1 Systems Engineering Life Cycles: Life Cycles for Research, Development, Test, and Evaluation; Acquisition; and Planning and Marketing

F. G. PATTERSON, JR.

1.1 INTRODUCTION

In this chapter we discuss a number of process models, which we also refer to as *life cycles*. In so doing, we discuss what a life cycle is with respect to our concepts of systems engineering, systems thinking, and the systems approach. As engineers, we ask "What is a life cycle good for?" and "How does it work?" As students, we are constantly looking for fundamental principles and models with which to structure an understanding of our subject (Bruner, 1960). Thus, a study of life-cycle models is an appropriate way to begin an investigation of systems engineering.

Our understanding of life cycles is based on our understanding of systems engineering or the systems engineering approach. Johnson et al. (1967) define a system as "an organized or complex whole; an assemblage or combination of things or parts forming a complex or unitary whole ... more precisely, an array of components designed to accomplish a particular objective according to plan." An unorganized collection of parts has no purpose. As pointed out as early as in the writings of Aristotle (Ogle, 1941), it is exactly the identification of purpose with the organization of components that defines the approach that we refer to as systems engineering.

Ackoff (1981) uses the term "systems thinking" to describe a means for understanding an entity in terms of its purpose. He formulates the concept of systems thinking as three steps:

1. Identify a containing whole (system), of which the thing to be explained is a part.
2. Explain the behavior or properties of the containing whole.
3. Explain the behavior or properties of the thing to be explained in terms of its *role(s)* or *function(s)* within its containing whole.

Handbook of Systems Engineering and Management, Second Edition, Edited by Andrew P. Sage and William B. Rouse

Strategic planning is purposeful, goal oriented, and goal setting. Systems engineering is the analysis and synthesis of parts, with an eye toward the goal set by strategic planning. Focusing on the problem is *analysis*, or taking the problem apart to identify and understand the pieces with which to do synthesis.

When we apply Ackoff's formulation of the systems approach to explain systems engineering life cycles, we first identify the containing whole, and we find that life cycles are attributes of processes, processes are elements of enterprises, and enterprises are actions of organizations. A life cycle is an application of the systems approach for the purpose of understanding and implementing processes. Every process has a life cycle. A *process* is a collection of one or more actions whose purpose is to accomplish a goal. In a systems engineering organization, this goal contributes to the fulfillment of a strategic plan. If we define a process very generally as a strict partial ordering in time of the strategically important activities of the enterprise, then we can regard a set of processes that includes all such activities of the enterprise in some process as a partitioning of the enterprise. To be interesting, a process should have at least one input and at least one output, and each activity in a process containing more than one activity should either depend on or be dependent on some other activity in the same process.

A process is an element of an enterprise, and an enterprise is itself a process. Every enterprise (process) may be approached as a system (Thome, 1993) and has a life cycle. Using a basic enterprise model due to Sage (1995), a process will be one of three basic types: system acquisition or production; research, development, test, and evaluation (RDT&E); or planning and marketing.

For a number of reasons, many of which relate to the systems engineering organization, it is useful to regard a process as an ordered collection of phases that, when taken together, produce a desired result. We refer to this ordered collection of phases as a *process life cycle*. Moreover, it is possible and desirable to identify life cycles for each of the three basic classes of processes. Thus, we may speak of a system production or system acquisition life cycle, an RDT&E life cycle, and a planning and marketing life cycle.

An enterprise is a collection of one or more processes undertaken by an organization to achieve a goal. An organization may have one or more enterprises. Every enterprise also has an organization, whose purpose is closely tied to the enterprise. Indeed, from the standpoint of purpose, the enterprise and the organization are identical, since each is an element of the other. Thus, to understand the purpose of the systems engineering life cycle, it is reasonable to consider it part of an organization and to examine it in that role.

Life cycles are both descriptive and prescriptive. In our study of systems engineering life cycles, it is important to abstract and examine life-cycle models not only for their simplicity for describing processes, but also for their *prescriptive* applicability to the organization of an enterprise. Bass and Stogdill (1990), in a major work on leadership, attribute to Deming the idea that managers need to know whether the organizational system in which they are involved is stable and predictable. A well-defined systems engineering life-cycle model that is imposed on the system from the beginning can give the maturity derived from lessons learned in using the time-tested model, thus adding an important element of stability to the organizational structure. Bass and Stogdill (1990)

also cite the need for strong leadership in an organization's early, formative period.[1] This can be related to the idea of the relatively strong need for a predefined systems engineering life cycle in early, immature stages of the organization, where "strong need" equates to strong leadership requirements. Leadership and structure can help an organization to avoid many problems, by eliminating several sources of conflict. For example, Bradford et al. (1980) enumerate three of the most common group problems, each of which may be reduced by strong leadership and well-defined structure:

1. Conflict or fight
2. Apathy and nonparticipation
3. Inadequate decision making

Burns (1978) associates both power and leadership with purpose:

> Leadership over human beings is exercised when persons with certain motives and purposes mobilize, in competition or conflict with others, institutional, political, psychological, and other resources so as to arouse, engage, and satisfy the motives of followers. This is done in order to realize goals mutually held by *both* leaders and followers.

Stogdill (1966), describing the classic view of the relationship of purpose to organizational structure, writes: "One of the most stable and enduring characteristics of organization, purpose serves as a criterion or anchorage in terms of which a structure of positions is differentiated and a program of operations is designed." Stogdill models the organization as an interbehavioral system in three dimensions:[2] interpersonnel, structure, and operations, shown in Figure 1.1.

The *operations* dimension, the third dimension of Stogdill's model, is concerned with processes, including business processes that both describe and determine the enterprise activities of the organization. In introducing this third axis, Stogdill notes that *the operations side of the model will determine to a large degree the structural and interpersonnel aspects of his model*. Thus, it may be argued that operations should lead, rather than follow, the elaboration of the other dimensions. We may introduce the systems approach that views a process as a synthesis of organizational elements to achieve a purpose. By introducing processes, as represented by their life-cycle models, we can purposefully influence the formation of the organization in all of its dimensions.

Stogdill points out that classic organizational theory is concerned mostly with the subdivision of work and with the differentiation of responsibility and authority. It constitutes an empirical and logical analysis of these aspects of organizations that has proved to be highly effective as a template or set of principles for structuring new

[1]Bass and Stogdill (1990) write: "Hersey and Blanchard's life cycle theory of leadership synthesizes Blake and Mouton's managerial grid, Reddin's 3-D effectiveness typology, and Argyris's maturity–immaturity theory. According to this theory, the leader's behavior is related to the maturity of the subordinates. As the subordinates mature, the leader's behavior should be characterized by a decreasing emphasis on task structuring and an increasing emphasis on consideration. As the subordinates continue to mature, there should be an eventual decrease in consideration. Maturity is defined in terms of subordinates' experience, motivation to achieve, and willingness and ability to accept responsibility."

[2]This is an example of *morphological analysis*, which, according to Hall (1977b), refers to the decomposition of a problem into its orthogonal basic variables, each becoming a dimension of a morphological box.

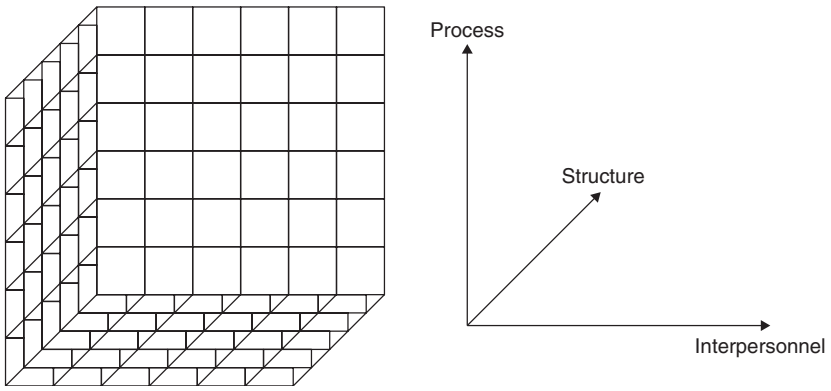

Figure 1.1 An organizational model by Stogdill.

organizations. With adequate domain knowledge, including knowledge of the nature, technology, size, and complexity of the tasks to be achieved, it is possible to create positions and to specify their structure, function, and status interrelationships; design communication channels; and chart the flow of operations (processes) necessary to perform the tasks. Training of existing personnel or hiring of new personnel to fill the positions is accomplished as a next step.

Life cycles are tools of management that allow the organization of enterprise activities around a purpose. When an organization undertakes an enterprise, members need to structure themselves according to their goals. Organizations establish themselves around processes according to some partition of the enterprise.[3] Their strategic plan is much enhanced by introducing a life-cycle model with proven success for the domain. An ad hoc organization will occur—this is the message of Stogdill—if not templated, especially if the organization is involved in an enterprise with which it is unfamiliar. Introducing the template saves much time and helps to assure success. Galbraith and Nathanson (1978) carry on the tradition of Chandler (1962) with their claim that "effective financial performance is obtained by the achievement of congruence between strategy, structure, processes, rewards, and people." Thus, in terms of Stogdill's model, the development of the operations side of the model can be enhanced by injecting a life cycle (or set of process life cycles) with proven effect in every aspect of the organization.

Now that we have identified the systems engineering organization as the whole that embodies corporate purpose and have identified many of the dynamic forces within the organization, we can continue our application of systems thinking by focusing on overall behavior. Life cycles model organizational behavior. Behavior is characterized by products, which may be organized into phases for manageability. Each phase is usually characterized by one or more major products emerging from the organization during that phase. In Figure 1.2, three classic views of the system life cycle are depicted. At the lowest level of management, each phase of the life cycle terminates with completion of one or more major products, shown as blackened rectangles. Many

[3]This is a basic assumption of business process reengineering, a procedure that involves examining, combining, replacing, or adding enterprise activities as necessary to repartition the organization into processes in a way that is more profitable. Reorganization during reengineering follows repartitioning of enterprise tasks.

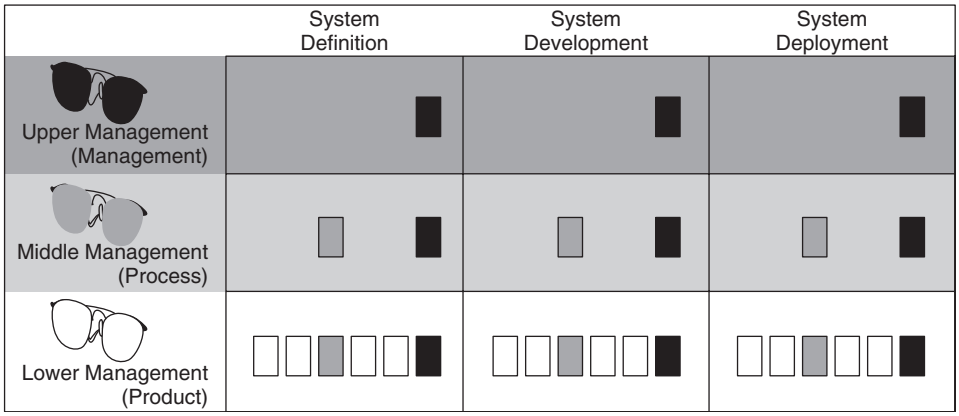

Figure 1.2 Management views of life-cycle products by life-cycle phases.

intermediate products may be identified that are important to product-level management, not because they are marketable products, but because they represent components of the finished product, or checkpoints, that are associated with progress and productivity, sometimes referred to as earned value. The intermediate products are also shown as visible to the lowest level of management. This level of the organization is responsible for ensuring the quality of the product through product inspection, measurement of partial products at checkpointed intervals, and other means that require all product details to be visible.

Middle management typically has much less responsibility, visibility, and cognizance relative to intermediate products than product-level management. In general, the focus of middle management is on process rather than product. This is depicted in Figure 1.2 by reducing the number of intermediate products shown to middle management to those deemed to be important for assessing process quality, and through this product quality, shown as rectangles shaded in gray. Thus, middle management is concerned with integrating product-level resources into a high-quality process. At the upper management level, the concern is with integrating process to achieve an organizational goal, a strategic purpose. Thus, upper management requires even less visibility into intermediate products, perhaps none at all. Instead, the focus is on the coordination and integration of production and acquisition, RDT&E, and planning and marketing life cycles.

1.2 CLASSIFICATION OF ORGANIZATIONAL PROCESSES

Drucker (1974) references several traditional methods of classifying and grouping activities in organizations. From a systems engineering point of view, most are analytic, in that they tend to direct attention inward to the skills required to accomplish tasks by grouping like skills together. An outward-looking, synthetic approach, that Drucker clearly finds to be more useful, results from grouping activities by their type of contribution to the organization. He defines four major groups:

1. Result-producing activities
2. Support activities

3. Hygiene and housekeeping activities
4. Top-management activity

Of Drucker's four categories, only *result-producing activities* have nontrivial life cycles. The other three types are ongoing, organic elements of the organization, whose activities are necessary, but whose efficacy cannot be measured by their contribution to corporate revenue.

Drucker identifies three subclassifications of result-producing activities:

(a) Those that directly generate revenue
(b) Those that do not directly generate revenue but are still fundamentally related to the results of the enterprise
(c) Information activities that feed the corporate appetite for innovation

Each one of Drucker's categories corresponds to a basic systems engineering process type identified by Sage (1992a), each with a life cycle that we will examine in detail. Respectively, we will discuss the following:

- The planning and marketing life cycle
- The acquisition or production life cycle
- The RDT&E life cycle

With respect to each life-cycle type, it is necessary to recognize, analyze, and synthesize a response to the market, which may be external or internal to the organization.

For example, one high-level representation due to Sage (1992a) of a generic systems engineering life cycle is a three-phase model: definition, development, and deployment (DDD). Each phase can be further described using recognize, analyze, and synthesize (RAS) to categorize activities contained in the phase. Activities and products can be organized in a matrix representation as suggested by Figure 1.3.

The DDD model is itself a prime example of RAS. System definition is essentially a recognition (R) activity during which requirements are recognized; development is an analysis (A) activity in which requirements, and alternatives for their realization, are considered, analyzed, and developed; and deployment is a synthesis (S) step, in that the results of the development phase are purposefully integrated into an operational system. RAS is *recursive* in that each phase can be considered to be a stand-alone process, with inputs and outputs. Each phase may then be further analyzed using RAS. For example, the definition phase of a generic acquisition process

	recognize	analyze	synthesize
definition			
development			
deployment			

Figure 1.3 Abstract matrix representation of a generic process.

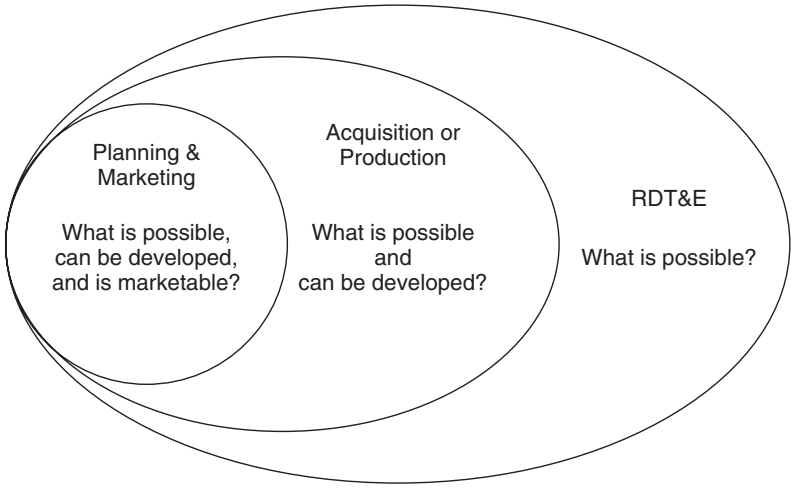

Figure 1.4 Narrowing the universe of possible products.

recognizes stakeholder and environmental needs and other constraints using such tools as requirements elicitation and classification. The analysis subphase analyzes and, as necessary, prototypes requirements to develop a requirements list. The synthesis subphase transforms the finished product of the analysis subphase into a specification. We can recursively apply RAS to "requirements elicitation and classification," since it is a process with an input that needs to be recognized and an output that must be synthesized based on an analysis step. The DDD model can also be referred to as *formulate, analyze*, and *interpret* (FAI) (Sage, 1992a). DDD, FAI, and RAS are congruent to one another.

