Chapter 1

Introduction to Systems Engineering

1.1 INTRODUCTION

A *system* is commonly defined to be "a collection of hardware, software, people, facilities, and procedures organized to accomplish some common objectives." The stakeholders for the system hold these objectives. Never forget that the system being addressed by one group of engineers is the subsystem of another group and the supersystem of yet a third group. The objective of the engineers for a system is to provide a system that accomplishes the primary objectives set by the stakeholders, including those objectives associated with the creation, production, and disposal of the system. To accomplish this engineering task, the engineers must identify the system's stakeholders throughout the system's life cycle and define the objectives of all of these stakeholders. These objectives typically address the triad of cost, schedule, and performance – cheaper, faster, and better.

A major characteristic of the engineering of systems is the attention devoted to the entire life cycle of the system. This life cycle has been characterized as "birth to death," and "lust to dust." That is, the *life cycle* begins with the gleam in the eyes of the users or stakeholders, is followed by the definition of the stakeholders' needs by the systems engineers, includes developmental design and integration, goes through production and operational use, usually involves refinement, and finishes with the retirement and disposal of the system. Ignoring any part of this life cycle while engineering the system can lead to sufficiently negative consequences, including

The Engineering Design of Systems: Models and Methods, Third Edition.
Dennis M. Buede and William D. Miller.

Companion website: www.wiley.com/go/engineeringdesignofsystems3e

failure at the extreme. In particular, developing a system that has not adequately addressed the stakeholders' needs leads to failures such as the "highway to nowhere" near San Francisco, which was stopped by political pressure brought to bear by homeowners on the surrounding hills overlooking the bay. The view of the bay that these homeowners enjoyed and thought was an associated right of the property they owned would have been blocked by the highway. Similar commercial failures that did not consider the needs of the stakeholders in sufficient detail include the personal computers IBM PC Jr. and the Apple LISA. This is not to say that the adherence to methods and models put forth in this book or any other will guarantee success or even the absence of failure. Rather the methods and models proposed here do attend to the entire life cycle of the system and provide a process that makes sense, can be tailored to various levels of detail as dictated by the complexity of the system being addressed, and attend to all of the details that many engineers during years of practice in systems engineering have determined to be useful.

The concepts of design and integration are critical to the methods addressed in this chapter and the book. The word *design* is used by many professions (artists, architects, all disciplines of engineering) and is claimed by each.

The *American Heritage Dictionary* [Berube, 1991] defines design as follows:

> de-sign (di-zin') v. - signed, - sign·ing, - signs.—tr. 1. To conceive in the mind; invent: *designed his dream vacation*. 2. To form a plan for: *designed a marketing strategy for the new product*. 3. To have a goal or purpose; intend. 4. To plan by making a preliminary sketch, outline, or drawing. 5. To create or execute in an artistic or highly skilled manner. –intr. 1. To make or execute plans. 2. To create designs. –n. 1. A drawing or sketch. 2. The invention and disposition of the forms, parts, or details of something according to a plan. 3. A decorative or artistic work. 4. A visual composition; pattern. 5. The art of creating designs. 6. A plan; project. 7. A reasoned purpose; intention. 8. Often designs. A sinister or hostile scheme: *He has designs on my job*. . . .

All but the third and eighth definitions for the noun usage will apply at various times during the course of this book. *Design* during the engineering of a system as discussed in this book is the preliminary activity that has the purpose of satisfying the needs of the stakeholders, begins in the mind of the lead engineer but has to be transformed into models employing visual formats in a highly skilled manner for success to be achieved. While this book addresses the engineering methods and models used during the design process, there is always an element of artistry that is required for the design process and the system to be successful.

Integration brings all of the detailed elements of the overall design together through a process of testing (or qualification) to achieve a valid system for meeting the needs of the stakeholders. Engineers of appropriate disciplines perform integration according to the specifications defined by the design of the systems engineers. The integration process involves testing or qualification of both the elements of the system and the system itself to ensure that the system meets the ultimate needs of the stakeholders.

This chapter first provides an overview of the issues and process associated with the engineering of a system. This overview addresses the phases of the system's life

cycle, describes the importance of performing the engineering of a system well, provides a definition for the engineering of a system, introduces the key process model for the engineering of a system called the Vee model, describes the richness of decisions that are inherent in the engineering process, and discusses the diversity of expertise required for this engineering process. Section 1.3 describes process models that have been adopted by the systems engineering community. Architectures play a key role in the engineering of systems and are introduced next. Requirements, Section 1.6, play a major role in the engineering of a system because they serve the role of defining the engineering design problem and capturing the key information needed to describe design decisions. The life cycle of the system is next examined in more detail. Then, the Vee model for engineering a system is described in more detail.

The key method addressed in this chapter is the process used to perform the engineering of systems. Supplementing this discussion of the engineering method are discussions of the key concepts needed to understand the method at an introductory level. This method is presented as a process model; models and modeling are discussed in detail in Chapter 3, so the reader is asked to accept the notion of the process discussion as a discussion of a model until more details on models can be provided in Chapter 3.

1.2 OVERVIEW OF THE ENGINEERING OF SYSTEMS

The development process in systems engineering is commonly viewed [Forsberg and Mooz, 1992; Lake, 1992] as a decomposition (or design) process followed by a recomposition (or integration) process (see Sidebar 1.1). During the decomposition process, the stakeholders' requirements are analyzed and defined in engineering terms and then partitioned into a set of specifications (or specs) for several segments, elements, or components. It is critical that this design process be *broad in perspective* so that nothing is left out and every contingency is considered. Systems engineers must be "big picture" people. Depth is achieved only by much iteration through the design process, as many as are needed until the system's specifications are sufficiently detailed for individual configuration items (CIs) to be built or purchased. This design process defines what the system must do, how well the system must do it, and how the system should be tested to verify and validate the system's performance. To do this, the systems engineers must maintain a very clear focus on the objectives that the system's stakeholders (users, owners, manufacturers, maintainers, trainers, etc.) have defined for the system.

One of many possible representations of the life cycle of a system is shown in Figure 1.1, beginning with the identification of the need for the system and progressing through the retirement of the system. Some of the phases of the life cycle are accomplished in parallel, as the diagram tries to depict; exactly which phases occur in parallel depends upon the type of system, the organization, and the context. For additional details, see Driscoll [2007].

