between boxes. An object-state diagram is a simple diagram that shows all the possible states of an object and the allowed transitions between states. A service chart is a diagram that depicts the detailed logic within an individual service, including object-state changes that trigger or result from the service.

While the above discussion is pertinent to a centralized object environment, different aspects become important in a distributed and heterogeneous environment (48). It is often the case that a company has (a) legacy systems that are costly to replace and (b) different platforms that are used depending on the task. To provide a standard for distributed objects, the object management group (OMG) (49) developed the common object request broker architecture (CORBA). Basically, CORBA follows the client–server architecture, facilitating the communication between clients and objects. The object request broker (ORB) is a software product that intercepts messages from an object, translates them for different languages or different machines in heterogeneous distributed environments, and routes them to the correct object. Since current business situations require at least some distributed processes implementation, an effort like CORBA to integrate heterogeneous systems using the object-oriented approach will support the technical side of business process reengineering.

An object-oriented approach is often considered to be parallel to the BPR movement. In other words, because BPR needs a project manager to change to process thinking, the object-oriented approach needs the system developer to change to object thinking. As of now, hundreds of object-oriented methods and tools exist. Currently there is a trend (50) to combine the advantage of BPR and the object-oriented method. However, application of this paradigm to organizations is not an easy matter. Perhaps BPR's track record of frequent failures may be the trigger to adopt object-oriented technology in a large way.

SYSTEMS ENGINEERING AND THE INTRANET

The Internet has proliferated with the advent of World Wide Web (WWW) technology. It is surprising that a system like the Internet can sustain its existence without any organized or intended controls. However, it is clear that the Internet is working as one complex system, leaving all subsystems to interact with each other. The Internet is a collection of networks not under any formal control mechanism, but the Internet as a whole contains emergent properties that happened to make the whole stable. Aside from the physical structure of the Internet that can be considered as the medium, people using the Internet are linked by means of information. The popularity of this medium has been achieved primarily because of its user-friendly interfaces like WWW and by the simplicity of open and standard protocols like Transmission Control Protocol/Internet Protocol (TCP/IP).

An Intranet is an information system within an organization based on Internet technologies such as WWW, HyperText Markup Language (HTML), TCP/IP and HyperText Transfer Protocol (HTTP). A more practical description of intranet is given by Hinrichs (51): “the Intranet is a technology that permits the organization to define itself as a whole entity, a group, a family, where everyone knows their roles, and everyone is working on the improvement and health of the organization.” Because “systems engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs (52), engineering the Intranet is definitely the subject of systems engineering, and we believe that this will be the future assignment for the information systems engineer.

Before we define the role of systems engineer in planning, developing, and deploying an Intranet, it would be better to understand the advantages and caveats of using Intranet. As is often the case in BPR projects, the bandwagon effect has resulted in failures due to the rush decision without careful analysis. Because Intranets are still growing, it is difficult to identify other dangers, but it is clear that this will be the essential tool that future organizations eventually must have whatever variation is added. Traditionally, Intranets have been used to reduce the costs of printing materials such as company newsletters, employee benefits handbooks, and training materials. The other major advantage would be enhanced speed of communication. Web-server-based communication has the ability to provide up-to-date information, and even a basic e-mail communication can ensure at most a one-day lag of communication. To address the use of Intranet, we can think of the Intranet as (1) decision-making tool via off-the-shelf information, (2) learning organization tool with faster analysis of business processes, opportunities, and goals, (3) a complete communication tool that integrates all the information into one place on the Web, (4) a collaboration tool with the form of forum, even with the use of video conferencing, electronic whiteboard, and single shared document, (5) an expert’s tool by storing sharing tips, tricks, pitfalls, and analysis about any topic in a threaded database, (6) an single invention tool with a common Web interface, (7) a process identification and process improvement tool with understanding the cross-functional information in a single place, (8) a partnering tool via the exchange of intranet information between organizations, which is now termed as Extranet, (9) a customer tool by opening information to the internet, (10) an International Organization for Standardization (ISO) tool via a singular repository which enables many of the ISO requirements, (11) a target marketing tool with the emphasis from mass market to market segment by storing two-way information flow, and (12) a human resource tool by letting employees learn new skills and access the various human resource materials online (48).

The Intranet is poised to become a candidate to make BPR initiatives successful in terms of process innovation and improvement. Intranets could help implement BPR philosophy in an autonomous way encouraged by the easiness of communication, especially in terms of process improvement. The Intranet gives employees support to think of the process in a natural way, so that resistance can be diminished. As for the modification of BPR, James (53) pointed out the problems of BPR that can be solved by Intranets and further ensure the success of BPR. Three reasons why reengineering failed and the reasons why Intranets can solve the problem are presented in Table 9.