The three basic systems engineering life cycles of Sage can be seen as three filters through which a product concept must pass to be deployed successfully by an organization (Fig. 1.4). Each successive filter subsets the universe of possibilities based on both internal and external factors.

In the RDT&E cycle, the feasibility of ideas is measured by the standards of existing technology, constrained only by the goals of the organization. These goals are normally more liberating than limiting, since the organization that is focused on a goal will in general be better endowed, in terms of its technology and its organization, and thus better positioned than competitors to create successful products. From the standpoint of the acquisition or production life cycle, the view is inward toward the capabilities of the organization. Finally, from the viewpoint of the planning and marketing cycle, the direction of focus is outward to the external market. We may note that these three successive filters have, in terms of their direction and focus, the same characteristics as the three parts of our RAS framework: "recognize" looks at possibilities within a goal orientation; "analyze" looks inward and considers what resources are available; and "synthesize" looks outward, answering patterns of need from the market in terms of feasible alternatives based on organizational capabilities.

We now look at three classes of systems engineering life cycles: the RDT&E life cycle; the acquisition, or production, life cycle; and the planning and marketing life cycle.

1.3 RESEARCH, DEVELOPMENT, TEST, AND EVALUATION LIFE CYCLES

We represent RDT&E as a separate life cycle from system acquisition and planning and marketing, since the staff, activities, goals, and constraints of research projects are sufficiently different to require special treatment. Research by its nature is proactive. Whether it is independent or directed, research is based on an innovative idea, the worth of which is to be tested by the RDT&E organization and evaluated by the planning and marketing organization. The innovation may be the result of independent, or basic, research, whose work may result in either a product or a process innovation. Alternately, in the case of directed research, often referred to as *commercial development*, the innovation can be found in terms of a poorly understood requirement for a system under development. In this case, prototype development may help articulate the requirement, which can be returned to the acquisition organization for development or to the RDT&E organization for further directed research. Poorly understood requirements may also originate in the marketing organization, who, having perceived a market need, subsequently wish to explore possibilities with the research organization.

RDT&E is often viewed as unnecessary overhead by poor management who, when faced with market pressures, reduce RDT&E expenditures in an effort to reduce cost through eliminating a probable loss. A well-managed RDT&E program is better regarded as a tool for risk mitigation, the use of which probably results in profit (Cooper, 1992). Risk can be identified for each fundamental element of a system: management, business processes, and products. Research is fundamentally either independent or directed. Within these categories, we can systematically examine areas of risk and corresponding techniques of addressing risk through RDT&E. Figure 1.5 depicts some of the relationships.

RDT&E provides a framework within which to manage research and development. The concept of an RDT&E life cycle can be defined abstractly. For example, we can postulate three major phases: definition, development, and deployment. These phases exist apart from the acquisition life cycle, however, and create inputs to it: proactively in the case of independent research, and interactively in the case of directed research.[4] As

I	independent research	directed research
management	obsolescence by new mngt. methods obsolescence by new work force norms obsolescence by new process requirements	poor understanding of market requirements poor understanding of organization poor understanding of business processes
process	obsolescence by new methodologies obsolescence by new tools obsolescence by new product requirements	poor understanding of models poor understanding of new methods poor understanding of new tools
product	obsolescence by new technology obsolescence by changing demand obsolescence by new market requirements	poor understanding of requirements poor understanding of design poor understanding of environment

Figure 1.5 A classification of problems addressed by RDT&E.

[4]For completeness we can speak of reactive research, such as failure analysis.

suggested by Figure 1.5, independent research and directed research mitigate different classes of risk. While directed research solves short-term problems and yields answers on demand, independent research deals with the long-term problems caused, ironically, by success, and attendant complacence, inertia, and lack of innovation in a competitive market. From the standpoint of life cycles, these two types of research are different, in that independent research can properly be said to have a life cycle of its own with definition, development, and deployment phases; but directed research "borrows" the life cycle of the process that it serves.

Pearce and Robinson (1985) describe four basic decision areas for research and development (R&D):

1. Basic research versus commercial development
   - To what extent should innovation and breakthrough research be emphasized? In relation to the emphasis on product development, refinement, and modification?
   - What new projects are necessary to support growth?
2. Time horizon
   - Is the emphasis short term or long term?
   - Which orientation best supports the business strategy? Marketing and production strategy?
3. Organizational fit
   - Should R&D be done in-house or contracted out?
   - Should it be centralized or decentralized?
   - What should be the relationship between the R&D unit(s) and product managers? Marketing managers? Production managers?
4. Basic R&D posture
   - Should the firm maintain an offensive posture, seeking to lead innovation and development in the industry?
   - Should the firm adopt a defensive posture, responding quickly to competitors' developments?

In terms of the Pearce–Robinson decision areas, both basic (independent) research and commercial development (directed research) are systems engineering processes. Each is a systems process in that it employs systems thinking to synthesize the goals of the organization and the requirements of the marketplace. Basic research is an engineering process in that it solves the general problem of generating a corporate response to the market at the levels of management, process, and product. The inputs to its definition phase may be provided by strategic planning. These inputs may be general, necessitating the search of a broad solution space, or very specific. Often the outputs from the definition phase are poorly understood and require further definition. While it is true that the refinement of requirements is an active task throughout all systems engineering life cycles, the RDT&E life cycle is especially tolerant of imprecision and lack of clarity in early stages of research.

Figure 1.6 depicts an RDT&E process model, organized in terms of the familiar three-phase systems engineering life cycle. In the definition phase, Melcher and Kerzner (1988) differentiate basic research as either well-defined or non-well-defined. An example of well-defined research is *defensive* research, undertaken to protect the

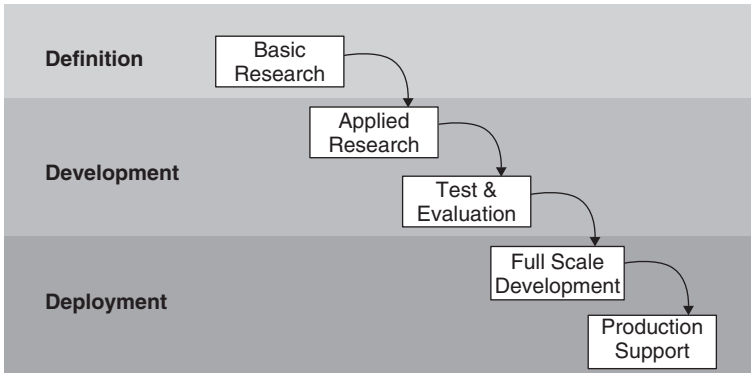

Figure 1.6 A life-cycle model for RDT&E.

organization's market position from market competition. Non-well-defined research is more likely to result in product diversification.

During the development phase, in which an actual product is designed and built, the constraining influences of organizational goals may be felt and perhaps measured in terms of the capacity and necessity for corporate change that the potential new product represents. In addition to new insights about the product, the prospect of realigning business processes to accommodate new production may provide both positive and negative insights needed by strategic planners to guide future research efforts. Product-level insights cause iteration on the requirements phase until an acceptable product is defined and built. In a very real sense, the development phase might be regarded as the prototyping element of the requirements phase. At the end of the development phase, requirements will be much more stable than they were at the end of the definition phase, but still not immutable. The purpose of the development phase is to provide support for a decision on whether the potential product is feasible and desirable.

Test and evaluation of the product provide the content of the deployment phase of the RDT&E life cycle. The goal of this phase is to deploy a useful model of a potential product for the consideration of management. The model provides much more information than would requirements alone about the impact potential of the product upon the organization in terms of start-up costs, perturbation of existing functions, and applicability of existing assets. Testing in this phase may take many forms, including product testing; product validation; and review by management, marketing, and process elements of the organization. This "internal marketing" is essential to assess the level of organizational buy-in, which is an indirect measure of consistency with strategic goals.

The output of the RDT&E life cycle may be viewed as input to the acquisition life cycle. Successful candidates from the RDT&E life cycle are moved to the requirements phase of the acquisition life cycle. While it is possible that, because a candidate has been chosen for development, requirements are complete, it may not be assumed. It is more often the case that "life-cycle issues," such as maintainability, safety, and reliability, have not been considered adequately and must be added in the definition phase of the acquisition life cycle. RDT&E provides a proof of concept that is generally not of production quality.

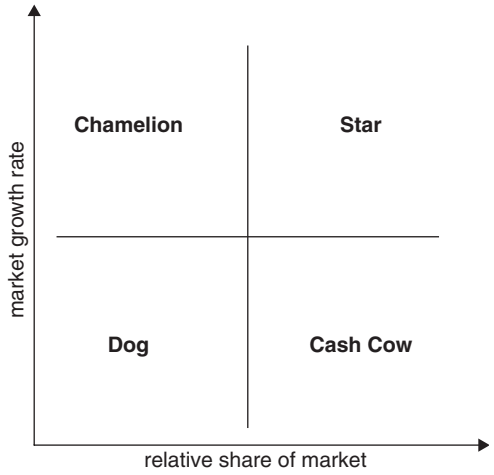

Figure 1.7 Market share versus market growth rate.

The planning and marketing cycle influences and thus provides input to the RDT&E process. Marketing may evaluate an existing product or line of products and services according to their market share and their potential for growth. Figure 1.7 catalogs classic market share and market growth rate concepts (Sage, 1995).

An organization should understand its position in the market to properly coordinate its RDT&E efforts. According to the dynamics of the market, management, through the planning and marketing process, may pursue any of three mathematically possible strategies: (1) increase market share, (2) maintain market share, or (3) reduce market share. We examine each in turn.

1. Pursuing an increase in market share is a cost-effective strategy in the case of a selected "star" or a carefully selected "chameleon" (Fig. 1.7). The RDT&E process will not be seeking a new product, but rather a way to save costs (especially in the case of a star) or improve quality (especially in the case of a chameleon). Solutions may be found in improvements to either process or product. Cash cows and dogs characteristically have low potential for growth. However, a dramatic product improvement or a new product based on either may be provided by RDT&E.
2. Maintaining market share is an appropriate strategy in the case of a selected star or a cash cow. In view of the relatively low potential for market share, a relatively defensive RDT&E strategy may be necessary. Pressure from competing products, such as product innovations, pricing pressure, or product availability, may be countered through product innovations from RDT&E that match or exceed competitive innovation or through process improvements that lower cost and allow defensive pricing without significant loss of profitability. RDT&E may also innovate through new applications of the product.
3. Reducing market share is appropriate for dogs, where profitability is modest or negative. A decision to remain in the market could be based on an expected breakthrough from RDT&E, with respect to either product or process, that would improve quality or lower the cost of production.

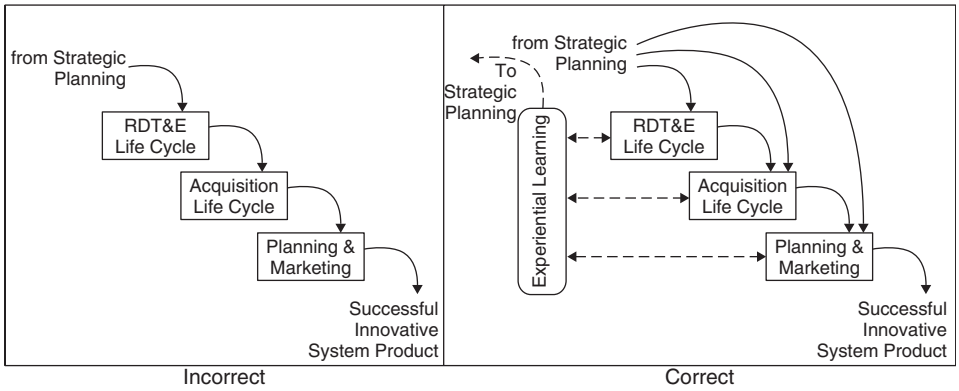

Figure 1.8 The wrong and right ways to picture life-cycle interrelationships (Sage, 1995).

The output of the RDT&E life cycle may also be viewed as input to the planning and marketing life cycle. Successful candidates from the RDT&E life cycle provide a principal input for the marketing function, in which a marketing plan is developed based both on market research and on the characteristics of the product prototype. Unsuccessful candidates from the RDT&E life cycle are also valuable, although their value is not directly measurable. Virtually all RDT&E activities contribute valuable data to the corporate knowledge base. In recognition of this contribution, Sage (1995) depicts both a right and a wrong view of the interrelationships among RDT&E, acquisition, and planning and marketing (Fig. 1.8).

1.4 SYSTEM ACQUISITION OR PRODUCTION LIFE CYCLES

A large number of life cycles have been proposed for systems production or acquisition. For example, Cleland and King (1983) give a five-step USAF Systems Development Life Cycle consisting of the following phases:

1. Conceptual phase
2. Validation
3. Full-scale development
4. Production
5. Deployment

King and Cleland (1983) also describe a slightly different five-phase life cycle consisting of the following:

1. Conceptual phase
2. Definition phase
3. Production or acquisition phase
4. Operational phase
5. Divestment phase

In addition, both the U.S. Department of Defense (DoD) and the National Aeronautics and Space Administration (NASA) (Shishko, 1995) have multiphased acquisition life cycles that have evolved over time and incorporated many expensive lessons. A comparison of the DoD and NASA acquisition life cycles to each other and to a generic six-phase commercial process model has been constructed by Grady (1993). There are various other names for the acquisition process model: production cycle, procurement process, project cycle,[5] and implementation process are often used interchangeably. In this discussion, we refer to all of these as *acquisition*. The basic shape of the acquisition life cycle follows the basic definition–development–deployment shape that is the basis of many action models. During the definition phase, we create and generally prefer to distinguish between its two basic products: requirements and specifications. The requirements for a product or service form a statement of need that is expressed by a person or group of persons who will be affected, either positively or negatively, by the product or service. Hall (1977a) refers to requirements as *the problem definition:*

> One may start with problem definition, the logical conclusion of which is the setting of definite objectives. Objectives function as the logical basis for synthesis. They also serve to indicate the types of analyses required of the alternate systems synthesized, and finally they provide the criteria for selecting the optimum system.

Requirements characteristically suffer from the ambiguity and imprecision usually attributed to natural language. Nevertheless, the requirements are the most accurate representation of actual need and should be treated as the standard by which the resulting product or service will be judged or measured. Specifications are engineering products and mark the end, rather than the beginning, of the requirements life cycle, a subset of the acquisition life cycle. This is a valid life cycle.[6] It sets forth steps in a process of eliciting requirements and then transforming them into a system requirements specification. The transition points between any two successive phases are well defined and represent a change in the activities of the requirements engineer and usually a change in the product as well. We may identify five phases in this life cycle, represented graphically in Figure 1.9:

1. Elicitation of requirements
2. Classification
3. Analysis
4. Prototyping
5. Requirements documentation and specification

An initial requirements list can be generated using a team approach or, for smaller endeavors, by interviewing stakeholders and discussing their needs. In either case,

[5]Kerzner (1992) notes that "there is no agreement among industries, or even companies within the same industry, about the life-cycle phases of a project. This is understandable because of the complex nature and diversity of projects."

[6]Although a life cycle for requirements is given here with a discrete beginning and end and steps in between, we must keep in mind that requirements issues are persistent and arise throughout the development and maintenance of the product.

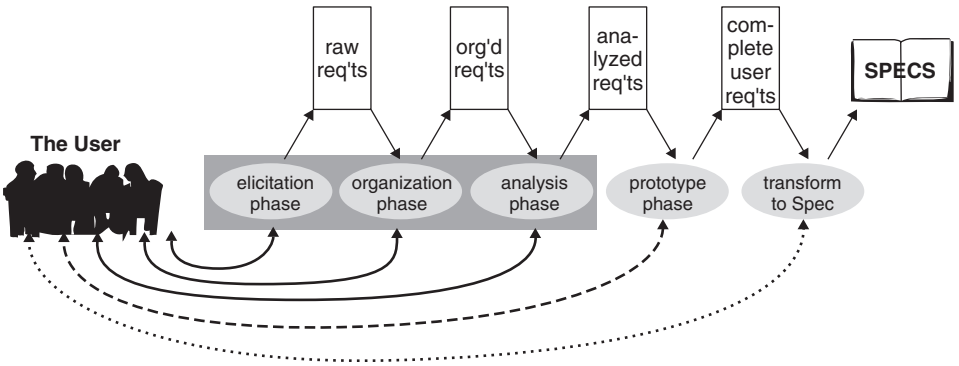

Figure 1.9 A requirements life cycle.

the starting point of the requirements engineering process is an elicitation process that involves a number of people to ensure consideration of a broad scope of potential ideas and candidate problems (Hill and Warfield, 1977). During requirements elicitation the following questions must be addressed:

- Who are potential end users? What do they do?
- What is currently available? What is new? What is old?
- What are the boundary conditions? (This is determined by the customer and governs the range of elicitation.)
- What work does the developer need to do?