As shown in Figure 1.1, design includes the preliminary system design as well as parts of the identification of need and concept definition. Parts of the identification of need and concept definition include the development of a basic idea and the first

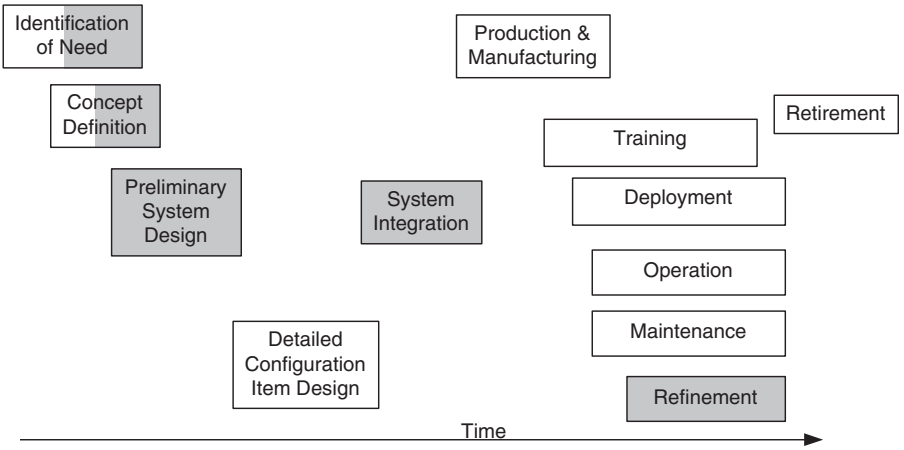

FIGURE 1.1 Phases of the system life cycle.

embodiment of the idea; these two initial activities are often called invention and are usually not part of the engineering of a system. Invention has a heavy technological and scientific focus. The last portions of the identification of need and concept design phases, plus preliminary system design, address the initial or follow-on commercialization of the idea based upon a specific statement of stakeholders' needs.

The products of the design process serve as the inputs to the hardware and software design of detailed configuration item design. The CIs then re-enter the systems engineering process during system integration for integration testing, verification, and validation. Further adjustments to the design occur during the refinement phase. The

SIDEBAR 1.1

The term *systems engineering* dates back to Bell Telephone Laboratories in the 1940s [Schlager, 1956; Hall, 1962; Fagen, 1978]. Fagen [1978] traces the concepts of systems engineering within Bell Labs back to early 1900s and describes major applications of systems engineering during World War II. RCA used the "systems approach" during the research and development of the electronically scanned black and white television [Engstrom, 1957]. In 1943, the National Defense Research Committee established a Systems Committee with Bell Laboratories support to guide a project called C-79, the first task of which was to improve the communication system of the Air Warning Service. An unpublished chapter on systems engineering in the Bell system suggested that the first use of the phrase "systems engineering" within the Bell system was in a memo in the summer of 1948. Systems engineering was identified as a unique function in the organizational structure of Bell Laboratories in 1951. Personnel at Massachusetts Institute of Technology (MIT) worked with Bell Labs on radars

during World War II. Ivan Getting at MIT discovered that electrical noise caused two components of a fire control system to behave differently when they worked together than when they worked independently. His group at MIT became a systems integrator for the Navy after the war. As a result of these efforts, the following statement appeared in what is called the Ridenour Report (Research and Development in the United States Air Force dated September 21, 1949: "The role of systems engineering should be substantially strengthened, and systems projects should be attacked on a 'task force' basis by teams of systems and components specialists organized on a semi-permanent basis." [Johnson, 2002]

Involvement in the earliest intercontinental ballistic missile (ICBM) program, starting with Atlas, is the most well known of early systems engineering activities. Simon Ramo and Dean Wooldridge formed a company (R-W) in 1954 to perform systems engineering for the Air Force's ICBM program. In 1956 R-W and the Air Research and Development Command (under direction from Bernard Schriever) reached a legal definition of systems engineering: (1) The solution of interface problems among all weapon system subsystems to ensure technical and schedule compatibility of the systems as a whole. (2) The surveillance over detailed subsystem and over all weapon design to meet Air Force required objectives. (3) The establishment and revision of program milestones and schedules, and monitoring of contractor progress in maintaining schedules, consistent with sound technical judgment and rapid advancement of the state of the art [Johnson, 2002]. R-W later became TRW.

Hall [1962] asserts that the first attempt to teach systems engineering as we know it today came in 1950 at MIT by Mr. Gilman, Director of Systems Engineering at Bell. The first book on Systems Engineering was written by Goode and Machol in 1957, titled *System Engineering – An Introduction to the Design of Large-Scale Systems*.

Hall [1962] defined systems engineering as a function with five phases: (1) system studies or program planning; (2) exploratory planning, which includes problem definition, selecting objectives, systems synthesis, systems analysis, selecting the best system, and communicating the results; (3) development planning, which repeats phase 2 in more detail; (4) studies during development, which includes the development of parts of the system and the integration and testing of these parts; and (5) current engineering, which is what takes place while the system is operational and being refined.

The RAND Corporation was founded in 1946 by the U.S. Air Force and created *systems analysis*, which is certainly an important part of systems engineering.

The Department of Defense entered the world of systems engineering in the late 1940s with the initial development of missiles and missile-defense systems [Goode and Machol, 1957].

Paul Fitts addressed the allocation of the system's functions to the physical elements of the system in the late 1940s and early 1950s [Fitts, 1951].

There is special bibliography at the end of the book devoted to historical references.

life cycle phases associated with the engineering of the system are shaded in Figure 1.1. The term *concurrent engineering* simply means that the systems engineering process should be done with all of the phases (and their associated requirements) of the system life cycle in mind [Prasad, 1996]. This notion of concurrent engineering is a key concept addressed in this book.

The importance of systems engineering is highlighted by examining a generally accepted relationship between the phases of the system life cycle and the commitment versus the incursion of costs. The time associated with the system's life cycle is plotted on the *x*-axis; note that the time increments are notional and should not be interpreted as equal to the relative length of the four stages being addressed. See Prang [1992] for an illustration based on computer boards. (Prang is also referenced in Scheiber [1995].) Figure 1.2 shows the major phases of the system life cycle on the horizontal axis. The curves represent the cost committed, based upon engineering design decisions, and the cost incurred, based upon actual expenditures. As can be seen, about 80% of the cost of the system is committed by the end of design and integration, while only about 20% of the actual cost for the system has been spent. Obviously, mistakes made in the front end of the system life cycle can have substantially negative impacts on the total cost of the system and its success with the users and bill payers.