Now comes the question about what the role of systems engineer is when building an Intranet. Clearly, the Intranet is also a system that we have already had experience in how to build. The difference will be the necessity for more detailed requirement specification and planning. Intranet is a highly complex system because of many interactions between users and information. One role for the systems engineer would be the role of physical designer for the infrastructure of the Intranet. It is true that employees can develop contents in the Intranet without difficulty, but determining bandwidth, leveraging off the existing system is the area for the systems engineers. From the strategic point of view, decisions about business plan, cost and time of development, and the objective of the Intranet should be made before the enrollment. One thing that should be considered is that the Intranet will consistently change as technology improves. Building the Intranet is not a one-time development, but an ongoing development process. Hence, a highly adaptive system, which has the ability to expand when necessary, should be the framework of the Intranet, and this framework could be obtained from prototyping approach. The most important and difficult job would be the prediction of sociotechnical structure throughout the life cycle of Intranet. However, we may not need to worry because the Internet has been successful in spite of the complex nature of system. In other words, it is natural that user-oriented development be successful. All the contents in Intra-

Table 9. Intranets and Reengineering

Reasons for Reengineering Failure	How Intranets Can Help
Top-down efforts gather little support from employee and middle managers, who tend to equate reengineering with layoff.	Intranets have typically been implemented as a result of grass-roots, bottom-up efforts.
Massive personnel retraining is usually necessary.	Intranets are an excellent vehicle for employees to use to share information on new processes and procedures, because of easy-to-use browser technology.
Projects require the participation of multiple departments, many of which have diverse and incompatible computer systems.	Web technology supports many different and diverse platforms.

Adapted from James (53).

net cannot be controlled in an organized fashion as the size increases. What we have to do is create the environment in which the users and developers assist each other and make ceaseless improvement of the Intranet. The cost of having its own Intranet is relatively low if an organization has facilities of the Internet, although the benefit is exceptional. According to IDC/Link—a subsidiary of International Data Corporation (IDC), by the year 2000 the annual shipments of Internet servers will approximate 450,000, while the shipment of Intranet servers will approximate 4,500,000. The emerging interests regarding the virtual organization is not so surprising once we understand the phenomenon that coordination cost declines notably as accessible information increases because of the Intranet. The trend toward process management or systems management by narrowing the communication gap is expected to continue.