The second step is the organization of the elicited requirements. In this step there is no transformation of the requirements, but simple classification and categorization. For example, requirements may be grouped into functional versus nonfunctional requirements.

The third step is analysis of the requirements. This represents a transformation: a *transformation* occurs if the requirements engineer makes any change that did not come from the stakeholder. For example, the adding of detail is a transformation. During this step, a number of checks are typically performed, such as to enable determination of the following:

Ambiguity	Performance
Completeness	Redundancy
Conflict or consistency	Reusability
Correctness	Testability
Orthogonality	Trade-off analyses

It is worth noting that if any of these analyses are done at all, they are usually done manually. NASA (Radley and Wetherholt, 1996) has cataloged a number of significant manual requirements analysis methods that are related to software safety.

The fourth step is the development of a prototype, a synthesis step in which some part of the problem is developed to some level of completion. In this way, poorly

understood requirements may be tested and perhaps strengthened, corrected, or refined. This activity is often done as a proof of concept and serves to induce feedback from both the stakeholders and engineers.

The fifth step represents the requirements as the finished product of the stakeholder requirements team. The requirements are compiled into a requirements list or into some equivalent document format. These collected requirements are then transformed into a specification.

The transformation from a requirements list to a specification represents a significant risk in the requirements engineering process. In this operation the requirements that were elicited from the user and refined through analysis, synthesis, prototyping, and feedback are transformed into a requirements specification. It is a transition of form (according to some prescribed document standard), substance (through addition of details necessary to support successful implementation), and language (through the use of rigid requirements language and nonnatural language representations, such as state charts and various mathematical notations).

The system specification is the engineering definition of the product to be developed or acquired. In the acquisition phase the specification is partitioned, enlarged, and refined in accordance with the detailed life cycle that is used. Typically, the specification is decomposed in the way that best utilizes the resources of the development organization, unless specific guidance for decomposition is provided in the requirements definition. For systems that are primarily composed of hardware, the benefits in terms of cost and quality of allowing the development organization a high degree of freedom in interpreting the specification are passed along to the customer. Software systems differ in that the product architecture must support the evolutionary nature of software maintenance, the part of the software life cycle that predominates in terms of both duration and cost. One of the principal advantages of in-house software development, as opposed to acquisition of either custom or commercial off-the-shelf (COTS) software, is that, because the development organization and the maintenance organization are the same, product cost and quality dividends are effectively realized twice, once during development and again during the maintenance phase, over a much longer time period.

There are several additional interesting points to be noticed about the requirements life cycle as depicted in Figure 1.9. First, we refer to the definition phase of the systems engineering life cycle as containing the requirements life cycle. That the requirements life cycle is a "proper life cycle" may be seen by noting that each of the steps is well defined, each with a product that characterizes the step. Moreover, the steps may be grouped into the familiar definition–development–deployment paradigm. Requirements elicitation defines the requirements. Classification, analysis, and prototyping develop the requirements, which are then deployed as a requirements list and transformed into a requirements specification. None of this is to say, however, that activities that have been ascribed to one phase or step, rather than to another, may not be performed or revisited in a subsequent phase or step. Decision making through problem solving is the essential nature of engineering (Hazelrigg, 1996); the more we know about a problem, the more we may define, develop, and deploy. Thus, the partitioning of activities into phases, and the resultant linear ordering of activities, is done to facilitate understanding, but certainly not to levy boundary conditions on phased process activities. Second, we may note that the more transformations that are performed on the elicited requirements, the less communication is possible with the

stakeholders (represented as "users" in Fig. 1.9). This constraining influence represents risk, in that the stakeholders may be unaware until a failure of possible misapplication of requirements occurs. Figure 1.9 depicts lines of communication with the stakeholders using solid and broken lines, where solid lines represent relatively good communication, and increasingly broken lines represent increasingly poor communication. Third, we may note that Figure 1.9 associates the first three steps, frames them together, and shows solid lines of communications to the stakeholders. This suggests that these activities may all be done together as a stakeholder team activity, thus further investing the stakeholders in the process of developing high-quality requirements. Because of the great potential benefits associated with keeping the stakeholder team involved, it is important to discipline the requirements engineer to use the language (terminology, examples, associations among requirements, etc.) of the stakeholder and to refrain from introducing engineering models, templates, paradigms, and tools during stakeholder involvement.

The definition phase of the system acquisition or production life cycle contains, in addition to requirements definition, three other possible elements, each of which creates a link to other processes in the enterprise:

1. Preparation of acquisition documents
2. Linkage with RDT&E life cycle
3. Linkage with planning and marketing life cycle

Acquisition documents create a link within the acquisition life cycle between the requirements definition process and the development of the product. Jackson (1995) describes specifications as the interface between the problem space (requirements) and the solution space (implementation). In the case of in-house development, the requirements specification is all that is required. In other cases, however, where two or more different organizations must cooperate, formal structure may include a request for proposal and a proposal containing a statement of work, a systems engineering management plan, a work breakdown structure, and a negotiated list of other deliverables.

The linkage with the RDT&E process may be of two basic types. If the research was directed by the requirements development process either to clarify requirements or to solve a technical problem, then the research results are needed to complete the requirements life cycle. Or, if the research product was the result of independent and innovative work of an on going nature through which the organization creates new products, then the product definition must be handed off to the requirements team to allow the product to be brought to market.

Finally, requirements from planning and marketing may influence product definition, especially to incorporate plans for and the results of market research. Input from the planning and marketing process is critical to setting and maintaining cost goals. Michaels and Wood (1989) discuss "design to cost." The basic idea of "design to cost" is that, through the analysis of design alternatives (additional analysis that can add 20% to design costs), trade-offs of functionality for affordability, and allocation of cost goals to the work breakdown structure, and through dedication and discipline on the part of management, cost goals can be built into a design-to-cost plan, and the plan can be carried out successfully. Marketing studies that provide projections

of sales, selling price, and sales life cycle[7] allow such cost goals to be set intelligently. Output from the acquisition process to the planning and marketing process allows early product information to be used to create marketing plans, tools, and schedules.

The second phase of the system acquisition life cycle is the development phase. A great deal of work has been done in this area, and there are many life-cycle models. This phase begins with the system specification and ends with the delivery of the developed system. Basically, a divide-and-conquer approach is almost universally employed. In this, the system architecture represented in the system specification is divided and subdivided into components that are individually specified, designed, built, tested, and integrated to compose the desired system. We can say that systems are designed from the top down, then implemented from the bottom up. Ould (1990) represents this approach as the "V" process model for software engineering, but which clearly applies more generally to systems. It is sometimes argued that new software requirements may be introduced throughout development; hence, the need for a flexible process model. As noted earlier, however, every stage of the development process is itself a process that is amenable to an RAS approach that recognizes and satisfies needs repeatedly. The conclusion that the software engineering process model is a rediscovery and alternate description of the systems engineering process model seems inescapable.

Figure 1.10 depicts the V as a composition of three layers or views of the system in increasing engineering detail. The top view is the *user model*, which associates system requirements with their realization as a delivered system. The user model is the perspective of the customer or stakeholder, who is interested in submitting a list of requirements and receiving a finished product that meets the requirements. The second or middle layer of detail is the *architectural model*, which addresses the decomposition of the system-level specification into system design and subsystem specifications and designs; paired together with built and tested subsystems, and finally the tested system. The architectural model is the perspective of the systems engineer, who is interested in decomposing the whole into manageable parts, respecifying and designing the parts, and integrating the parts to compose the finished system. The third and lowest level is the *implementation model*, which couples component specifications and designs with fully tested components. The implementation model is the perspective of the contractor, who is interested in component-level specifications, designs, and products. The subtlety of these models lies in the role of the systems engineer, whose activities must encompass three fundamental activities: recognizing the product or component as a system, analyzing the system requirements, and synthesizing the system components. These three activities—recognize, analyze, synthesize—are the necessary parts of the systems engineering process, not only in development but also in definition and deployment. For example, the reengineerability of a software system can be shown to be directly related to the ability of the system to be recognized, analyzed, and synthesized; and metrics may be ascribed to each of these dimensions, thus providing indirect measures of reengineerability (Patterson, 1995a).

Ould (1990) also describes a variation of the V process model known as the VP, where P refers to prototyping. When requirements are not well understood for system

[7]The sales life cycle refers to a set of phases through which the product moves after it has been placed into the market. For example, a four-phase sales cycle contains (1) establishment, (2) growth, (3) maturation, and (4) declining sales phases (King and Cleland, 1983).

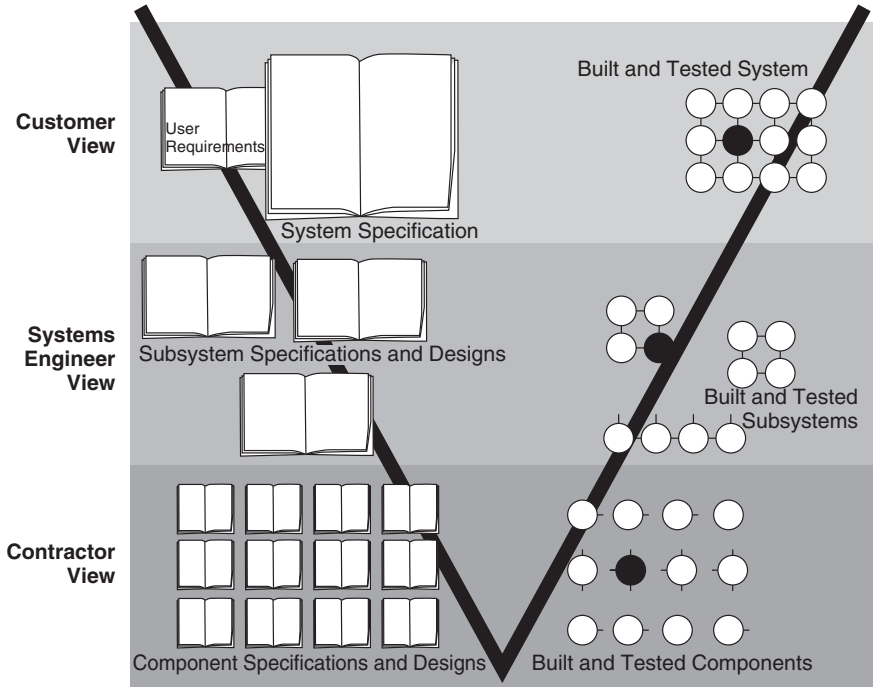

Figure 1.10 Acquisition process life-cycle V.

elements, prototypes may be developed to provide insights and answers to problems and questions. The use of a prototype may be viewed as a form of directed research and development, just as in the definition phase.

Both verification and validation (V&V) can be visualized using the V process model (Whytock, 1993). Verification, a form of document tree analysis that ascertains that every requirement in a higher level specification is decomposed successfully into requirements at a lower level in the tree (and vice versa), follows the shape of the V. It is essentially a vertical concern, as depicted in Figure 1.11. According to ANSI/IEEE Standard 729, software verification is "the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase" (American National Standards Institute/Institute of Electrical and Electronics Engineers (ANSI/IEEE), 1983; Boehm, 1990a). The horizontal dimension asks a different question: namely, whether the built and tested product at some level of completion successfully represents the specifications and designs *at the same level of the model*. This is validation. Again according to the ANSI/IEEE standard definition, software validation is "the process of evaluating software at the end of its software development process to ensure compliance with software requirements" (ANSI/IEEE, 1983; Boehm, 1990a). These software-specific definitions may be, and often are, generalized to systems. As shown in Figure 1.11, considerations of risk often result in a departure in practice from these definitions. In both the case of verification and the case of validation, the cost of detecting and correcting problems later, rather than earlier, in the life cycle is

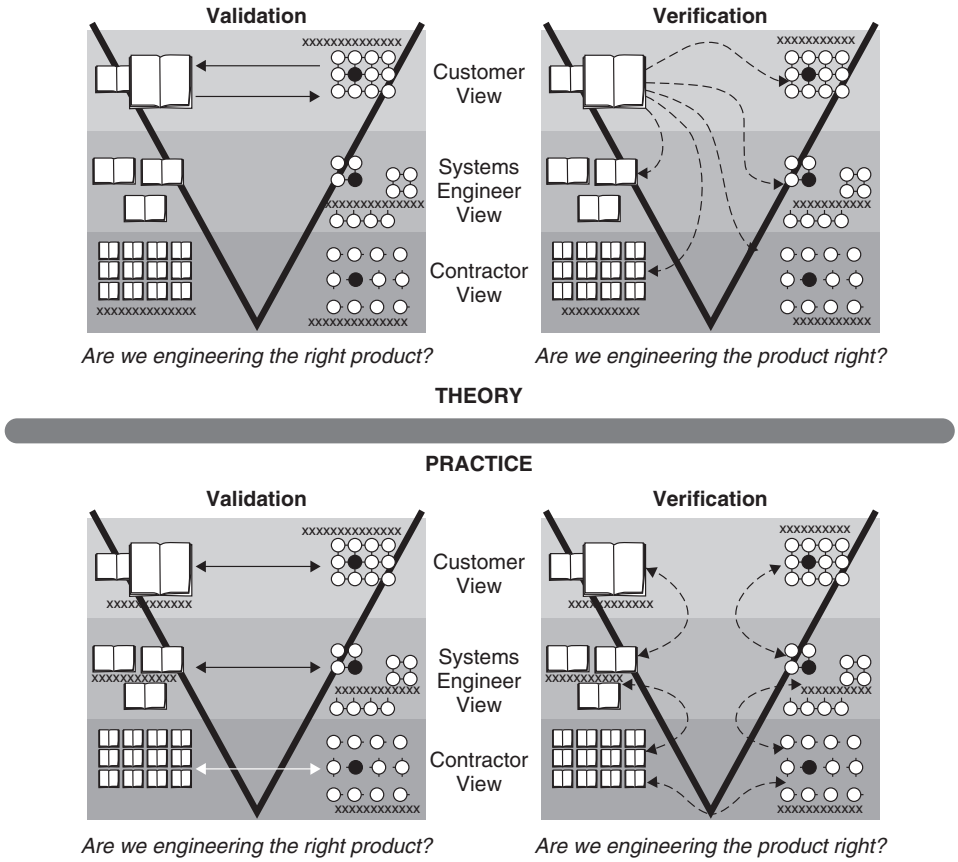

Figure 1.11 Theory and practice of verification and validation in the acquisition life cycle.

nonlinear with the distance between the beginning of the project and the point of problem detection. As a result, the added overhead expense of reducing this distance by validating at each stage and by verifying at each engineering step is likely to result in cost savings.

Forsberg and Mooz (1991, 1995) have developed a more comprehensive V chart, including a repeated V for incremental strategies (Shishko, 1995). On the downside of the V, definition and development phase activities, including requirements specification and design, lead to the fabrication activity, located at the bottom of the V. Inspection, verification, integration, and validation activities are found on the upside.

The system development phase may be described as a complete process with its own life cycle. The development phase begins with the principal products of the definition phase, namely, the system specification, the systems engineering management plan, and various contractual vehicles. The system specification is the ultimate source of guidance for the development phase. A system architecture should be derived from the system specification, and it should be consistent with the systems engineering management plan. Both the specification and the plan determine the architecture both of the product and of the systems engineering organization. Thus, the

derivation of architectural detail can be viewed as a management activity. The engineering activity that corresponds to this management activity is the identification of subsystems and the beginning of configuration control. To minimize effort and the associated expense of testing and integration, subsystems should be largely self-contained. Interfaces among subsystems should be few, small, and very well understood. This understanding should be captured in the subsystem specifications and subsystem test requirements.

The next level of detail comprises the specification and design of subsystem components, including hardware, software, firmware, and test procedures. Following design approval comes the fabrication and testing of hardware components, and coding and unit testing of software and firmware components. These activities may include the development of test equipment for hardware testing and scaffolding[8] for software unit testing.