There have been many definitions of systems engineering put forward since the 1950s when systems engineering became a profession. Table 1.1 provides several of

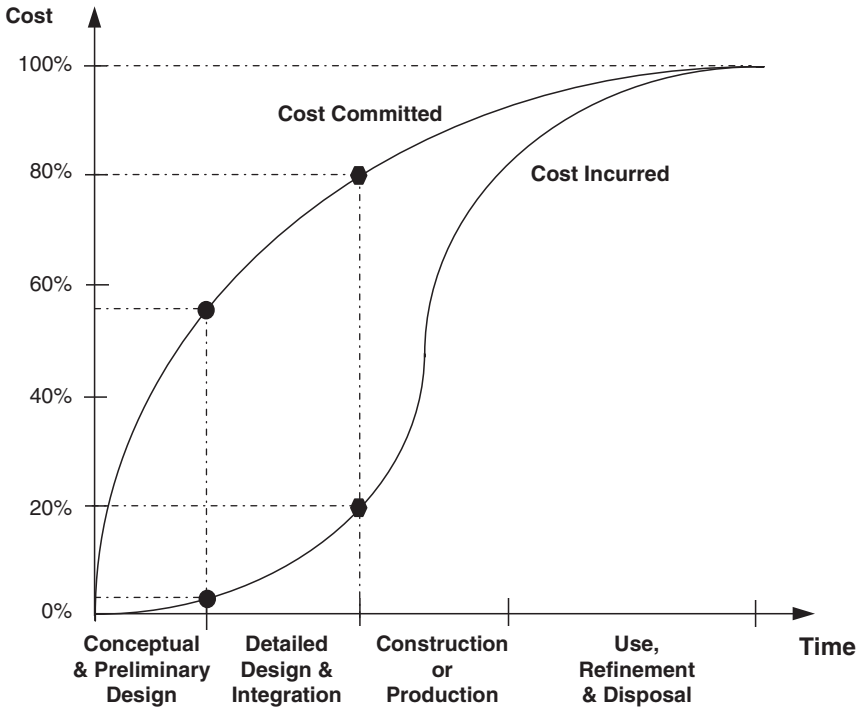

FIGURE 1.2 Cost commitment and incursion in the system life cycle.

TABLE 1.1 Definitions of Systems Engineering

Source	Definitions of Systems Engineering
Mil-Std 499A, 1974	The application of scientific and engineering efforts to (1) transform an operational need into a description of system performance parameters and a system configuration through the use of an iterative process of definition, synthesis, analysis, design, test, and evaluation; (2) integrate related technical parameters and ensure compatibility of all related, functional, and program interfaces in a manner that optimizes the total system definition and design; (3) integrate reliability, maintainability, safety, survivability, human, and other such factors into the total technical engineering effort to meet cost, schedule, and technical performance objectives
Sailor, 1990	Both a technical and a management process; the technical process is the analytical effort necessary to transform an operational need into a system design of the proper size and configuration and to document requirements in specifications; the management process involves assessing the risk and cost, integrating the engineering specialties and design groups, maintaining configuration control, and continuously auditing the effort to ensure that cost, schedule, and technical performance objectives are satisfied to meet the original operational need
Sage, 1992	The design, production, and maintenance of trustworthy systems within cost and time constraints
Forsberg and Mooz, 1992	The application of the *system analysis and design process* and the *integration and verification process* to the logical sequence of the *technical aspect of the project life cycle*
Wymore, 1993	The intellectual, academic, and professional discipline, the primary concern of which is the responsibility to ensure that all requirements for a bioware/hardware/software system are satisfied throughout the life cycle of the system
Mil-Std 499B Draft, 1993	An interdisciplinary approach encompassing the entire technical effort to evolve and verify an integrated and life-cycle-balanced set of system people, product, and process solutions that satisfy customer needs. Systems engineering encompasses (a) the technical efforts related to the development, manufacturing, verification, deployment, operations, support, disposal of, and user training for system products and processes; (b) the definition and management of the system configuration; (c) the translation of the system definition into work breakdown structures; and (d) development of information for management decision making
INCOSE, 1999	An interdisciplinary approach and means to enable the realization of successful systems

INCOSE is the International Council on Systems Engineering, a professional society of systems engineers. INCOSE's definition of a system is an interacting combination of elements, viewed in relation to function.

these definitions. There are two important trends to note over the 20-year span of these definitions. First, the role of management in the systems engineering process is made explicit in the definitions from the 1990s. Second, the three pillars of engineering success (cost, schedule, and technical performance) from the 1970s evolve to concerns over the life cycle, namely, concurrent engineering.

The *American Heritage Dictionary* [Berube, 1991] defines engineering as follows:

> The application of scientific and mathematical principles to practical ends such as the design, construction, and operation of efficient and economical structures, equipment, and systems.

The following definitions of engineering and the engineering of systems are adopted here:

Engineering: discipline for transforming scientific concepts into cost-effective products through the use of analysis and judgment.

Engineering of a System: engineering discipline that develops, matches, and trades off requirements, functions, and alternate system resources to achieve a cost-effective, life-cycle-balanced product based upon the needs of the stakeholders.

Figure 1.3 shows the design and integration process as a "Vee" with the emphasis of this model of the engineering process for a system being on the activities that the