BIBLIOGRAPHY

1. R. L. Ackoff, Toward a system of systems concepts, *Manage. Sci.,* **17**: 661–671, 1971.
2. J. O'Connor, What is a system? [Online]. Available www: http://www.radix.net/~crbnblu/assoc/oconnor/chapt1.htm
3. G. Bellinger, Systemic University on the Net (SUN) [Online]. Available http://www.outsights.com/systems
4. L. von Bertalanffy, *General System Theory; Foundations, Development, Applications,* New York: Braziller, 1969.
5. N. Wiener, *Cybernetics: or Control and Communication in the Animal and the Machine,* New York: Wiley, 1948.
6. J. W. Forrester, *Industrial Dynamics,* Boston: MIT Press, 1961.
7. F. Heylighen et al., What are cybernetics and systems science? [Online]. Available www: http://pespmcl.vub.ac.be/SYSTHEOR.html
8. W. P. Chase, *Management of System Engineering,* New York: Wiley, 1982.
9. A. P. Sage, *Systems Engineering,* New York: Wiley, 1992.
10. A. P. Sage, *Systems Management: For Information Technology and Software Engineering,* New York: Wiley, 1995.
11. IEEE P1220, *IEEE Standard for Systems Engineering,* Preliminary, 1993.
12. W. W. Royce, Managing the development of large software systems: concepts and techniques, *Proc. WESCON,* 1970, pp. 1–70.
13. B. W. Boehm, Software engineering, *IEEE Trans. Comput.,* **25**: 1126–1241, 1976.
14. U.S. Dept. of Defense, Defense System Software Development, DoD STD-2167A, June 1985.
15. B. W. Boehm, A spiral model of software development and enhancement, *IEEE Comput.,* **21** (5): 61–72, 1988.
16. M. A. Mahmood, Systems development methods—a comparative investigation, *MIS Quart.,* **11** (3): 293–311, 1987.
17. J. C. Wetherbe and N. P. Vitalari, *Systems Analysis and Design Best Practices,* 4th ed., St. Paul: West, 1994.
18. J. L. Whitten and L. D. Bentley, *Systems Analysis and Design Methods,* 4th ed., Boston: Irwin/McGraw-Hill, 1998.
19. S. D. Dewitz, *Systems Analysis and Design and the Transition to Objects,* New York: McGraw-Hill, 1996.
20. T. DeMarco, *Structured Analysis and System Specification,* New York: Yourdon Press, 1978.
21. E. Yourdon, *Modern Structured Analysis,* Englewood Cliffs, NJ: Yourdon Press, 1989.
22. J. Martin, *Information Engineering,* Vols. 1–3, Englewood Cliffs, NJ: Prentice-Hall, 1989 (Vol. 1), 1990 (Vols. 2 and 3).
23. J. Wood and D. Silver, *Joint Application Design,* New York: Wiley, 1989.
24. R. J. Norman, *Object-Oriented Systems Analysis and Design,* Englewood Cliffs, NJ: Prentice-Hall, 1996.
25. J. Martin, Timebox methodology, *Syst. Builder,* **April/May**: 22–25, 1990.
26. R. D. Hackathorn and J. Karimi, A framework for comparing information engineering methods, *MIS Quart.,* **12** (2): 203–220, 1988.
27. U.S. Dept. of Defense, Configuration management practices for systems, equipment, munitions, and computer programs (MIL-STD-483), Washington, DC, 1970.
28. ANSI/IEEE Std 828-1990, IEEE standard for software configuration management plans, New York, 1990.
29. R. W. Wolverton, The cost of developing large-scale software, *IEEE Trans. Comput.,* **C-23**: 615–636, 1974.
30. E. A. Nelson, *Management Handbook for Estimation of Computer Programming Costs,* AD-A648750, Santa Monica: Systems Development Corporation, 1966.
31. J. W. Bailey and V. R. Basili, A meta-model for software development resource expenditures, *Proc. 5th Int. Conf. Softw. Eng.,* 1981, pp. 107–116.
32. B. W. Boehm, *Software Engineering Economics,* Englewood-Cliffs, NJ: Prentice-Hall, 1981.
33. A. J. Albrecht, Measuring application development productivity, *Proc. IBM Appl. Dev. Symp.,* Monterey, CA, 1979, pp. 83–92.
34. A. P. Sage and J. D. Palmer, *Software Systems Engineering,* New York: Wiley, 1990.
35. S. H. Kan, *Metrics and Models in Software Quality Engineering,* Reading, MA: Addison-Wesley, 1995.
36. K. Ishikawa, *Guide to Quality Control,* New York: Quality Resource, 1989.
37. Integrated Quality Dynamics, TQM: Definition of total quality management [Online]. Available http://www.iqd.com/tqmdefn.htm
38. T. H. Davenport, *Process Innovation: Reengineering Work Through Information Technology,* Boston: Harvard Business School Press, 1993.
39. M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution,* London: HarperCollins, 1993.
40. T. H. Daveport and J. E. Short, The new industrial engineering: Information technology and business process redesign, *Sloan Manage. Rev.,* Summer: 11–27, 1990.
41. V. Grover and M. K. Malhotra, Business process reengineering: A tutorial on the concept, evaluation method, technology and application, *J. Oper. Manage.,* **15**: 193–213, 1997.
42. J. R. Caron, S. L. Jarvenpaa, and D. B. Stoddard, Business reengineering at CIGNA corporation: experiences and lessons learned from the first five years, *MIS Quart.,* **18** (3): 233–250, 1994.
43. M. C. Paulk et al., *Capability maturity model for software,* Version 1.1, CMU/SEI-93-TR-24, Software Engineering Institute, 1993.
44. S. Girish, Continuous process improvement—why wait till level 5?, *Proc. 29th Annu. Hawaii Int. Conf. Syst. Sci.,* 1996, pp. 681–692.
45. C. D. Buchman, Software process improvement at Allied Signal Aerospace, *Proc. 29th Annu. Hawaii Int. Conf. Syst. Sci.,* 1996, pp. 673–680.
46. R. Bate et al., *A systems engineering capability maturity model,* Version 1.1, CMU/SEI-95-MM-003, Software Engineering Institute, 1995.
47. P. Coad and E. Yourdon, *Object-Oriented Analysis,* 2nd ed., Englewood Cliffs, NJ: Prentice-Hall, 1991.

48. S. Vinoski, CORBA: Integrating diverse applications within distributed heterogeneous environments, *IEEE Commun. Mag.,* **14** (2): 46–55, 1997.
49. The Object Management Group (OMG) Homepage [Online]. Available www:http://www.omg.org
50. I. Jacobson, M. Ericsson, and A. Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology,* Workingham, UK: Addison-Wesley, 1995.
51. R. J. Hinrichs, *Intranets: What's The Bottom Line?,* Mountain View: Sun Microsystems Press, 1997.
52. INCOSE (The International Council on Systems Engineering) Homepage [Online]. Available www:http://www.incose.org/
53. G. James, Intranets rescue reengineering, *Datamation,* **42** (18): 38–45, 1996.

H. RAGHAV RAO
S. PARK
State University of New York

SYSTEMS, KNOWLEDGE-BASED. See EXPERT SYSTEMS.
SYSTEMS, LINEAR. See MULTIVARIABLE SYSTEMS.
SYSTEMS, MULTIMEDIA. See AUTHORING SYSTEMS.
SYSTEMS, MULTIVARIABLE. See MULTIVARIABLE SYSTEMS.
SYSTEMS OF POLYNOMIAL EQUATIONS. See POLYNOMIALS.