Integration of components into subsystems is the next step, which involves not only integration testing but also validation of the built and tested subsystems with respect to subsystem specifications. Next, the subsystems are integrated into a system that can be integration tested and validated against the system specification. Integration test procedures are a product of this phase that may be useful in the system maintenance part of the system life cycle. When system testing is complete, full attention can be given to other aspects of preparing to deliver the system, such as installation materials and procedures, training manuals and special training equipment and procedures, maintenance equipment and manuals, and user's guides.

The third part of the systems engineering life cycle is deployment. This part of the life cycle begins with the actual delivery and installation of the system and ends with the decision to decommission the system. As before, this phase can be viewed as a process with its own life cycle. The deployment life cycle begins with delivery and installation, acceptance testing, operational testing and evaluation, and acceptance by the customer. The second phase accommodates changes to the system. The software community refers to change as software maintenance. There may be a formal maintenance agreement with a contractor that includes a maintenance plan, maintenance test plans, and maintenance test procedures. The third and final phase of the deployment life cycle is the decision to decommission the system. This phase is often preceded by a trade-off study to compare options for continuing to support the business process, such as hardware upgrade or replacement using existing, reengineered, or COTS products, or the acquisition of a new system. This version of the systems engineering life cycle is summarized in Figure 1.12.

The end of the deployment life cycle as depicted in Figure 1.12 is very much linked to the planning and marketing process. This part of the life cycle is concerned with how a deployed system is used to support a business process in some organization. In step 6, the concept of a trade-off study to compare options for continuing to support the business process is a very general notion. In the case of a manufacturing process, market factors are the primary criteria for continuing, renovating, replacing, or retiring

[8]Scaffolding refers to software written to simulate the essential aspects of the environment of the software that is to be tested. Normally, scaffolding is not a part of delivered software. Incremental and evolutionary development, however, may deploy scaffolding as a part of an interim product. For example, in incremental software reengineering, scaffolding may be delivered as part of early increments to provide an interface between new software and a legacy system that is gradually being replaced.

DEFINITION
1. elicitation of requirements
2. classification of requirements
3. analysis of requirements
4. prototyping of requirements
5. requirements documentation and specification
6. preparation of transition documents: request for proposal and a proposal, containing a statement of work, a systems engineering management plan, a work breakdown structure, and a negotiated list of other deliverables

DEVELOPMENT
1. creation of a system architecture and an organizational structure consistent with the system engineering management plan
2. establishment of configuration control
3. identification of subsystems
4. production of subsystem specifications and subsystem test requirements
5. specification of subsystem components
6. design of subsystem components
7. fabrication and testing of hardware components
8. coding and unit testing of software and firmware components
9. the development of environments and test equipment for hardware testing and scaffolding for software unit testing
10. integration of components into subsystems
11. integration testing of each subsystem
12. validation of the built and tested subsystems
13. integration of subsystems into a system
14. integration testing of the system
15. system validation against the system specification
16. integration test procedures
17. provide product information to planning and marketing
18. preparation of transition documents and other aspects of preparing to deliver the system, such as installation materials and procedures, training manuals and special training equipment, maintenance equipment and manuals, and user's guides

DEPLOYMENT
1. delivery and installation of the system
2. acceptance testing of the system
3. operational testing and evaluation of the system
4. system acceptance by the customer
5. formal maintenance agreement with a contractor that includes a maintenance plan, maintenance test plans, and maintenance test procedures
6. trade-off study to compare options for continuing to support the business process
7. decision to decommission the system
8. provision of current system description to requirements team (linkage to new cycle)

Figure 1.12 Steps in a systems acquisition life cycle.

the system that supports the process. More subtly, in the case of nonmanufacturing processes, the performance of the organization is the primary consideration. In both cases, change in the business process drives change in the system that supports it, although technological improvements alone may result in changes, especially where improvements in economy, reliability, or safety are possible.

Sage (1992a) describes a 22-phase acquisition life-cycle model. The steps can be organized into three general phases: (1) definition, (2) design and development, and (3) operations and maintenance. The 22 steps are shown in Figure 1.13.

SYSTEM DEFINITION

1. Perception of Need
2. Requirements Definition
3. Draft Request for Proposal (RFP)
4. Comments on the RFP
5. Final RFP and Statement of Work
6. Proposal Development
7. Source Selection

SYSTEM DESIGN AND DEVELOPMENT

8. Development of Refined Conceptual Architectures
9. Partitioning of the System into Subsystems
10. Subsystem Level Specifications and Test Requirements Development
11. Development of Components
12. Integration of Subsystems
13. Integration of the Overall System
14. Development of User Training and Aiding Supports

SYSTEM OPERATION AND MAINTENANCE

15. Operational Implementation or Fielding of the System
16. Final Acceptance Testing
17. Operational Test and Evaluation
18. Final System Acceptance
19. Identification of System Change Requirements
20. Bid on System Changes or Prenegotiated Maintenance Support
21. System Maintenance Change Development
22. Maintenance Testing by Support Contractor.

Figure 1.13 The 22-step acquisition cycle in Sage (1992a).

1.5 THE PLANNING AND MARKETING LIFE CYCLE

According to Kast and Rosenzweig (1970), *planning* is the systematic, continuous process of making decisions about the taking of risks, calculated on the basis of forecasts of future internal and external market conditions. Planning is modeled in several dimensions:

1. Repetitiveness, for which the authors visualize a continuum from single event to continuous
2. Organizational level
3. Scope, ranging from a functionally oriented activity to a total organizational endeavor
4. Distance in time into the future

The model yields an eight-step approach to business planning. A similar list is given by Briner and Hastings (1994), who refer to the steps as preconditions for effective strategic management:

1. Appraising the future political, economic, competitive, or other environment
2. Visualizing the desired role of the organization in this environment
3. Perceiving needs and requirements of the clientele
4. Determining changes in the needs and requirements of other interested groups—stockholders, employees, suppliers, and others

5. Providing a system of communication and information flow whereby organizational members can participate in the planning process
6. Developing broad goals and plans that will direct the efforts of the total organization
7. Translating this broad planning into functional efforts on a more detailed basis—research, design and development, production, distribution, and service
8. Developing more detailed planning and control of resource utilization within each of these functional areas—always related to the overall planning effort

We can identify basic phases in the planning and marketing process and organize the eight steps into twelve activities within the familiar framework of the generic three-phase life-cycle model:

1. Definition
   - Assessment of the market
   - Perception of demand
   - Strategic response to the market in the light of organizational goals
   - Articulation of market demand into product requirements
2. Development
   - Providing a system of communication
   - Promoting the flow of information
   - Research
   - Design and development
   - Production
3. Deployment
   - Distribution
   - Service
   - Planning in greater detail

Effective planning requires careful external market analysis. This is a principal reason that we treat planning and marketing as part of a single life cycle. While an organization has a uniquely informed perspective on its own capabilities, goals, and interests, a free market does not respect individual organizations. Demand is an unbiased market force. Thus, an organization, especially its leadership,[9] must perform exceptionally to recognize, analyze, and synthesize a unique response to market demand. The first three steps of the Kast–Rosenzweig approach address assessment of the market, perception of demand, the strategic response of the organization to the market in the light of its own goals, and the articulation of market demand into product requirements. Together with other similarly inspired activities, such as product prototyping, user testing, and test marketing, we can refer to these steps collectively as the external market analysis phase.

[9]The leader's role is essentially that of a politician. As Buckley (1979) noted: "The successful political leader is one who 'crystallizes' what the people desire, 'illuminates' the rightness of that desire, and coordinates its achievement."

There is also what can be called the internal market. This is the collection of interests within the organization and its support environment, such as the stockholders, employees (individuals, formal groups, and informal groups), suppliers and other creditors, governmental authorities, and other special interests. The responsiveness of the organization depends on a collection of internal market requirements that must be satisfied to assure the success of the organization. By providing a system of communication, promoting the flow of information, and encouraging broad intraorganizational participation in goal development and planning, total quality management (TQM) (Deming, 1982; Sage, 1992b, 1995) connects internal and external requirements in a manner designed to promote growth in all areas of the internal marketplace, the goal of which is the creation of high-quality products that satisfy requirements in the external marketplace. Steps 4 and 5 of the Kast–Rosenzweig approach address these issues.

After having decided on the destination, the organization must plan its journey. Goals must be formulated into functional efforts. Referring to steps 6, 7, and 8, these efforts include research, design and development, production, distribution, and service. Again, the internal organization must be consulted to develop more detailed planning and control of resource utilization within each functional area. Kast and Rosenzweig believe that providing organizational members with a voice in the planning process, resulting in an investment in the plans, serves to integrate the organization.

Planning and marketing is tightly coupled with both RDT&E and with the acquisition process. Blake and Mouton (1961, 1980) define a process for achieving an objective that may be vague initially and that may change over time with refinement. Given this statement of a goal, planning may proceed, leading to an action step. The action step enables the fact-finding or feedback stage that is critical to a goals-oriented management program, because it provides "steering data." Three qualitative measures may be evaluated as well: (1) progress toward the goal; (2) adequacy of planning; and (3) quality or effectiveness of any given action step. These measures can be quantified using a 9-point scale system or some similar heuristic. Experience and feedback using a prototype in the context of the RDT&E process is extremely useful for evaluating both products and processes for eventual deployment. Mistakes corrected in the RDT&E cycle are much less costly than those detected in the marketplace. In this light, RDT&E can be seen as part of the overall definition phase for the organization's strategic planning process.

1.6 SOFTWARE ACQUISITION LIFE-CYCLE MODELS

It is our opinion that software engineering is a subdiscipline of systems engineering. Some authors refer to *software systems engineering* (Sage and Palmer, 1990). Like with many issues, however, there are two opposite points of view on this opinion. Undoubtedly, software engineering, whose roots are in computer programming, has been slow to emerge as a discipline at all and for many years was regarded more as art than engineering. Even in recent years software development processes have seemed to diverge rather than to converge to a single methodology.[10] Different process models have different life cycles. In this section we present several important representatives.

[10]A *methodology* is defined as a collection of three sets: (1) tools, (2) activities, and (3) relations among the tools and activities (Warfield and Hill, 1972).

DEFINITION	create requirements and specifications		
	DEVELOPMENT	begins with the system specfication and ends with the delivery of the developed system	
		DEPLOYMENT	begins with delivery of the system and ends with system retirement

Figure 1.14 Three-level acquisition model.

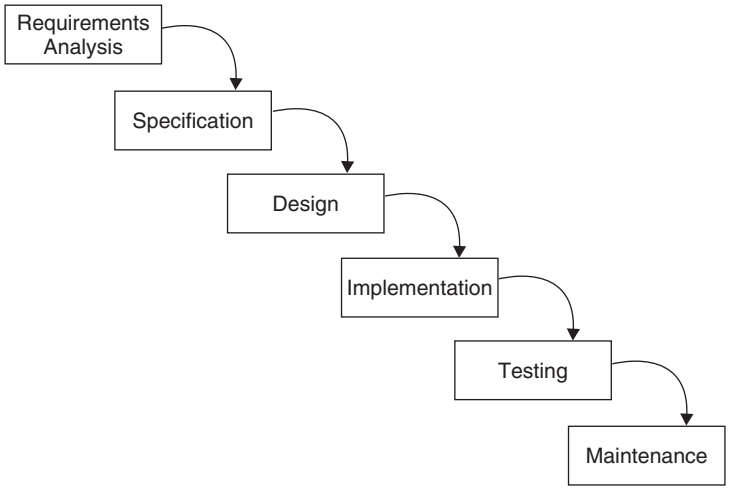

Figure 1.15 Royce's waterfall process model.

Many system acquisition or production life cycles have been proposed and used. A three-level model, shown in Figure 1.14, defines the basic shape.

It is widely acknowledged that Royce (1970) presented the first published *waterfall* model, depicted in Figure 1.15. The principal idea is that each phase results in a product that spills over into the next phase. The model is pleasing for several important reasons. First, the stages follow the basic systems engineering life-cycle template. Requirements analysis and specification fit into the definition phase. Design and implementation make up the development phase, while test and maintenance constitute the deployment phase. Second, the stepwise nature of the waterfall suggests that only one phase is active at any one time, reducing the scope of possible interests, and thus simplifying the engineering process. Third, the logical ordering and the temporal ordering of the steps are identical. Fourth, and perhaps most importantly from an engineering management point of view, each step is represented as being largely self-contained, suggesting that different organizations may be designated for each phase without loss of

continuity. Since the introduction of the waterfall model by Royce, many other software process models have adopted a similar format. For example, at the Software Engineering Laboratory at NASA, a seven-phase life-cycle model is used as the basis for cost estimation (McGarry et al., 1984). More recently, an essentially identical life cycle was used as the basis of the *NASA Software Configuration Management Guidebook* (National Aeronautics and Space Administration (NASA), 1995). The seven phases are as follows:

1. Requirements analysis
2. Preliminary design
3. Detailed design
4. Implementation
5. System integration and testing
6. Acceptance testing
7. Maintenance and operation

The purpose of any model is to provide a framework that is better understood than the reality that it models, and whose outcomes are sufficiently close to reality to counter the easy, obvious, and devastating objection: "Does this reflect the way things really are?" Most variations of the waterfall model may be represented as attempts to overcome this objection.

When we apply the binary "reality measure" to the waterfall model of Royce, several incongruities may be noticed. One problem is the duration of the phases: although phases tend to begin in the logically predictable sequence, in reality they never end. The products from each phase, if perfect, would indeed bring an end of the phase.

Yet all products are fatally flawed and must be dealt with throughout the software development life cycle. To see that they are flawed, consider software requirements, which are subject to many metrics. To present our case, we need only consider one: *completeness*. Requirements are complete if they anticipate all needs of the user. This is an impossible goal, one that is addressed by various techniques that are essential for progress, but which constrain the engineering process, such as configuration management, and verification and validation. Therefore, requirements are never complete and are with us throughout the life cycle.

A second problem is the suggestion that only one phase is active at one time. Once again, we need only consider requirements and recognize that this phase is active throughout the development of the product. A third problem is the possible claim that separate organizations may have complete control of the project at different times in the development process without the loss of continuity. It is clear, however, that a requirements team must remain intact throughout the development process to avoid *loss of information*, subsequent retraining, and learning curve inefficiencies.

Boehm (1981) presents a waterfall model, shown in Figure 1.16, that addresses these problems. It is based on nine products, the creation of which Boehm characterizes as sequential, and which achieve necessary software engineering goals:

1. *Concept.* Concept definition, consideration of alternative concepts, and determination of feasibility.
2. *Requirements.* Development and validation of a complete specification of functional, interface, and performance requirements.

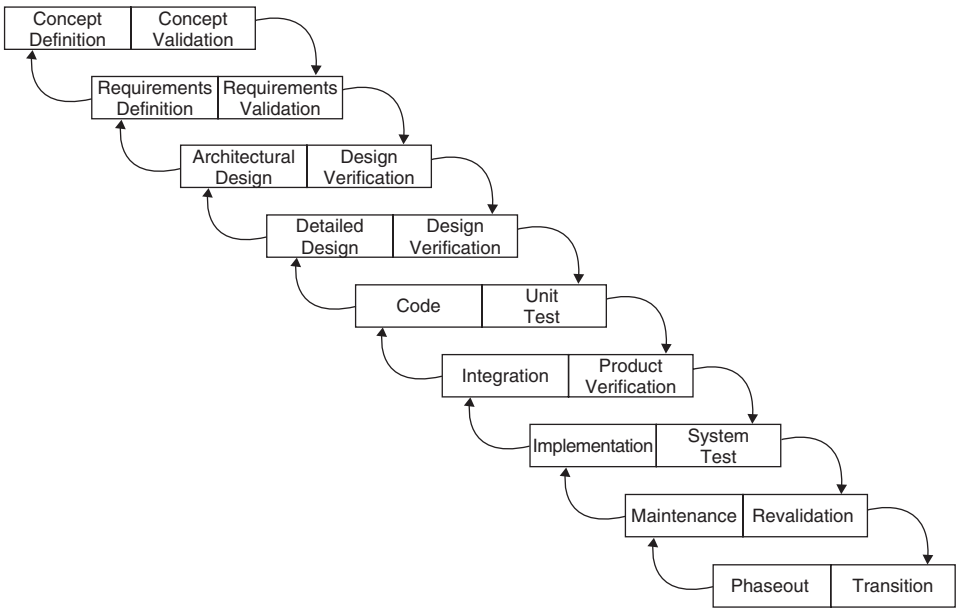

Figure 1.16 Boehm's nine-level version of the waterfall life cycle.