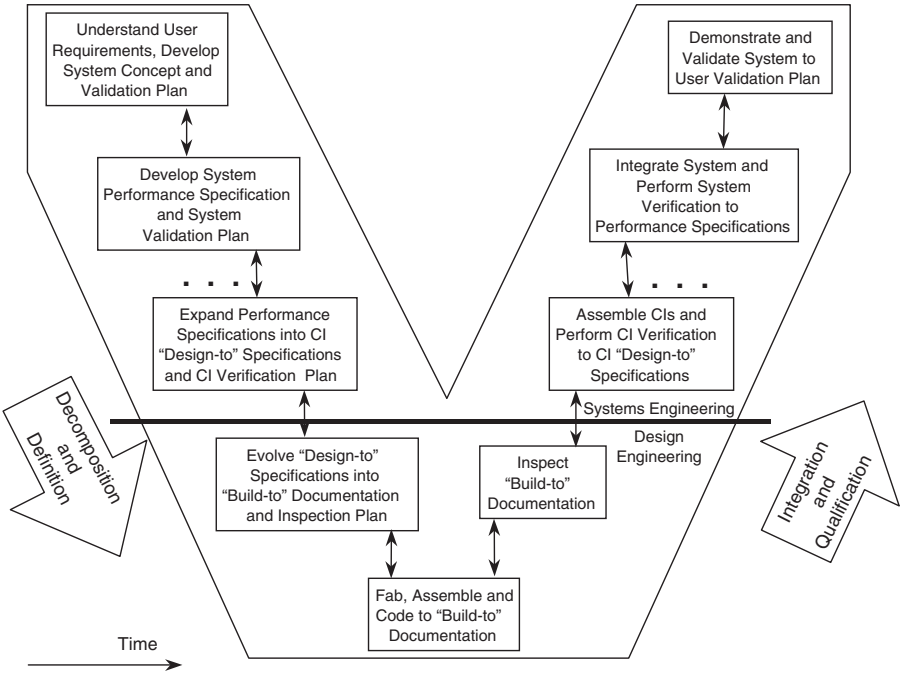

FIGURE 1.3 Systems engineering "Vee." (After Forsberg and Mooz [1992].)

engineers perform. The left or decomposition side of the Vee coincides with the three phases at the beginning of the life cycle from Figure 1.1. Time proceeds from left to right in Figure 1.3, just as it did in Figure 1.1. The process is initiated at the top left of the Vee with the definition of the operational need of the stakeholders. The focus of the decomposition and definition process (or design) is the movement from an operational need to system-level requirements to specifications for each component to the specifications (or specs) for each CI. Since time is moving from left to right in Figure 1.3, parallel work on high- and low-level design activities is not only permitted but also encouraged. The iterative nature of this design process, from high-level issues such as stakeholders' requirements to low-level issues such as component and CI design, is accomplished by moving vertically in the Vee over short increments of time. This vertical movement during the design process is critical to success and has been observed in studies of expert designers [Guindon, 1990]. Note that this Vee model does not emphasize the interaction with the stakeholders even though that interaction is assumed to occur in order to enable the engineering processes depicted in the Vee model.

The horizontal line, drawn just under the middle intersection of the Vee in Figure 1.3, depicts the hand off of the final products of the design process, the CI specs, to the discipline (or design) engineers, those engineers whose orientation is electrical, mechanical, chemical, civil, aerospace, computer science, and the like and whose job is to produce a physical entity. This dividing line can be drawn higher or lower to signify decreasing or increasing overlap between design and integration activities. As the dividing line is drawn in Figure 1.3, the sloping lines of the middle portion of the Vee can be extended until they meet the dividing line, with the resulting very modest overlap between design and integration. If the dividing line is raised above the intersection of the sloping lines of the Vee, there would be no intersection of design and integration. This complete separation of design and integration is often sought in practice to enhance contractual relationships between procurer and supplier of the system; however, this separation negatively impacts the schedule and cost associated with the development of the system. There is significant integration and qualification activity that should take place during design, as is discussed in Chapter 11. In many systems engineering activities, the horizontal dividing line between systems engineering and the discipline engineers is drawn significantly lower than shown in Figure 1.3.

The right-hand side of the Vee depicts the integration and qualification activities of the engineering of a system. Integration involves the assembly of the CIs into components, the assembly of lower level components into higher level components, and the assembly of high-level components into the system. All of this assembly involves testing (or qualification) of the newly assembled system elements to determine whether the assembled element meets the set of requirements (or spec) that the design phase had established for that element; this qualification is called verification. Finally, after the system is verified against the system requirements, the system must be validated. After validation, the stakeholders determine whether the system is acceptable. Naturally, there are problems throughout this process that require modifications to be made either to the design of the elements of the system or to the requirements that were developed during design. Recall that time is running from left to right in Figure 1.3; the

TABLE 1.2 Race Car Example of Requirements and Tests

Operational Need or Mission Requirements: Partially Validated by Operational Test (Proven by Real-World Experience)	System-Level Requirements: Verified by System-Level Tests	Component-Level Requirements: Verified by Component-Level Tests
• Win the Indianapolis 500 • Pretrial average speed of 215 mph • Average speed in the "500" of 190 mph	• Top speed of X mph • Acceleration in all directions, "g–g" space • Average standard pit time of *Y* s.	• Engine horsepower of *x* Btu • Body's drag coefficient of *y* • Range per tank of gas of *z* mi

Vee process allows for the low level of verification of CIs to be happening in parallel with some high-level validation and even acceptance activities.

A sample of the movement from operational need to CI specs is given for a race car in Table 1.2. The first column states the operational need or mission requirement: Win the Indianapolis 500. Associated with this need are stakeholders' requirements concerning the pretrial average speed and the average speed during the race with the expected number of yellow flags and pit stops (note that the numbers in Table 1.2 are notional and are not accurate reflections of race conditions). System-level requirements can then be derived that are more meaningful during engineering. As an example, the key system-level requirement involves the *g–g* space of a vehicle [Milliken and Milliken, 1995]. Race cars, when driven by experienced drivers, are always changing velocity in speed or direction. (Recall that speed is the velocity you are traveling in your direction of travel. But when traveling around a curve, you also have a component of velocity perpendicular to your direction of travel.) Therefore, the acceleration ability of the car in both longitudinal and lateral directions (see Fig. 1.4) is critical in the design process. Figure 1.4 portrays the *g–g* curve for a single car driven by three racers (charts a–c); the bottom right space (chart d) is the inferred *g–g* space of the vehicle. Finally, each of these system-level requirements is "flowed-down" to component-level requirements, such as the engine's horsepower and the drag coefficient of the body of the race car. (Note that the true values of these parameters are closely guarded secrets of racing teams.) This process continues until the requirements for CIs are defined, establishing a hierarchy of requirements, from mission or need down to the CIs.