3. *Architectural Design.* Development and verification of a complete specification of the overall system architecture (hardware and software), including control and data structures.
4. *Detailed Design.* Development and verification of a complete specification of the control and data structures, interfaces, algorithms for each program component (defined as a program element of 100 lines of code or less).
5. *Code.* Development and verification of a complete set of program components.
6. *Integration.* Development of a working software system.
7. *Implementation.* Development and deployment of a working hardware and software system.
8. *Maintenance.* Development and deployment of a fully functional update of the hardware and software system.
9. *Phaseout.* The retirement of the system and the functional transition to its replacement.

In this version of the waterfall life-cycle model, as in Royce's version, Boehm includes a return path from each step in the model to the previous step to account for the necessity to change the product of a previous step. This rendition offers an orderly procedure for making changes as far back as necessary, both to meet the standards of verification and validation and to satisfy the underlying concept definition.

Boehm's waterfall life cycle is logically complete, insofar as that, at any given step, it successfully deals with deficiencies of previous steps. Moreover, it does so without violating the twin quality goals and constraints of configuration management and of validation and verification. The inadequacies remaining in the waterfall model are largely temporal and organizational.

A temporal problem is created, as noted in the Royce waterfall model, when a phase is deemed to have ended. The most striking example is the requirements phase. This phase does in fact logically precede the design phase. However, the requirements phase does not end when the design phase begins. It is of the nature of the waterfall model that each step begins with a set of requirements upon which action is required. This set of requirements is the result of actions in a previous step on a previous set of requirements. Likewise, when the current step is complete, the result will be requirements for the following step. Thus, at each successive step in the waterfall, requirements beget requirements (as well as changes to previous sets of requirements), the requirements for the whole governing the derivation of requirements for its parts. Every step is a requirements step, and not one of them ever ends.

One can picture the organizational consequences of living with this problem over the lifetime of a project. At least four scenarios can be generated, as shown in Figure 1.17, depending on the size of the engineering problem and the size of the organization. For simplicity, we can characterize problems, as well as organizations, as large or small, examining each case in turn.

Case 1. This is the case in which a small organization has a small engineering problem to solve. Provided that the organization has both the engineering knowledge (skill) and the tools required, a small team can perform all life-cycle tasks with great efficiency and with a very high probability of success. There are many success stories about significant systems that have been specified, designed, built, tested, and successfully marketed by entrepreneurs with meager, but adequate, resources. There are doubtlessly countless untold success stories about more typical efforts of small engineering organizations.

Case 2. In this case, a large organization has a small engineering problem to solve. Management may decide to exploit the small problem size to advantage by using the approach of Case 1. Alternately, several small teams may be engaged, thus introducing additional learning curve costs, necessitating the continuous employment and communication among a larger number of engineers, and lowering the probability of successful completion.

Case 3. When a small organization undertakes a large engineering task, management must make several decisions based on available resources. If time is not a critical factor, a small team approach offers the high quality, the low risk, and the economy of Case 1. In effect, the large problem has become manageable by subdivision into small problems without introducing the problems of using

	small engineering problem	large engineering problem
small organization	1. small team may perform all steps in the life cycle	3. small team may perform all steps in the life cycle; or other small teams may be employed
large organization	2. one or more small teams may perform the steps in the life cycle	4. one large team, many small teams, or a compromise organizational structure may be used to perform the steps in the life cycle

Figure 1.17 Organizational size versus problem size.

subcontractors. Unfortunately, time is almost always a critical factor, and management is forced into the role of general contractor, and assuming the high risk of using independent contractor teams, each with its own layer of management, its own development process, and its own strategic goals.

Case 4. Assuming that the large organization has the required skills, methods, and tools, the high risk of Case 3 can be mitigated somewhat by performing all tasks without resort to external organizations. In this way, a single development process, and a single set of strategic goals, improves communication and cooperation among groups of engineers. However, large organizations are inherently plagued by problems of management complexity and communication complexity (Brooks, 1975). Moreover, because of the need to retain the organizational expertise from previous steps at each successive step in the life cycle, the total organization may be very large, and largely unoccupied, at any given time.

We can see that problem management is a logical issue, and that organizational management is a temporal issue. Partially in recognition of the organizational problem, Boehm (1981, 1990b) devised the spiral model, depicted in Figure 1.18.

We have looked at the structure and the function of the waterfall. An insight into the nature of the possible problems with the waterfall can be gained by reflecting on the purpose. Life-cycle models represent an attempt to manage complexity through division of resources. The ordering of processes and products into phases—the organizational structuring—is necessary to managing the complexity of the solution. The complexity of the problem, however, cannot be reduced by dealing exclusively with the resources assembled to solve the problem. The problem itself must also be reduced into a tractable form and size, where tractability varies according to whether the domain is well understood. The waterfall model is well able to be applied to manageable problems in familiar domains.

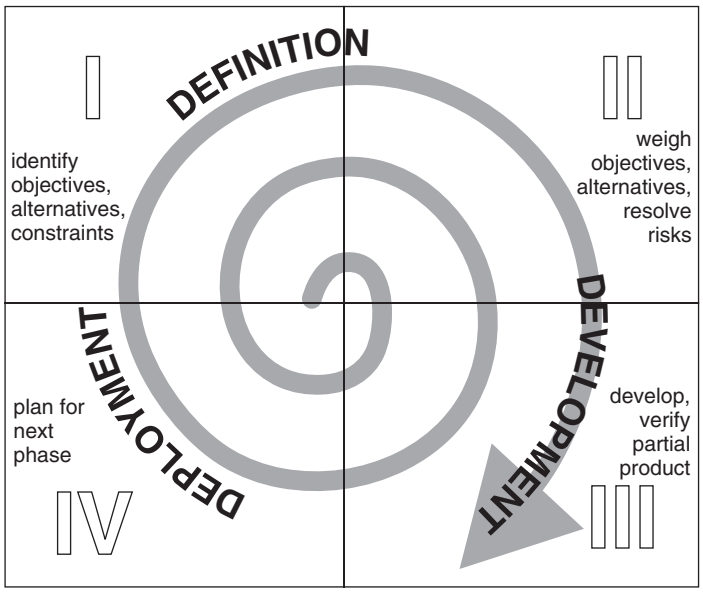

Figure 1.18 Boehm's spiral development model.

As noted previously, Boehm identified a repetitive aspect to the spiral model. Whytock (1993) describes the spiral as comparable to a repeated waterfall model. As shown in Figure 1.18, the spiral model attempts to abstract a set of activities that every layer of the spiral has in common. Boehm divides the model into four quadrants, shown in Figure 1.18. Quadrant I is the beginning of each level of the spiral and is concerned with identifying needs, possible ways of satisfying the needs, and factors that may constrain the solution. If Quadrant I may be viewed as the early part of the requirements development life cycle, Quadrant II represents the latter part, during which requirements are analyzed, problems are resolved, and trade-offs are made. These two quadrants represent the parts of the waterfall model concerned with definition of the system to be built. In Quadrant III, a partial product is developed and verified, and this product is "deployed" in Quadrant IV, although such deployment simply represents a stepping stone for beginning a new definition effort in a new layer of the spiral model.

It should be noted that the spiral is a variation of the waterfall model that adds generality by including repetition as its basic feature. By unwinding the spiral we may recover the waterfall model, as well as solutions based on partial implementation, such as incremental development and evolutionary development.

The waterfall process model can be seen to be most appropriate for the development of products in a familiar domain, where the risk of building poor products is reduced by an experience base that may include reusable specifications and designs. In an unfamiliar domain, or in large or complex projects, an incremental approach reduces risk, since the cost of each increment is relatively small. An increment may even be discarded and redeveloped without catastrophic cost consequences. The great strength of the spiral model is the capability to develop increments, or prototypes, with each full turn of the spiral. The prototype that is specified, planned, built, tested, and evaluated is now a working core version of the final system. Subsequent possible failures in later turns of the spiral will likely not impact the successfulness of previous increments.

Incremental development is a variation of the divide-and-conquer strategy in which software is built in increments of functional capability. In this approach, as defined by Boehm (1981), the first increment will be a basic working system. Each successive increment will add functionality to yield a more capable working system. The several advantages of this approach include ease of testing, usefulness of each increment, and availability during development of user experiences with previous increments. Boehm's modification of the waterfall to allow incremental development is to provide a number of successive copies of the section of the waterfall model, starting with detailed design; followed by code, integration and product verification, implementation and system test; and ending with deployment and revalidation. The spiral model provides a more succinct representation in which each successive increment is assigned to the next higher level of the spiral.

The evolutionary development model is an attempt to achieve incremental development of products whose requirements are not known in advance (Rubey, 1993; U.S. Department of Defense, 1994). Boehm discusses a process that can be used with iterative rapid prototyping, especially automatic program generation, and user feedback to develop a full-scale prototype (Boehm, 1981). This prototype may be refined and delivered as a production system, or it may serve as a de facto specification for new development. More generally, where evolutionary development is possible, it can be

represented using a waterfall model in the fashion used by Boehm to represent incremental development. This is the approach taken in MIL-STD-498 (U.S. Department of Defense, 1994). However, representing evolutionary development entails repetition of the requirements and specification development activities; thus, substantially more of the waterfall must be repeated. Again, the spiral model is a more natural representation, since deployment for a given level of the spiral is directly linked to the definition portion of the next higher level, allowing necessary requirements growth in an orderly fashion.

From the standpoint of organizational management, the incremental software development model affords a more economical approach than the "grand design" strategy (U.S. Department of Defense, 1994) by allowing better utilization of fewer people over a comparable period of time (Boehm, 1981). Referring again to the Stogdill model in Figure 1.1, stability of processes is closely related to stability of structure in the organization. The changes in organizational size and structure that are suggested by the use of the grand design model can have a destabilizing influence on the organization. Davis et al. (1988a,b) suggest metrics for use as a possible basis for deciding among alternative software process models.

A similar use of the prototyping concept is *rapid development* (Reilly, 1993). The term *rapid development* refers to a sequence of prototype development efforts, each brought to operational status using a waterfall process, and deployed into actual operational environments. This is not to imply that the entire system is built rapidly, but that operational subsets are engineered in 10- to 18-month cycles. Each deployed system after the first is a replacement for its predecessor. This incremental process ensures that meaningful user feedback and operational testing will take place prior to the final release of the system. Moreover, as new requirements emerge, stimulated by operational experience with previous deliveries, they can be incorporated into subsequent iterations of the waterfall.

A standard released by the United States Department of Defense, MIL-STD-498, *Software Development and Documentation* (1994), was designed to accommodate three basic development strategies, discussed earlier: *grand design, incremental*, and *evolutionary development*. These strategies are also found in related standards DoDI 5000.2 (U.S. Department of Defense, 1991) and DoDI 8120.2 (U.S. Department of Defense, 1993). In this way, MIL-STD-498 represents an improvement over its predecessors, such as DoD-STD-2167a (U.S. Department of Defense, 1988a), DoD-STD-7935A (U.S. Department of Defense, 1988b), and DoD-STD-1703 (U.S. Department of Defense, 1987). To achieve this flexibility, MIL-STD-498 abstracts from all three development strategies the concept of a "build," one or more of which is necessary to implement the mature software product. Each successive build incorporates new capabilities as specified and planned into an operational product.

In 1995 the joint standard, *Software Life Cycle Processes* (ISO/IEC 12207), was approved by the International Organization for Standardization and the International Electrotechnical Commission (1995). This standard contains 17 processes grouped into three sets:

I. Primary Processes (5)
   (1) Acquisition
   (2) Supply
   (3) Development

(4) Operation
(5) Maintenance

II. Support Processes (8)
(6) Documentation
(7) Configuration management
(8) Quality assurance
(9) Verification
(10) Validation
(11) Joint review
(12) Audit
(13) Problem resolution

III. Organizational Processes (4)
(14) Management
(15) Infrastructure
(16) Improvement
(17) Training

Each process contains activities, and each activity contains tasks. There are 74 activities and 224 tasks. Each process incorporates a TQM "plan–do–check–act" cycle. The processes, activities, and tasks are building blocks that must be assembled according to the project or organization. In the summer of 1998, MIL-STD-498 was replaced by a new standard, IEEE/EIA 12207, which is an adaptation of ISO/IEC 12207 that implements the intent of MIL-STD-498 (Sorensen, 1996).

1.7 TRENDS IN SYSTEMS ENGINEERING LIFE CYCLES

The essence of life-cycle management is division of resources (Buck, 1966). Kingsley Davis (1949) recognized division of labor as one of the fundamental characteristics of socialized humans. The division of tasks into subtasks and the documentation of progress through intermediate products, these are important concepts that did not begin in abstraction, but in concrete problems. The essence of engineering is problem solving; and divide-and-conquer strategies, process definition and improvement, process modeling, and life-cycle management are all problem-solving methods that begin by reducing complex challenges into tractable parts and conclude by integrating the results.

The failures of some systems engineering efforts have led some theoreticians to criticize or abandon the traditional ways of dividing resources. The trend is broad and can be seen in many areas. Life cycles are criticized for many reasons. For example, the concept that a requirements phase is self-contained, self-supportive, and separable from other phases has come under sharp attack from both academia and industry. The deeply ingrained tradition in industry that requirements must not dictate any given specific design is also coming into question. The justification for this belief may readily be derived from our basic activity model, shown in Figure 1.19.

Each of the three orthogonal activities is necessary for successful systems engineering. It is not clear, however, that separation in time, separation by assignment to more than one action team, or separation in terms of any other resource based on the

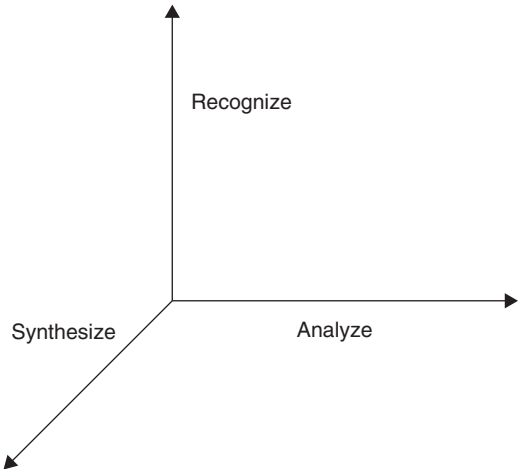

Figure 1.19 Engineering activity model.

orthogonality of a process model (i.e., a life cycle) is effective in reducing engineering complexity issues. In fact, the opposite seems to be true. In the natural course of raising and resolving problems, it is much more efficient to solve problems when and where they arise. The inefficiencies associated with following each life-cycle step to completion before beginning the next step tend to result in a loss of information, and this tendency is exacerbated in proportion not only to the size of the engineering problem but also to the size of the engineering resources (Brooks, 1975). This is a major obstacle in engineering big systems.

A solution based on partitioning big systems into a number of small systems reduces the inefficiency inherent in life-cycle-based approaches. However, this solution does not answer the emerging generation of systems engineering theorists who maintain that life-cycle activities are dependent on each other in a nontrivial way that neither a waterfall model, nor even a spiral model in its full generality, can adequately address. This suggests that specialties, such as *requirements engineering*, are undesirable, unless their scope of activity spans the entire life cycle: definition, development, and deployment. An alternate way to view this suggestion is that the number of specialties is effectively reduced to one: *systems engineering*. The systems engineer is ever cognizant that a system must be specified, built, tested, and deployed not only in terms of its internal characteristics but also in terms of its *environment*. Thus, the job of integrating becomes a global concern that subsumes all of the steps in the life cycle.

1.7.1 Process Improvement

The trend away from the use of prescribed standard life cycles, while widespread in the private sector, has only recently begun to affect government acquisitions. The U.S. Department of Defense no longer supports the use of MIL-STD-499A (1974), the successor of MIL-STD-499 (U.S. Department of Defense, 1969) for the acquisition of systems. (MIL-STD-499B (U.S. Department of Defense, 1992), in "draft" form since 1992, has never been approved.) Moreover, the new software development standard,

MIL-STD-498, after years of development, has been canceled, a loss that will be unlamented because of the standard's virtual replacement by the U.S. Commercial Standard (Sorensen, 1996) as previously noted. Other military standards are receiving similar treatment. Developers have their own processes that are specific to their own organizations. Also, commercial standards are emerging. In terms of Stogdill's model of organizational dynamics (Fig. 1.1), it is no longer clear that process models should precede the development of structure and interpersonnel attributes, since, without a customer-mandated process model, a conceivably more efficient and efficacious process may evolve naturally. This change in emphasis should not be viewed as freedom from the use of a disciplined approach, but rather freedom to customize the approach to optimize quality attributes.