The system integration process starts during the decomposition and definition (or design) process. As part of design, the integration and qualification plans are developed. The purpose of qualification is the verification and validation of the system's design. *Verification* addresses the following question: Does the component, element, segment, or system meet its requirements, or have we built the component, . . . , system right? On the other hand, *validation*, which is often combined with acceptance testing, demonstrates that the system satisfies the users' needs, or have

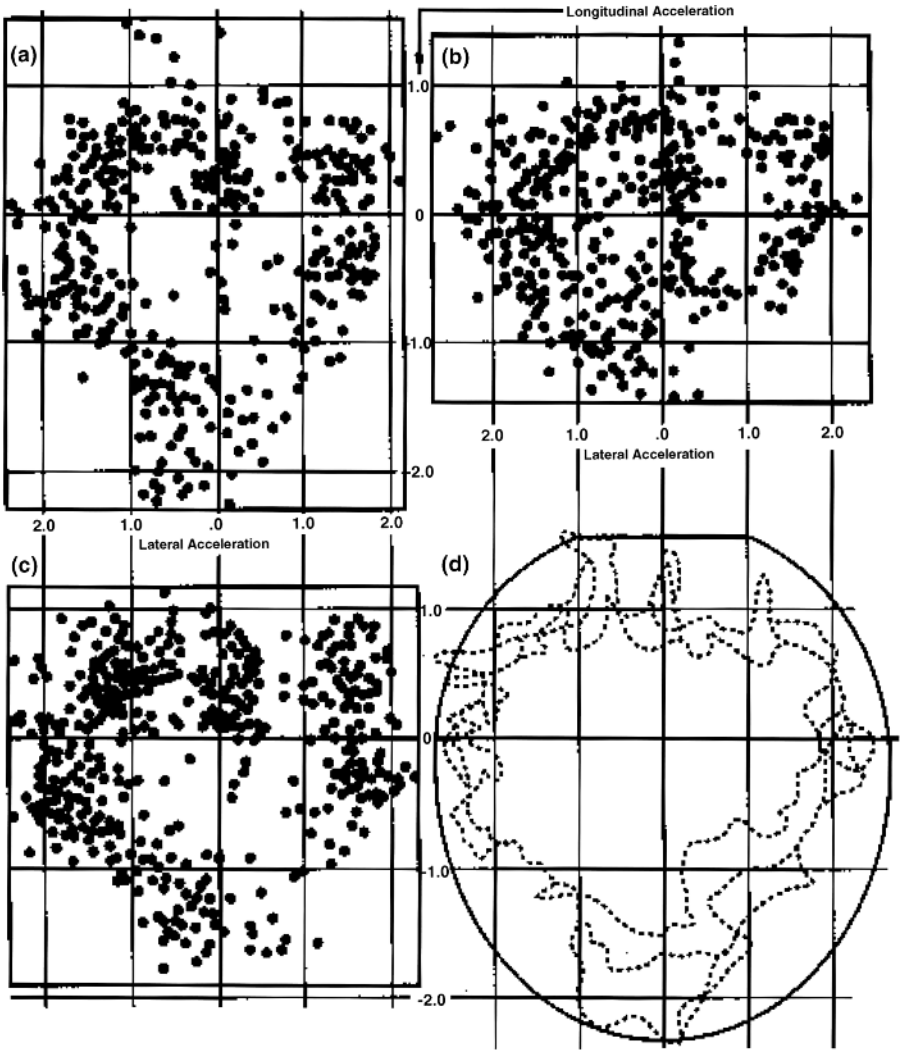

FIGURE 1.4 "g–g" design region for a race car. (From Milliken and Milliken [1995].)

we built the right system? Note that as verification moves farther from the CIs and closer to the system, it is not possible to conduct enough testing to prove anything statistically. Demonstration is often the best that can be done. It is expected, though not desired, that there will be issues and problems that arise as part of this qualification process. Decisions must be made concerning relaxation of requirements versus design changes to specific CIs and components. During the design phase, integration activities should be planned to maximize the effectiveness of qualification within the resources and time available. These planned activities are then carried out during integration, with

adaptations as needed. There should have been some thought given during design about what the most likely adaptations would be so that the integration phase has sufficient, built-in flexibility.

To be successful, the engineering design of systems must embrace the notion that many decisions are made during the development process. This is not a controversial position to take. However, adopting the notion that these decisions should be made via

TABLE 1.3 Sample of Decisions Made during System Design

Development Phase	Examples of Decisions in Systems Engineering
Conceptual design	• Should a conceptual design effort be undertaken?
	• Which system concept (or mixture of technologies) should be the basis of the design?
	• Which technology for a given subsystem should be chosen?
	• What existing hardware and software can be used?
	• Is the envisioned concept technically feasible, based on cost, schedule, and performance requirements?
	• Should additional research be conducted before a decision is made?
Preliminary design	• Should a preliminary design effort be undertaken?
	• Which specific physical architecture should be chosen from several alternatives?
	• To which physical resource should a particular function be allocated?
	• Should a prototype be developed? If so, to what level of reality?
	• How should validation and acceptance testing be structured?
Full-scale design	• Should a full-scale deign effort be undertaken?
	• Which configuration items should be bought instead of manufactured?
	• Which detailed design should be chosen for a specific component given that one or more performance requirements are critical?
Integration and qualification	• What is the most cost-effective schedule for implementation activities?
	• What issues should be tested?
	• What equipment, people, and facilities should be used to test each issue?
	• What models of the system should be developed or adapted to enhance the effectiveness of integration?
	• How much testing should be devoted to each issue?
	• What adaptive (fallback testing in case of a failure) testing should be planned for each issue?
Product refinement	• Should a product improvement be introduced at this time?
	• Which technologies should be the basis of the product improvement?
	• Which redesign is best to meet some clearly defined deficiency in the system?
	• How should the refinement of existing systems be implemented given the schedule, performance, and cost criteria?

a rational, explicit process is not consistent with much of the current practice in the engineering of systems. Table 1.3 lists a sample of the many categories of development decisions. Chapter 14 provides a method for addressing these decisions. An important philosophical point in decision making is that decisions have to be made with the best information available at the time, realizing that the outcomes associated with the decision remain uncertain when the decision is made. Therefore, distinguishing between a good decision and a good outcome is important. The material in this book will also distinguish between the level of details needed to make decisions in the engineering of a system and the level of details needed to ensure proper implementation of the system's components and CIs.

In order to accomplish this difficult job of engineering a system, people with many different specialties must be involved on the systems engineering team. The stakeholders are central to the success of this effort and need to be represented on the systems engineering team. Discipline engineers with knowledge of the technologies associated with the system's concept are needed to provide the expertise needed for design and integration decisions throughout development. Discipline engineers come not only from traditional engineering fields such as electrical, mechanical, and civil but also from the social sciences to address psychological, informational, physical, and cultural issues. In addition, systems engineers who model and estimate system-level parameters such as cost and reliability fall in the category of discipline engineers. Analysts skilled in modeling and simulation, more and more of which is done on the computer rather than with scaled-down mock-ups of the system, are also important members of this team. Engineers skilled in the processes (or methods) of systems engineering form the nucleus of this collection of skills. *These processes and associated models are the nucleus of this book*. Finally, managers that are in charge of meeting cost and schedule milestones need to be present. These five disciplines are depicted in the Venn diagram in Figure 1.5. Sidebar 1.2 describes Joe Shea, who was hired by the National Aeronautics and Space Administration (NASA) in 1961 to take charge of systems engineering for the Office of Manned Space Flight.

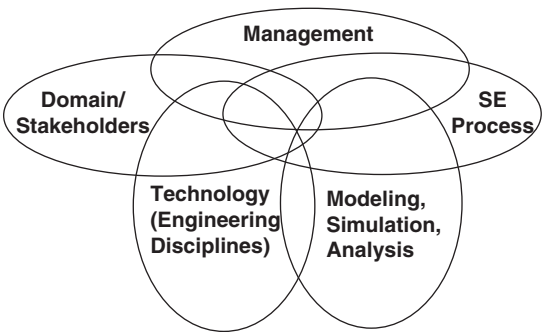

FIGURE 1.5 Expertise required on the engineering team for a system.

SIDEBAR 1.2

It was 1943 when he graduated (from high school), wartime, and Shea heard about a special Navy program that would send him to college. . . . Then the Navy sent him to M.I.T., and after that to the University of Michigan. . . .

For the next several years Shea moved back and forth between Michigan, where he eventually obtained his engineering doctorate, and Bell Labs. It was an educational odyssey that took him from engineering mechanics to electrical engineering to theoretical mathematics to physics to inertial guidance. "The nouns change but the verbs remain the same" became one of Shea's sayings as he went from one specialty to another.

Then in 1956 Shea found out how it all fit together. At the age of twenty-nine, Shea was named systems engineer for the radio guidance project connected with the Titan I. "I didn't know what 'systems engineer' meant," Shea said, but he learned quickly, traveling around to the subcontractors on the Titan I, becoming a member of the small fraternity of engineers who were coming of age in this new field. At night after work they gather at a bar near the plant where they had been working that day. They didn't even drink that much, Shea recalled, they were so busy talking—about testing, grounding, vibrational spectrums, weights, stability, electrical interfaces, guidance equations, all the myriad elements of the system that some lucky guy, like a systems engineer, got to orchestrate.

By 1959 Shea had acquired enough of a reputation within the ballistic missile fraternity for General Motors to hire him to run the advanced development operation for its A.C. Sparkplug Division, which was trying to wedge its way into the missile business. Shea was in charge of preparing a proposal for the inertial guidance contract for the Titan II. After the proposal won, Shea went back to administering the advanced development office. But a year later, in September 1960, the contract he had won was six months behind and Shea was called away to rescue it.

Shea began to discover that he had a knack for leading. His was not a gentle style, but if he was tough on people who fell short, he was generous and loyal to those who didn't. . . . It didn't make any difference what your specialty was. Shea's maxim was that if you understood it, you could make him understand it—and once he did, you never had to explain it again. The only problem was keeping up.

It was about this time that Shea discovered the uses of what he would come to call his "controlled eccentricity." When he was still at Bell, his wife had bought him a pair of red socks as a joke. One day in a meeting he absent mindedly put his feet up on the table, getting some laughs and loosening up the meeting. So Shea started wearing red socks, not all the time, but to important meetings. Eventually the socks were accepted as a good-luck charm to wear to presentations. Even senior management at General Motors, where putting one's feet on a desk was discouraged and wearing red socks was unthinkable, got used to the idea. . . .

Armed with his red socks and his puns and an emerging sense of how good he was getting to be at this sort of engineering, Shea set out to rescue the lagging

Titan contract. He moved into the plant, and for five days a week, all three shifts, he was there, catching catnaps on a cot set up in his office. It was a pattern he would repeat later, during Apollo. The reasons were partly motivational—people work harder when they see the boss working all three shifts. "But it also lets you find out everything that's going on," Shea said. "Things I'd find out at night, I'd get corrected during the daytime." Shea began handing out red socks as an award for good performance. His enthusiasm and energy were infectious.

Shea pulled it off, making up the six months. [Murray and Cox, 1989, pp. 121–123]

1.3 APPROACHES FOR IMPLEMENTING SYSTEMS ENGINEERING

We have just provided a description of what happens inside the process associated with the design of an engineered system and are about to describe several approaches for organizing that process. But let us step back a minute to look at the bigger picture, as summarized in Figure 1.6. The system that we have been tasked to design exists in a

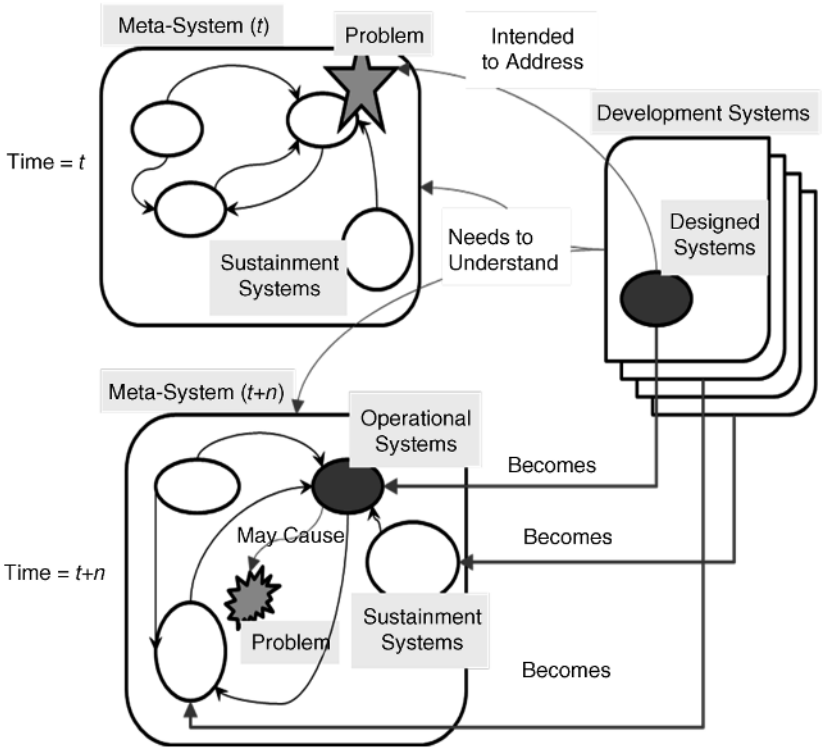

FIGURE 1.6 Characterizing the broader systems' design problem. (After Martin [2004].)

broader system, called the meta-system. This meta-system contains other systems and is purposefully pursuing some objectives. There is likely a sustainment system that is part of this meta-system that is providing supplies and support to one or more of the systems that comprise the meta-system.