The wide acceptance of ISO-9000 as an international standard is actually a trend away from standardized process models. The basic philosophy of ISO-9000 has been summarized as "say what you do; then do what you say" (Ince, 1994; Rabbitt and Bergh, 1994). ISO-9000 certification (International Benchmarking Clearinghouse, 1992) is a goal to which many companies aspire in order to gain competitive advantage. Certification demonstrates to potential customers the capability of a vendor to control the processes that determine the acceptability of the product or service being marketed.

The Software Engineering Institute, in the late 1980s and early 1990s, developed a family of models, known as the Capability Maturity Models (CMMs) (Humphrey, 1989; Paulish and Carleton, 1994; Software Engineering Institute, 1991), upon which methods and tools for process assessment and improvement in the software development arena have been based. As the name suggests, these model are based on the existence of documented and dependable processes that organizations can use with predictable results to develop products. In effect, the details of the process are of little interest, as long as the process is repeatable.

The CMM family has evolved into the CMMI (CMMI Product Team, 2006). The "I" refers to integration of a number of critical elements, each of which must stand alone and effectively support other elements of an engineering organization. The CMMI Version 1.2 contains 22 distinct, but interrelated, process areas (PAs):

1. Causal Analysis and Resolution (CAR)
2. Configuration Management (CM)
3. Decision Analysis and Resolution (DAR)
4. Integrated Project Management + Integrated Product and Process Development (IPM + IPPD)
5. Measurement and Analysis (MA)
6. Organizational Innovation and Deployment (OID)
7. Organizational Process Definition + IPPD (OPD + IPPD)
8. Organizational Process Focus (OPF)
9. Organizational Process Performance (OPP)
10. Organizational Training (OT)
11. Product Integration (PI)
12. Project Monitoring and Control (PMC)
13. Project Planning (PP)
14. Process and Product Quality Assurance (PPQA)

15. Quantitative Project Management (QPM)
16. Requirements Development (RD)
17. Requirements Management (REQM)
18. Risk Management (RSKM)
19. Supplier Agreement Management (SAM)
20. Technical Solution (TS)
21. Validation (VAL)
22. Verification (VER)

With the release of CMMI Version 1.2 (2006), the Software Engineering Institute has made several changes that strengthen the CMMI level ratings. Both uniformity and rigor characterize the new model. The SEI might be seen as leveraging years (Chrissis et al., 2003) of very successful marketing of a "Have It Your Way" CMMI into a more respectable, and arguably a preeminent, competitor to various ISO and military standards worldwide (e.g., see ISO/IEC (2004)). A number of changes to the content of the model include increased attention to hardware issues, the inclusion of two work environment practices (in Organizational Process Definition (OPD) and Integrated Project Management (IPM)), and an increased emphasis on the importance of project start-up. Other significant changes include planned enforcement of a "shelf life" of ratings to a maximum of 3 years, and increased authority of CMMI appraisal teams over the scope (choice of process areas) and applicability (choice of project teams) of SCAMPI appraisals.

Originally, the CMM was adapted from the five-level model of Crosby (1979) to software development by Humphrey. The five levels of the CMM model are as follows:

1. *The Initial Level.* Ad hoc methods may achieve success through heroic efforts; little quality management, no discernible process; nothing is repeatable except, perhaps, the intensity of heroic efforts; results are unpredictable.
2. *The Repeatable Level.* Successes may be repeated for similar applications; thus, a repeatable process is discovered that is measurable against prior efforts.
3. *The Defined Level.* Claims to have understood, measured, and specified a repeatable process with predictable cost and schedule characteristics.
4. *The Managed Maturity Level.* Comprehensive process measurements enable interactive risk management.
5. *The Optimization Level.* Continuous process improvement for lasting quality. According to Sage (1995): "There is much double loop learning and this further supports this highest level of process maturity. Risk management is highly proactive, and there is interactive and reactive controls and measurements."

The CMM helps software project management to select appropriate strategies for process improvement by (1) examination and assessment of its level of maturity, according to a set of criteria; (2) diagnosis of problems in the organization's process; and (3) prescription of approaches to cure the problem by continual improvement.

Even though managers may be seasoned veterans, fully knowledgeable about the problems and pitfalls in the engineering process, they may disagree with each other on how to cope with problems as they occur. If agreement is difficult to produce in

an organization, the resultant lack of focus is taxing on organizational resources and may endanger the product. Thus, the management of an organization must be greater than the sum of its managers by providing strategies for management to follow and tools for management to utilize. Such strategies and tools will be the result of previous organizational successes and incremental improvements over time, and measured by a level of maturity. The CMM, developed at the Software Engineering Institute (SEI) at Carnegie Mellon University, provides a framework that is partitioned by such levels of maturity. Although the CMM was developed to measure the maturity of software development processes, the ideas on which it is based are quite general, applying well to systems engineering, and extensible to such processes as software acquisition management and even unto the software engineer's own personal software process (Humphrey, 1995).

In analyzing the CMM, it is helpful to look closely at the five levels or stages in quality maturity attributable to Crosby (1979):

1. *Uncertainty.* Confusion, lack of commitment. "Management has no knowledge of quality at the strategic process level and, at best, views operational level quality control inspections of finished products as the only way to achieve quality."
2. *Awakening.* Management wakes up and realizes that quality is missing. "Statistical quality control teams will conduct inspections whenever problems develop."
3. *Enlightenment.* Management decides to utilize a formal quality improvement process. "The cost of quality is first identified at this stage of development which is the beginning of operational level quality assurance."
4. *Wisdom.* Management has a systematized understanding of quality costs. "Quality related issues are generally handled satisfactorily in what is emerging as strategic and process oriented quality assurance and management."
5. *Certainty.* Management knows why it has no problems with quality.

In each of these environments, a particular kind of person is required. There is a shifting in focus from one type of key individual to another as we move from one CMM level to the next. The progression seems to be roughly as follows:

1. *Heroes.* Necessary for success in a relatively unstructured process, the hero is able to rise above the chaos and complete a product.
2. *Artists.* Building on the brilliance of the heroes, the artists begin to bring order, resulting through repetition in a codifiable process.
3. *Craftspeople.* These are the people who follow the process, learning from experience handed down from previous successes.
4. *Master Craftspeople.* These people are experts in their respective facets of the development process, who understand and appreciate nuances of process and their relationship to quality.
5. *Research Scientists.* Finally, master craftspeople appear, who, through experiential learning and attention to process integration, are able to fine tune the process, improving the overall process by changing steps in the process, while avoiding harmful side effects.

The characteristics of the organizational culture are directly related to organizational learning. The organization appears to depend primarily on two factors: (1) the people

who compose the organization and (2) the environment internal to the organization. Of course, a great case may be made for including the external environment, since the overall success of the organization (and its probability of survival) are directly related to its correct adaptation to external factors, including both the market and technological factors. Following the CMM, some of the organizational characteristics may be organized in five stages, as follows:

1. *Heroes and Supporters.* Dependent on the ability of heroes to rise above the chaos, the organization grows up around the activities of each hero, each of whom may require low-level support services. The hero's processes are largely self-contained and very loosely coupled with other heroes' processes. While there are very efficient aspects of this kind of organization (viz., the hero's own activities), there is no overall efficiency induced by integration of activities into an overall process. Thus, at this CMM level, there are really two levels of workers: heroes and others.
2. *Artist Colony.* Through mutual respect and attention to the successful practices of the previous generation of heroes, these people work together to recreate the successes of the past, creating new processes along the way. Management begins to be able to track progress.
3. *Professional Cooperative Organization.* Through long service and attention to process, master craftspeople have emerged, creating more hierarchical structure in the organization, as less experienced individuals are able to learn from the more experienced. There now exists the concept of "the way to do the job," a concept that must be measurably adhered to. Management's role is to control adherence to the process by defining metrics and implementing a metrics program.
4. *Society of Professionals.* At this point, the organization is mature enough to be able to receive from its individual members meaningful suggestions on how to improve selected parts of its process and to implement them in the overall process. This is largely a shift in the organization's ability to learn.
5. *Institute of Professionals.* The organization is now so mature that it is able to look continuously for ways to improve processes. Outside influences are no longer repelled, but are welcomed and evaluated.

The trend away from rigid standardization of systems engineering process models can be understood better when viewed in the context of system acquisition. A relatively new trend in acquisition management is the widespread use of performance-based contracting (PBC), a procurement tool based on the systems engineering approach that emphasizes the *purpose* of the acquisition. Figure 1.20 is based on a systems engineering model of Sage that relates three orthogonal dimensions of systems engineering projects: purpose, structure, and function (Sage, 1992a, 1995). By focusing attention on purposeful aspects (goals, needs of the consumer, the *why*), as opposed to the structure (the *what*) and function (the *how*) of the solution, the vendor is not constrained to life-cycle models, development tools, methodologies, and products specified by the consumer.

In effect, the consumer and the vendor enter a partnership agreement at a higher level of abstraction than in conventional contracting methods. Figure 1.21 depicts a relationship among four acquisition strategies:

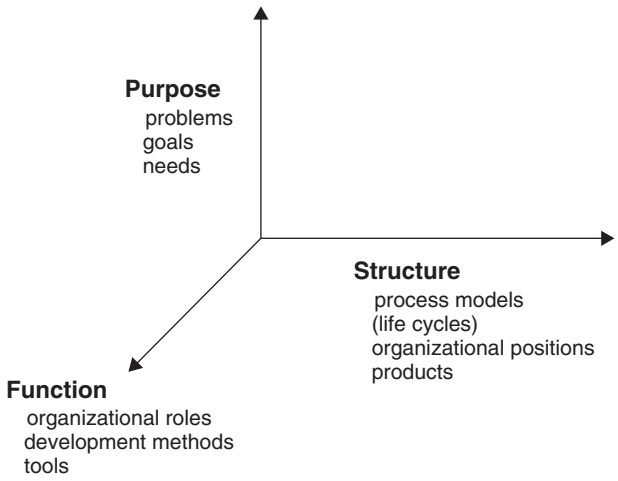

Figure 1.20 Dimensions of systems acquisition.

1. In-house development, using contracting strategies such as "level of effort (LOE)"
2. Conventional strategies based on "cost plus" contracting
3. PBC
4. COTS acquisition

The key determinant in Figure 1.21 is the level of agreement between the consumer and the vendor. In the case of in-house development, the needs of the consumer are satisfied through internal means, and any externally procured components, personnel, or other resources are managed by the consumer. The agreement of the consumer with the vendor is at the product level.

In the case of conventional contracting strategies, the consumer and vendor enter into a partnership at the process level of agreement. The consumer, driven by an unsatisfied need, analyzes the problem, synthesizes a solution to the problem, and creates a product specification. The specification is presented to the contractor, who writes a development plan. It is agreement upon the vendor's development plan that is the basis of the transaction between consumer and vendor. The development plan sets forth a process model (development life cycle), methods, tools, and, in some cases, organizational roles and responsibilities that the vendor agrees to follow. The consumer accepts an oversight role and a responsibility to convey to the vendor enough information from the problem space (the *purpose* dimension) to ensure the successful completion of the solution. Whenever a problem is reformulated in terms of a solution, however, information is lost. All the answers to the "why" question have been replaced by a specification. The "lost" information is retained by the *consumer* organization. Unfortunately, the organization that needs this information is not the consumer, but the vendor. While this lossy contracting procedure may work for the procurement of small and well-understood systems, large, innovative efforts magnify the loss and improve the probability of failure.

PBC reduces the loss of information by raising the level of abstraction of the information handed off from the consumer to the vendor. As shown in Figure 1.21, the organizational interface is at the *management* level. Both organizations address the

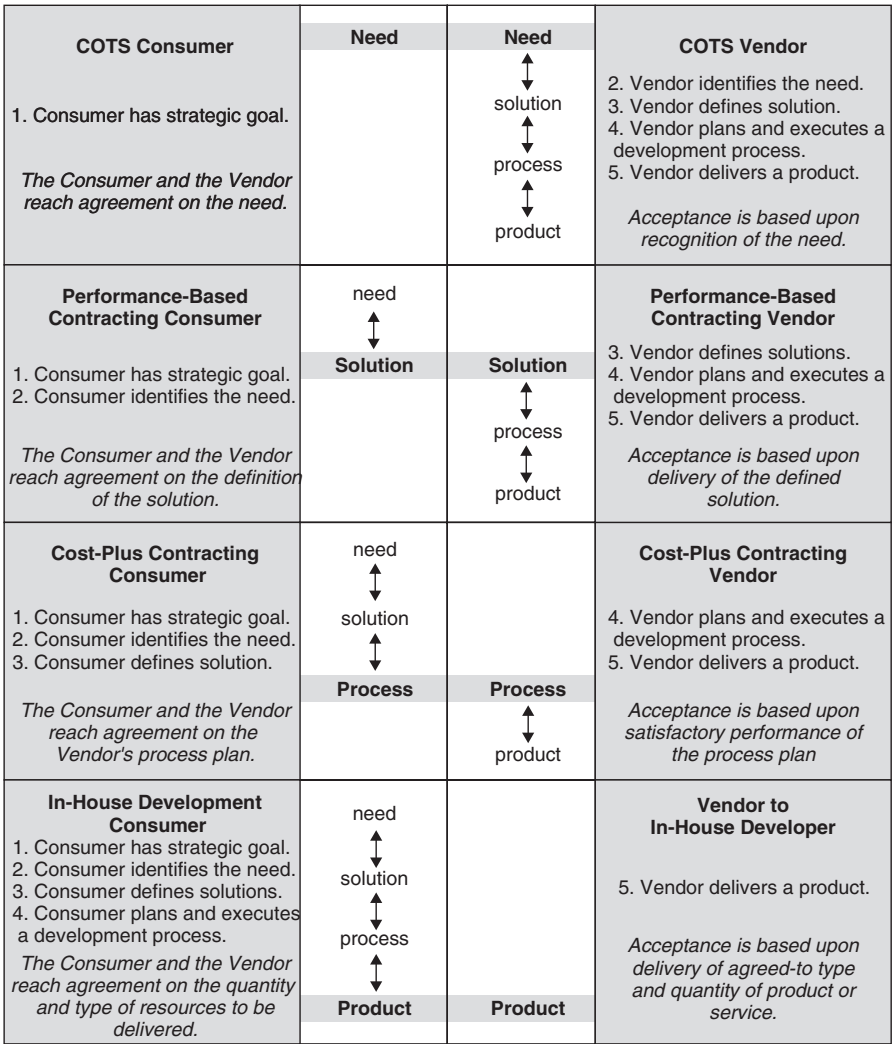

Figure 1.21 A comparison of contracting strategies based on the level of agreement.

problem space. It is the role of the vendor to define, develop, and deliver a solution, beginning with a vendor-developed specification. It is this specification that is the basis of agreement between the consumer and the vendor. By reversing the roles of the contracting parties, the vendor is responsible for the entire solution to the consumer's problem. The probability of failure has been reduced by eliminating an extremely complex and difficult process, the communication of purpose from one organization to another. Of course, both the consumer and the vendor can still fail, by reaching agreement on a poor solution.

Even this risk can be reduced substantially by the fourth contracting strategy shown in Figure 1.21. COTS procurement raises the level of communication to an even higher level of abstraction, so that a product can be purchased that directly satisfies the purpose for which it is desired.

1.7.2 Concurrent Engineering

Concurrent (or simultaneous) engineering is a technique that addresses the management of total life-cycle time (Carter and Baker, 1992; Rosenau, 1990; Slade, 1993), focusing on the single most critical resource in a very competitive market: time to market (or time to deployment). This is accomplished primarily by shortening the life cycle through the realization of three engineering subgoals:

1. Introduction of customer evaluation and engineering design feedback during product development
2. A greatly increased rate of focused, detailed technical interchange among organizational elements
3. Development of the product and creation of an appropriate production process in parallel rather than in sequence

Concurrent engineering is a metaprocess in which domain experts from all the departments concerned for developing a product at any stage of the life cycle work together as a concurrent engineering (CE) team, integrating all development activities into one organizational unit. The formation of the team does not per se shorten the engineering life cycle; however, through early involvement with the CE team, organizational learning and analysis activities can be removed from the critical path to market. There is an explicit trade-off of workers for time to market. That is, the CE team involves more personnel for a greater fraction of the life cycle[11] than in the case of the waterfall model. However, the time to market can greatly be reduced. In terms of the abstract life-cycle model, the activities labeled "recognize," "analyze," and "synthesize" can occur concurrently for all organizational elements involved in the development of the product. Of course, there will be some activities that have temporal, as well as logical, sequential dependence on other activities (see Fig. 1.22a,b). Marketing, drawing upon organizational expertise, including RDT&E products, begins the process through the generation of an idea of a product, based on market analysis. Marketing will generate targets for the selling price and the production costs of the proposed product to support management in deciding whether to proceed with product

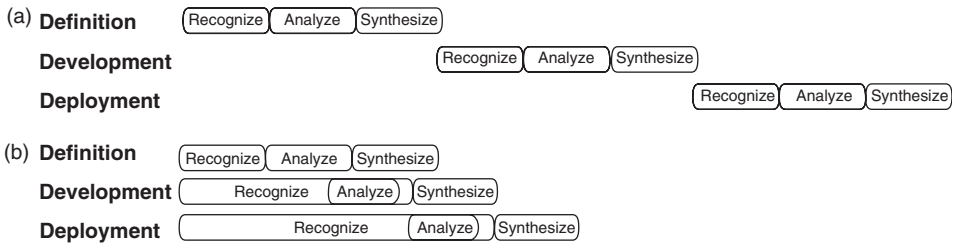

Figure 1.22 (a) Waterfall representation of abstract life cycle. (b) The compressing effect of concurrent engineering on the waterfall model.

[11]Although the CE team remains together for a greater percentage of the total life cycle, the life cycle is significantly shorter than in traditional models. Consuming a greater portion of a smaller resource may not increase cost and, in some cases, may actually decrease cost.

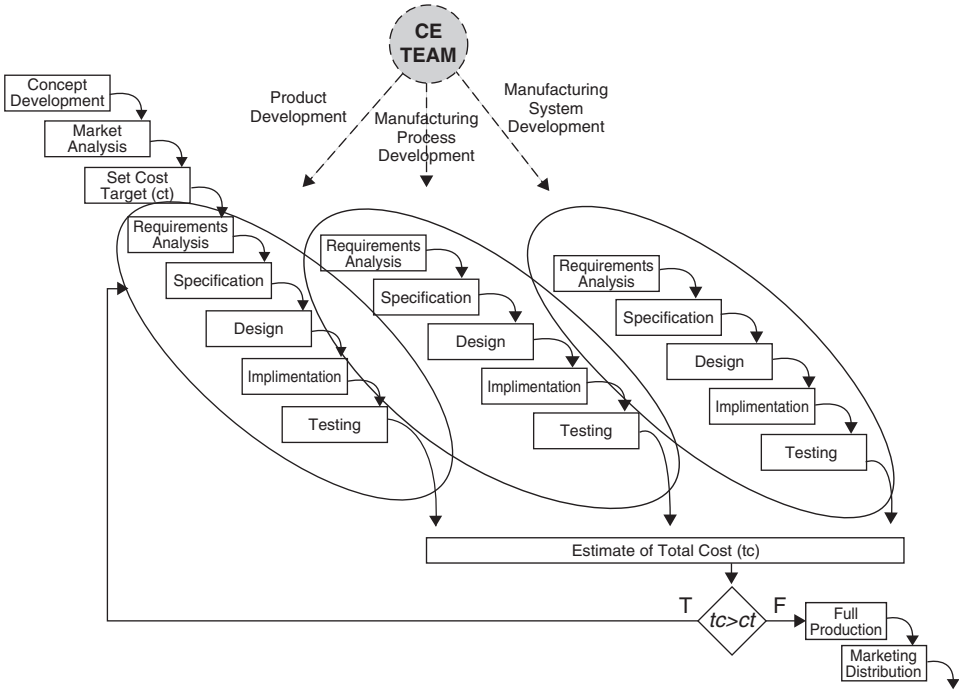

Figure 1.23 A concurrent engineering life-cycle model.

development. During development, the CE team works simultaneously with the design team to generate in parallel a design for the manufacturing *process*.

A CE life-cycle model is shown in Figure 1.23. The principal feature of this process model is the concurrent development of the product, the manufacturing process, and the manufacturing system through the continuous participation of the CE team (Yeh, 1992). A notable feature of the life cycle is the absence of a return path from production to design. This deliberate omission is in recognition of the extremely high risk of losing market share because of engineering delays due to design errors.

The organizational response to a change to concurrent engineering from traditional methods is likely to be fraught with difficulty. An organization that has formed around a particular life-cycle model and that has experienced a measure of success, perhaps over a period of many years, will almost certainly resist change. Effort in several specific areas appears to be basic to any transition:

1. We have noted that life-cycle models are purposeful, in that they reflect, organize, and set into motion the organizational mission. A change of life-cycle model should be accompanied by a clearly stated change of mission that may be digested, assimilated, and rearticulated by all organizational elements: individuals, formal and informal team structures, and social groups.
2. Formal team structures should be examined, destroyed and rebuilt, replaced, or supplemented as necessary to conform to the new life-cycle model. Informal team structures and individual expectation, such as those described by Stogdill, may be

replicated, and thus preserved, by the formal organizational structure to minimize loss.

3. Particular attention should be paid to intramural communication and cooperation among individuals, teams, and departments in the organization (Carter and Baker, 1992). Communication and cooperation are essential elements of concurrency. Acquisition or improved availability of communications tools, developed through advances in communications technology, may reduce the cost and increase the rate of concurrency.

The NASA *Systems Engineering Handbook* (Shishko, 1995) defines concurrent engineering very generally as "the simultaneous consideration of product and process downstream requirements by multidisciplinary teams." A slightly more conservative view is that of Grady (1993), who characterizes concurrent engineering as the correction of a mistake that has been made by many organizations, which he refers to as "transom engineering," wherein engineers working in isolation throw their respective products over transoms to their successors in a linear process. Grady believes that this is a gross distortion of the correct concept of systems engineering, and that concurrent engineering is merely a return to what systems engineering should be. Additional discussions of concurrent engineering are presented in Chapter 9.

1.7.3 Software Reengineering for Reuse

On the basis of two types of reuse identified by Barnes and Bollinger (1991), it is useful to distinguish between two different types of software reengineering (Chikofsky and Cross, 1990) for reuse:

1. Software reengineering for maintenance (adaptive reusability) improves attributes of existing software systems, sometimes referred to as legacy systems, which have been correlated to corrections that improve software maintainability (Basili, 1990; Sneed, 1995).
2. Software reengineering for reuse (compositional reuse) salvages selected portions of legacy systems for rehabilitation to enable off-the-shelf reuse in assembling new applications (Arnold and Frakes, 1992; Cimitile, 1992).

Both types of reengineering for reuse share a common life cycle (Patterson, 1995a) (shown in Fig. 1.24 and discussed later) for reengineering to an object-oriented software architecture.

Reengineering Concept Development			Reengineering Product Development						Deployment		
			Reverse Engineering			Forward Engineering					
Feasibility	Scope	Level of Reengineering	from Code to Design	from Design to Spec	Indentification of Candidate Objects	Respecification	Design	Coding and Unit Testing	Software Integration and Testing	System Integration and Testing	Production and Deployment

Figure 1.24 A software reengineering life cycle.

The life cycle is divided into three successive phases: reengineering concept development, reengineering product development, and deployment. During the concept development phase, reengineering is considered as an alternative to new software development. Considerations of scope and of level of reengineering allow planning and cost estimation prior to the development phase.

Reengineering product development proceeds according to the scope and level of reengineering planned in the previous phase. Reverse engineering of the old software is followed by forward engineering to create a new product. During the reverse engineering stage, products are recreated at the design and specification levels as needed to recapture implementation decisions that may have been lost over the lifetime of the legacy software. During the entire reverse engineering stage, candidate objects are repeatedly created, modified, or deleted as necessary to provide the basis of an object-oriented design for the forward engineering stage.

During the forward engineering stage, the candidate objects from the reverse engineering stage are used to create an object-oriented specification and design. Implementation through coding and unit testing complete the development phase. During the deployment phase, software integration and testing, followed by system integration and testing, allow production and deployment to proceed.

Software reengineering is often associated with business process reengineering (Davenport, 1993; Hammer and Champy, 1993; Morris and Brandon, 1993). A study by Patterson (1995b) shows that there is a reciprocal relationship between business process reengineering and software reengineering. The enabling role of information technology makes possible the expansion of the activities of business processes. Moreover, changes in support software may influence changes in the business process. In particular, changes in support software make the software more useful or less useful to a given business process, so that the business process (or, indeed, the software) must adapt. Changes in the potential for software functionality that are due to improvements in the technology may enable changes in business processes, but must not drive them. In general, successful technology follows, rather than leads, humans and organizations.

Similarly, changes in the business process create changes in the requirements for support software. However, this cause-and-effect relationship between business process reengineering and software reengineering cannot be generalized and is inherently unpredictable. Each case requires independent analysis. The effect of business process reengineering on software can range from common perfective maintenance to reconstructive software reengineering. New software may be required in the event that the domain has changed substantially. Because the software exists to automate business process functions, the purpose of the support software can be identified with the functions making up the process. Therefore, reengineering the process at the function level will in general always require reengineering the software at the purpose level. This is equivalent to changing the software requirements. Software reengineering can be the result of business process reengineering, or it can be the result of a need to improve the cost-to-benefit characteristics of the software. An important example of software reengineering that may have little impact on the business process is the case of reengineering function-oriented software products into object-oriented products, thereby choosing the more reactive paradigm to reduce excessive cost due to poor maintainability.

There are many levels of business process reengineering and of software reengineering, ranging from redocumentation to using business process reengineering as a form of maintenance. In both there is a continuum between routine maintenance (minor engineering) and radical, revolutionary reengineering. At both ends of the spectrum, change should be engineered in a proactive, not a reactive, manner. As Sage (1995) notes, reengineering "must be top down directed if it is to achieve the significant and long-lasting effects that are possible. Thus, there should be a strong purposeful and systems management orientation to reengineering." Additional commentaries on systems reengineering are presented in Chapter 23.

1.7.4 Knowledge-Based Software Engineering

Lowry and Duran (1989), in assessing the adequacy of the waterfall model, cite the lack of adequate support for incremental and evolutionary development, such as many artificial intelligence applications. The spiral model is much more natural, especially as computer-aided software engineering (CASE) tools shorten the production cycle for the development of prototypes. In terms of the spiral model (Fig. 1.18), the amount of time needed to complete one turn has been shortened for many of the turns through the use of CASE tools. A potentially greater savings can be realized by reducing the number of turns as well through the development and use of knowledge-based tools. Lowry (1992) believes that much of the process of developing software will be mechanized through the application of artificial intelligence technology. Ultimately, the specification to design to code process will be replaced by domain-specific specification aids that will generate code directly from specifications. This interesting vision is based on much current reality, not only in the CASE arena, but also in the area of domain-based reuse repositories (Gaffney and Cruickshank, 1992; Kozaczynski, 1990; Moore and Bailin, 1991). In terms of the life cycle implications, the waterfall will become shorter, and the spiral will have fewer turns.

1.8 CONCLUSION

Use of the engineering term *life cycle* is relatively new, although the concept is as old as humankind. One of the earliest uses of the term was Juran's reference in 1951 to life-cycle costs, which he referred to as a military term used to describe the operational costs of a system (Juran et al., 1951), or the cost of ownership. The term "life-cycle cost" has come to include the cost of system acquisition (Aslaksen and Belcher, 1992; Blanchard, 1991; Blanchard and Fabrycky, 1998). Today the term "life cycle" is synonymous with "process model." Because of engineering failures, some spectacular, on many systems engineering projects, the engineering community has given a great deal of attention to both the structure and the function of life cycles and, to a lesser degree, their purpose.

In this chapter we have examined basic types of life-cycle models and their variants: the waterfall, and the use of feedback and prototyping to improve the system definition; the V model; the incremental and evolutionary approaches, such as the repeated waterfall with feedback, the spiral model, and rapid development; concurrent engineering; and models that are specific to domains or organizations requiring

process measurement for certification and for process improvement. We have identified the organization with the purpose of the life cycle and noted the interaction of process models with other dynamic organizational forces. We have discussed the life cycle as a tool for the division of resources: for the division and interrelationships of the human resources, through formation of structural and interpersonnel relationships both within and among organizations; for the division and interrelationships of types of business process; and for division of business process into phases, phases into steps, and for the orderly completion of steps through sequencing of the definition, development, and integration of partial products. We have discerned several fundamental relationships along a continuum of four fundamental types of acquisition: commercial off-the-shelf, performance-based contracting, cost-plus contracting, and in-house development. Finally, we have recognized several trends in life cycles, such as those related to reengineering, reuse, and automated software generation.

It seems apparent that we are moving to a more domain-specific way of conducting the business of systems engineering. We are seeing confirmation in the software reuse area, also in the ISO-9000 standard and the CMM family of standards. We

TABLE 1.1. Five Frequently Cited Reasons for Systems Engineering Failures

Complaint	Possible Cause	Recommended Approach
1. Ineffective management	Poor leadership, which could be caused by many factors related to organizational processes Shortage of domain knowledge, which could be caused by either unfamiliarity with the domain or loss of information between organizations	Use in-house development, or development by a single organization Use an organization with a mature process for the domain Inject initial or early use of life cycle with proven success in the domain
2. Uncertain funding	Long development cycles during which no product is deployed or demonstrated	Use a life cycle that fields or demonstrates an early prototype Automate processes
3. Unmanaged complexity	Addressed by division of labor; division of the problem; division of processes into phases Inadequate process controls	Use a life cycle that divides the problem as well as the resources into manageable parts
4. Poor requirements	Addressed by spiral and rapid development models, which enable incremental and evolutionary approaches; domain engineering	Use a life cycle that repeats or extends the requirements cycle (e.g., concurrent engineering, spiral model)
5. Poor specification or other products	Loss of information caused by using several independent organizations	Use acquisition strategy that uses a single organization, (e.g., in-house development)

are seeing a move to associate the process with the product requirements from the domain of choice. Ultimately, this leads to automation of the kind that Lowry (1992) is advocating for automated software generation from requirements. But, meanwhile, we are specifying a process, which in the most mature domains is most likely to be well defined, when we specify our requirements. This is good news, since a mature process moves system creation a step farther along the path from art to engineering. From an organization point of view this is also good, because an organization is liberated in a sense to optimize processes according to its own strengths. This is the same sense in which process standards are sometimes regarded as denigrating to quality.

There are several persistent themes associated with systems engineering failures, especially in large systems, including the five addressed in Table 1.1. Many of these problems have their roots in the life-cycle issues that have been discussed in this chapter. We have the luxury of having observed and having learned from many of the mistakes of the past. We conclude that systems engineering life cycles are purposeful tools that should not be substituted for the systems approach, but rather should be used as part of an integrated systems approach that addresses not only the what and the how, but also the why.

REFERENCES

Ackoff, R. L. (1981). *Creating the Corporate Future*. Hoboken, NJ: Wiley.

American National Standards Institute/Institute of Electrical and Electronics Engineers (ANSI/IEEE). (1983). *IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE.

Arnold, R. S., and Frakes, W. B. (1992). Software reuse and re-engineering. In: *Software Reengineering* (R. S. Arnold, ed.), pp. 476–484. Los Alamitos, CA: IEEE Computer Society Press.

Aslaksen, E., and Belcher, R. (1992). *Systems Engineering*. Englewood Cliffs, NJ: Prentice Hall.

Barnes, B. H., and Bollinger, T. B. (1991). Making reuse cost-effective. *IEEE Software* **8**(1):13–24.

Basili, V. R. (1990). Viewing maintenance as reuse-oriented software development. *IEEE Software* **7**(1):19–25.

Bass, B. M., and Stogdill, R. M. (1990). *Handbook of Leadership: Theory, Research, and Managerial Applications*, 3rd ed. New York: Macmillan (Free Press).

Blake, R. R., and Mouton, J. S. (1961). *Group Dynamics—Key to Decision Making*, Chap. 8, pp. 97–112. Houston, TX: Gulf Publishing Company.

Blake, R. R., and Mouton, J. S. (1980). Power styles within an organization. In: *Models for Management: The Structure of Competence* (J. A. Shtogren, ed.), pp. 60–79. The Woodlands, TX: Teleometrics Int.