Some group has identified a problem with the achievement of the objectives being attained by the meta-system and has tasked a development system (organization) to design a system that will replace or upgrade one or more of the systems in the meta-system. In order to understand how to design this new or upgraded system, the people in the development system must understand the meta-system or else they will have little chance of success. Understanding the meta-system includes the interaction between the system to be replaced and other systems in the meta-system as well as the context or environment in which that meta-system operates. We will refer many times in this book to the creation of meta-system (or mission) requirements and an operational concept as approaches to achieving this understanding of the meta-system.

At some later time, after the meta-system has gone through many changes for which the development system must be tracking and making adjustments, the designed system will be deployed and become an operational system within the changed meta-system first studied. Not only will the context of the meta-system have changed, but many of the systems inside the meta-system will also have changed. In fact, there may well be other development systems working on some of these other systems in the meta-system, including the sustainment system. The introduction of the operational system may in fact introduce new problems into the meta-system. Such potential problems should be imagined as part of the development process and avoided or minimized via the design.

A final caution to the reader is that the development system (an organization of systems engineers and other engineers and experts) must design itself to have any chance of success. This design of the development system must emphasize adaptability to the inevitable change going on in the meta-system as described in Figure 1.6 as well as in another meta-system in which the development system exists.

The traditional, top-down systems engineering (TTDSE) process has evolved from the 1950s. Software engineers have evolved several approaches, starting with a waterfall process, moving to spiral development, and currently focused on object-oriented design (OO). Object-oriented software design gained popularity in the early 1990s shortly after object-oriented programming languages became available.

1.3.1 TTDSE

TTDSE (described in the overview in Section 1.2 and shown in Figure 1.7) is a process for systems engineering that begins a thorough analysis of what the problem is that needs to be solved; this is usually done with an analysis of the current meta-system (the system of interest and its peers (external systems)) performing one or more missions for the primary stakeholders. The result of this analysis is a statement of the *problem to be solved*. Based on this statement of the problem to be solved, several potential, competing concepts for implementing the system of interest are

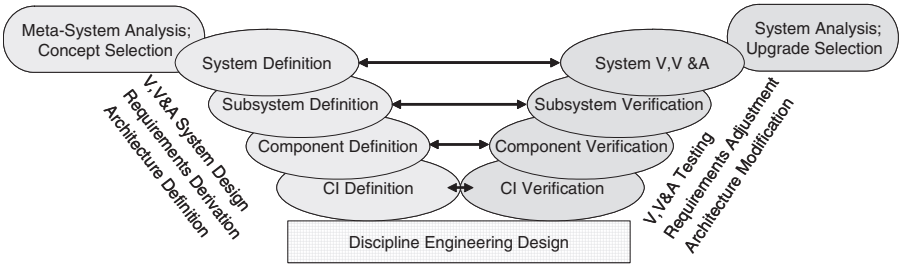

FIGURE 1.7 Traditional top-down systems engineering.

defined; this set of concepts should initially include a very broad range of ideas, some of which are relatively inexpensive while others are very expensive. Next there will be an analysis of the competing concepts, resulting in the selection of the most favorable concept for implementation. Note that this analysis could really be many analyses. (Note that this book does not address the problem definition and evaluation of concepts. This material is covered by most texts on problem solving for defining the problem to be solved. See Checkland [1993], Klir [1985], and Warfield [1990]. Decision analysis (Chapter 14) addresses the evaluation of concepts.)

On the basis of this selection, an operational concept and system-level requirements are defined for that solution concept. These two products (operational concept and system-level requirements) are a statement of the *problem being solved*. Next a layered (or onion peeling) iterative process begins for creating an architecture, deriving requirements, and refining the needed test system and associated data collection requirements. This layered process can have as many layers as are needed; the bottom layer addresses the configuration items that the discipline engineers will design. Each layer repeats the same process (defined in detail in Chapters 6–10). Systems engineers commonly perform a great deal of analysis and modeling at each layer of this process; trade studies are often conducted to examine alternate ways to proceed or solutions that optimize some objective (e.g., cost, reliability, and weight) while minimizing the impact on all other objectives.

Once the CIs have been designed and delivered for integration, the verification, validation, and acceptance testing process begins. Each layer of the decomposition process is verified against the associated derived requirements. During the process, requirements may be adjusted or the architecture and design of the system may be modified as needed. At the system level, validation against the concept of operations and acceptance testing (as defined by the stakeholders) is conducted. Chapter 11 defines this process. If a positive result is obtained, the system is deployed and systems engineering continues by analyzing the usage of the system for needed modification and selecting upgrades that will be implemented in the future. The actual upgrading of the system should follow the same process as defined by the Vee-like structure in Figure 1.7.

TTDSE is primarily a process for designing the many pieces of a system in such a way that many different organizations can be tasked to design one or several pieces

and all of the pieces can be integrated easily and effectively to achieve the desired system. Other references for TTDSE are Blanchard and Fabrycky [1998], Hatley and Pirbhai [1988], Sage [1992], and Wymore [1993].

1.3.2 The Waterfall Model of Software Engineering

One of the earliest concepts of the software engineering process was called the "waterfall" model by Boehm [1976], but introduced by Royce [1970]. The waterfall model (Fig. 1.8) is characterized by the sequential evolution of typical life cycle phases, allowing iteration only between adjacent phases. The waterfall model is known and discussed throughout the software and systems engineering communities and was the basis for Military Standard 2167A for software development. The major problem with the waterfall process is that iteration between phases that are widely separated is all too common.