Blanchard, B. S. (1991). *System Engineering Management*. Hoboken, NJ: Wiley.

Blanchard, B. S., and Fabrycky, W. J. (1998). *Systems Engineering and Analysis*, 3rd ed. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. W. (1981). *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B. W. (1990a). Verifying and validating software requirements and design specifications. In: *System and Software Requirements Engineering* (M. Dorfman and R. H. Thayer, eds.), pp. 471–484. Los Alamitos, CA: IEEE Computer Society Press. Reprinted from *IEEE Software* **1**(1):75–88 (1984).

Boehm, B. W. (1990b). A spiral model of software development and enhancement. In: *System and Software Requirements Engineering* (M. Dorfman and R. H. Thayer, eds.), pp. 513–527. Los Alamitos, CA: IEEE Computer Society Press. Reprinted from *Software Engineering Project Management*, pp. 128–142. Los Alamitos, CA: IEEE Computer Society Press, 1987.

Bradford, L. P., Stock, D., and Horwitz, M. (1980). How to diagnose group problems. In: *Models for Management: The Structure of Competence* (J. A. Shtogren, ed.), pp. 373–390. The Woodlands, TX: Teleometrics Int.

Briner, W., and Hastings, C. (1994). The role of projects in the strategy process. In: *Global Project Management Handbook* (D. I. Cleland and R. Gareis, eds.), Chap. 15, pp. 1–24. New York: McGraw-Hill.

Brooks, F. P. Jr. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Reading, MA: Addison-Wesley. (Reprinted with corrections, January 1982.)

Bruner, J. S. (1960). *The Process of Education*. Cambridge, MA: Harvard University Press.

Buck, V. E. (1966). A model for viewing an organization as a system of constraints. In: *Approaches to Organizational Design* (J. D. Thompson, ed.), 1971 paperback ed., pp. 103–172. Pittsburgh, PA: University of Pittsburgh Press.

Buckley, W. F., (1979). Let's define that "leadership" that Kennedy says we need (quoted in Bass and Stogdill, 1990). *Press-Bulletin*, Binghamton, NY, September 22.

Burns, J. MacG. (1978). *Leadership*. New York: Harper & Row.

Carter, D. E., and Baker, B. S. (1992). *Concurrent Engineering: The Product Development Environment for the 1990s.* Reading, MA: Addison-Wesley.

Chandler, A. D. (1962). *Strategy and Structure: Chapters in the History of the Industrial Enterprise*. Cambridge, MA: MIT Press.

Chikofsky, E. J., and Cross, J. H., II. (1990). Reverse engineering and design recovery: a taxonomy. *IEEE Software* **7**(1):11–12.

Cimitile, A. (1992). Towards reuse reengineering of old software. In: *Proceedings of IEEE Fourth International Conference on Software Engineering and Knowledge Engineering*, pp. 140–149. Los Alamitos, CA: IEEE Computer Society Press.

Cleland, D. I., and King, W. R. (1983). *Systems Analysis and Project Management*, 3rd ed. New York: McGraw-Hill.

CMMI Product Team. (2006). *CMMI® for Development, Version 1.2 (CMMI-DEV, V1.2)* (Technical Report No. CMU/SEI-2006-TR-008). Pittsburgh, PA: Software Engineering Institute.

Cooper, R. G. (1992). *Winning at New Products: Accelerating the Process from Idea to Launch*, 2nd ed. Reading, MA: Addison-Wesley.

Chrissis, M. B., Konrad, M., and Shrum, S. (2003). *CMMI®: Guidelines for Process Integration and Product Improvement*. Boston, MA: Addison-Wesley Professional.

Crosby, P. B. (1979). *Quality is Free*. New York: McGraw-Hill.

Davenport, T. H. (1993). *Process Innovation: Reengineering Work Through Information Technology*. Boston: Harvard Business School Press.

Davis, A. M., Bersoff, E. H., and Comer, E. R. (1988a). A strategy for comparing alternative software development life cycle models. *IEEE Trans. Software Eng.* **14**(10):1453–1461.

Davis, A. M., Bersoff, E. H., and Comer, E. R. (1988b). In: *System and Software Requirements Engineering* (R. H. Thayer and M. Dorfman, eds.), pp. 496–503. Los Alamitos, CA: IEEE Computer Society Press.

Davis, K. (1949). *Human Society*. New York: Macmillan.

Defense Acquisition Guidebook (2008). Ch. 11.8, Integrated Product and Process Development, DoDI 5000.02.

Deming, W. E. (1982). *Out of the Crisis*. Cambridge, MA: MIT Press.

Drucker, P. F. (1974). *Management: Tasks, Responsibilities, Practices*. New York: Harper & Row.

Forsberg, K., and Mooz, H. (1991). The relationship of system engineering to the project cycle. In: *Proceedings of the Joint Conference of the National Council for Systems Engineering (NCOSE)/American Society of Engineering Management (ASEM)*, Chattanooga, TN, pp. 57–65.

Forsberg, K., and Mooz, H. (1995). Application of the "Vee" to incremental and evolutionary development. In: *Proceedings of the National Council for Systems Engineering (NCOSE), 5th Annual International Symposium*, St. Louis, MO, pp. 801–808.

Gaffney, J. E. Jr., and Cruickshank, R. D. (1992). A general economics model of software reuse. In: *14th Proceedings of the International Conference on Software Engineering*. New York: Association for Computing Machinery.

Galbraith, J. R., and Nathanson, D. A. (1978). *Strategy Implementation: The Role of Structure and Process*. St. Paul, MN: West Publishing Company.

Grady, J. O. (1993). *System Requirements Analysis*. New York: McGraw-Hill.

Hall, A. D. III. (1977a). Systems engineering from an engineering viewpoint. In: *Systems Engineering: Methodology and Applications* (A. P. Sage, ed.), pp. 13–17. New York: IEEE Press. Reprinted from *IEEE Trans. Syst. Sci. Cybern.* **SSC-1**:4–8 (1965).

Hall, A. D. III. (1977b). Three-dimensional morphology of systems engineering. In: *Systems Engineering: Methodology and Applications* (A. P. Sage, ed.), pp. 18–22. New York: IEEE Press. Reprinted from *IEEE Trans. Syst. Sci. Cybern.* **SSC-5**:156–160 (1969).

Hammer, M., and Champy, J. (1993). *Reengineering the Corporation: A Manifesto for Business Revolution*. New York: HarperCollins.

Hazelrigg, G. A. (1996). *Systems Engineering: An Approach to Information-Based Design*. Upper Saddle River, NJ: Prentice Hall.

Hill, J. D., and Warfield, J. N. (1977). Unified program planning. In: *Systems Engineering: Methodology and Applications* (A. P. Sage, ed.), pp. 23–34. New York: IEEE Press. Reprinted from *IEEE Trans. Syst. Man Cybern.* **SMC-2**:610–621 (1972).

Humphrey, W. S. (1989). *Managing the Software Process*. Reading, MA: Addison-Wesley. (Reprinted with corrections in August 1990.)

Humphrey, W. S. (1995). *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley.

Ince, D. (1994). *ISO 9001 and Software Quality Assurance*. Maidenhead, Berkshire, England: McGraw-Hill Book Company Europe.

International Benchmarking Clearinghouse. (1992). *Assessing Quality Maturity: Applying Baldrige, Deming, and ISO 9000 Criteria for Internal Assessment*. Houston, TX: American Productivity and Quality Center.

International Organization for Standardization/International Electrotechnical Commission. (1995). *Software Life Cycle Processes* (ISO/IEC 12207). Geneva: IOS/IEC.

ISO/IEC. (2004) Software engineering—Guidelines for the application of ISO 9001:2000 to computer software. JTC 1/SC 7 Software and system engineering: International Organization for Standardization. (Available at http://www.iso.org.)

Jackson, M. (1995). *Software Requirements and Specifications*. Wokingham, England: Addison-Wesley.

Johnson, R. A., Kast, F. E., and Rosenzweig, J. E. (1967). *The Theory and Management of Systems*. New York: McGraw-Hill.

Juran, J. M., Gryna, F. M., and Bingham, R. S. (eds.). (1951). *Quality Control Handbook*. New York: McGraw-Hill.

Kast, F. E., and Rosenzweig, J. E. (1970). *Organization and Management: A Systems Approach*. New York: McGraw-Hill.

Kerzner, H. (1992). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 4th ed. New York: Van Nostrand Reinhold.

King, W. R., and Cleland, D. I. (1983). Life cycle management. In: *Project Management Handbook* (D. I. Cleland and W. R. King, eds.), pp. 209–221. New York: Van Nostrand Reinhold.

Kozaczynski, W. (1990). The "catch 22" of reengineering. In: *12th International Conference on Software Engineering*, p. 119. Los Alamitos, CA: IEEE Computer Society Press.

Lowry, M. R. (1992). Software engineering in the twenty-first century. *AI Mag*. **14**:71–87.

Lowry, M. R., and Duran, R. (1989). A tutorial on knowledge-based software engineering. In: *The Handbook of Artificial Intelligence* (A. Barr, P. R. Cohen, and E. A. Feigenbaum, eds.), Vol. 4. Reading, MA: Addison-Wesley.

McGarry, F. E., Page, J., Card, D., Rohleder, M., and Church, V. (1984). *An Approach to Software Cost Estimation*. National Aeronautics and Space Administration Software Engineering Laboratory Series (SEL-83-001). Greenbelt, MD: Goddard Space Flight Center.

Melcher, B. H., and Kerzner, H. (1988). *Strategic Planning: Development and Implementation*. Blue Ridge Summit, PA: TAB Books.

Michaels, J. V., and Wood, W. P. (1989). *New Dimensions in Engineering. Design to Cost*. Hoboken, NJ: Wiley.

Moore, J. M., and Bailin, S. C. (1991). Domain analysis: framework for reuse. In: *Domain Analysis and Software System Modeling* (R. Prieto-Diaz and G. Arango, eds.). Los Alamitos, CA: IEEE Computer Society Press.

Morris, D., and Brandon, J. (1993). *Re-engineering Your Business*. New York: McGraw-Hill.

National Aeronautics and Space Administration (NASA). (1995). *Software Configuration Management Guidebook* (NASA-GB-9503 or NASA-GB-A501). Washington, DC: NASA, Office of Safety and Mission Assurance.

National Aeronautics and Space Administration (NASA). (1996). *Software Process Improvement Guidebook*, NASA Software Engineering Program (NASA-GB-001-95). Greenbelt, MD: Goddard Space Flight Center.

Ogle, W. (1941). De partibus animalium. In: *The Basic Works of Aristotle* (R. McKeon, ed.), pp. 641–661. New York: Random House.

Ould, M. A. (1990). *Strategies for Software Engineering: The Management of Risk and Quality*. Chichester, UK: Wiley.

Patterson, F. G. Jr. (1995a). Reengineerability: metrics for software reengineering for reuse. Unpublished Ph.D. dissertation, George Mason University, Fairfax, VA.

Patterson, F. G. Jr. (1995b). A study of the relationship between business process reengineering and software reengineering. *Inf. Syst. Eng.* **1**(1):3–22.

Paulish, D. J., and Carleton, A. D. (1994). Case studies of software-process-improvement measurement. *IEEE Comput*. **27**:50–57.

Pearce, J. A. II, and Robinson, R. B. Jr. (1985). *Strategic Management: Strategy Formulation and Implementation*, 2nd ed. Homewood, IL: R. D. Irwin.

Rabbitt, J. T., and Bergh, P. A. (1994). *The ISO 9000 Book: A Global Competitor's Guide to Compliance and Certification*, 2nd ed. New York: American Management Association (AMACOM).

Radley, C. F., and Wetherholt, M. S. (1996). *NASA Guidebook for Safety and Critical Software Analysis and Development* (NASA-GB-1740.13–96). Cleveland, OH: National Aeronautics and Space Administration, Glenn Research Center.

Reilly, N. B. (1993). *Successful Systems Engineering for Engineers and Managers*. New York: Van Nostrand-Reinhold.

Rosenau, M. D. Jr. (1990). *Faster New Product Development: Getting the Right Product to Market Quickly*. New York: American Management Association (AMACOM).

Royce, W. W. (1970). Managing the development of large software systems: concepts and techniques. *Proc. WESCON*, pp. 1–70.

Rubey, R. J. (1993). *Software Management Guide*, 3rd ed., Vol. 2. Hill Air Force Base, UT: U.S. Air Force Software Technology Support Center.

Sage, A. P. (1992a). *Systems Engineering*. Hoboken, NJ: Wiley.

Sage, A. P. (1992b). Systems engineering and information technology: catalysts for total quality in industry and education. *IEEE Trans. on Syst. Man Cybern.* **22**(5):833–864.

Sage, A. P. (1995). *Systems Management for Information Technology and Software Engineering*. Hoboken, NJ: Wiley.

Sage, A. P., and Palmer, J. D. (1990). *Software Systems Engineering*. Hoboken, NJ: Wiley.

Shishko, R. (1995) (ed.). *NASA Systems Engineering Handbook (SP-6105)*. Washington, DC: National Aeronautics and Space Administration.

Slade, B. N. (1993). *Compressing the Product Development Cycle*. New York: American Management Association (AMACOM).

Sneed, H. M. (1995). Planning the reengineering of legacy systems. *IEEE Software* **12**(1):24–34.

Software Engineering Institute. (1991). *Capability Maturity Model* (CMU/SE-91-TR-24). Pittsburgh, PA: Carnegie Mellon University.

Sorensen, R. (1996). Adopting MIL-STD-498: the steppingstone to the U.S. commercial standard. *Crosstalk: J. Def. Software Eng.* **9**(3):8–12. (Hill Air Force Base, UT: Software Technology Support Center.)

Stogdill, R. M. (1966). Dimensions of organization theory. In *Approaches to Organizational Design* (J. D. Thompson, ed.), 1971 paperback ed., pp. 1–56. Pittsburgh, PA: University of Pittsburgh Press.

Thome, B. (1993). Definition and scope of systems engineering. In: *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering* (B. Thome, ed.), pp. 1–23. Chichester, UK: Wiley.

U.S. Department of Defense. (1969). *Engineering Management* (MIL-STD-499). Washington, DC: USDoD.

U.S. Department of Defense. (1974). *Engineering Management* (MIL-STD-499A). Washington, DC: USDoD. (Canceled notice 1, February 27, 1995.)

U.S. Department of Defense. (1987). *Software Products Standard* (DoD-STD-1703). Washington, DC: USDoD. (Canceled December 5, 1994.)

U.S. Department of Defense. (1988a). *Defense System Software Development* (DoD-STD-2167A). Washington, DC: USDoD. (Replaced December 5, 1994 by MIL-STD-498.)

U.S. Department of Defense. (1988b). *DoD Automated Information Systems (AIS) Documentation Standard* (DoD-STD-7935A). Washington, DC: USDoD. (Replaced December 5, 1994 by MIL-STD-498.)

U.S. Department of Defense. (1991). *Defense Acquisition Management Policies and Procedures* (DoDI 5000.2) Washington, DC: USDoD.

U.S. Department of Defense. (1992). *Systems Engineering* (MIL-STD-499B). Washington, DC: USDoD. (Draft May 6, 1992; never approved by OSD; final decision to not approve was in April 1994.)

U.S. Department of Defense. (1993). *Automated Information System (AIS) Life-Cycle Management (LCM) Process, Review, and Milestone Approval Procedures* (DoDI 8120.2). Washington, DC: USDoD.

U.S. Department of Defense. (1994). *Software Development and Documentation* (MIL-STD-498). (December 5, 1994 ed.). Washington, DC: USDoD.

Warfield, J. N., and Hill, J. D. (1972). *A Unified Systems Engineering Concept*. Columbus, OH: Battelle Memorial Institute.

Wheatley, M. J. (1992). *Leadership and the New Science*. San Francisco: Berrett-Koehler Publishers.

Whytock, S. (1993). The development life-cycle. In: *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering* (B. Thome, ed.), pp. 81–96. Chichester, UK: Wiley.

Williamson, O. E. (1970). *Corporate Control and Business Behavior: An Inquiry into the Effects of Organization Form on Enterprise Behavior*. Englewood Cliffs, NJ: Prentice Hall.

Yeh, R. T. (1992). Notes on concurrent engineering. *IEEE Trans. Knowl. Data Eng.* **4**(5):407–414.