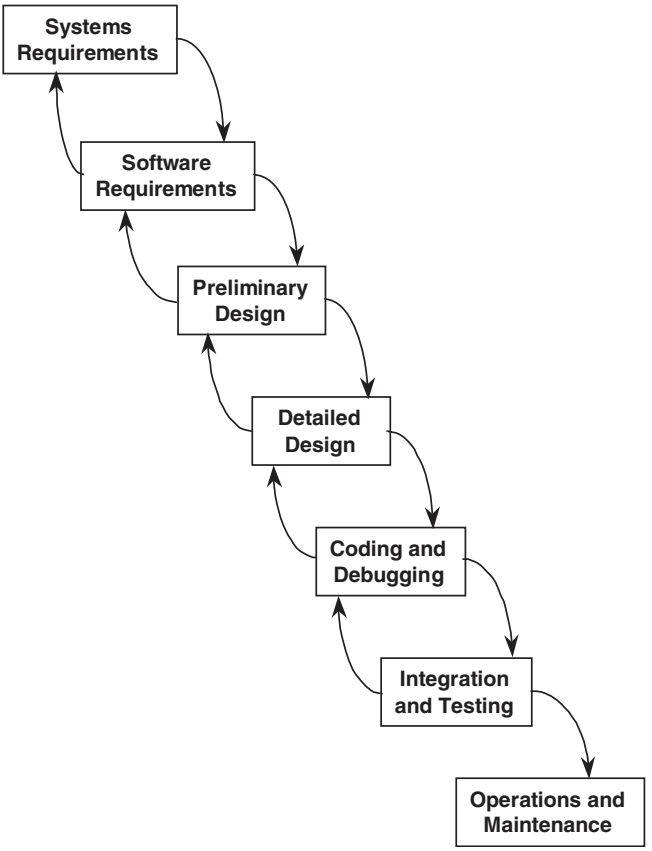

FIGURE 1.8 Waterfall model.

1.3.3 The Spiral Model of Software Engineering

The spiral model (Fig. 1.9), developed in the 1980s [Boehm and Papaccio, 1988] and then modified several times [Boehm, 1986, 1988], addressed the need to shorten the time period between the users' statement of requirements and the production of a useful product with which the users could interact. Too many systems and software implementations were being produced and rejected because the development life cycle took too long; valid requirements at the beginning of the cycle were no longer valid at the time of delivery. In addition, new systems were degraded because the vestiges of learning about the system domain tainted the early designs.

The spiral model has four major processes, starting in the top left of Figure 1.9 and moving clockwise: design, evaluation and risk analysis, development and testing, and planning with stakeholder interaction and approval. These four processes are repeated as often as needed. The radial distance to any point on the spiral is directly proportional to the development cost at that point. The spiral model views requirements as objects that need to be discovered, thus putting requirements development in the last of the four phases as part of planning. The early emphasis is on the identification of objectives, constraints, and alternate designs. These objectives and constraints become the basis for the requirements in the fourth step. There is

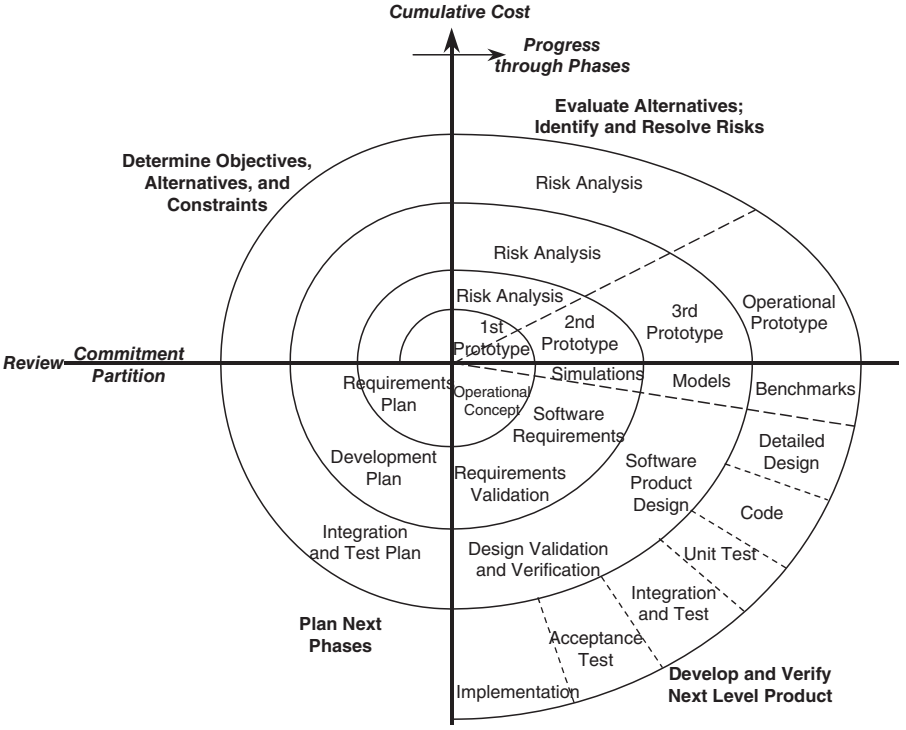

FIGURE 1.9 Spiral model.

also a major emphasis on evaluation and risk analysis as part of the management activities. This management activity is to identify which requirements are most important to discover early in order to minimize problems associated with cost, schedule, and performance. The development effort is composed of prototyping activities, which provide mock-ups of the software or system that will enable the stakeholders to define their requirements. This third step ends with evaluation and testing. The fourth step involves documenting requirements gleaned from the intense prototyping interaction with the users during the current trip around the spiral and planning the next trip around the spiral. The number of iterations around the spiral is variable and defined by the software or systems engineers. The final cycle integrates the stakeholders' needs into a tested and operational product.

Shortly after the spiral model was introduced, various authors [e.g., Boar, 1984] spoke of rapid prototyping as a development process. The rapid prototyping process is meant to produce early, partially operational prototypes. The use of these operational prototypes by stakeholders generates new and improved requirements, as well as provides the stakeholders with increased functionality via early releases of the system. Thus, one could view rapid prototyping through the spiral process model in which the prototypes were partially operational.

1.3.4 Object-Oriented Design

OO design followed from object-oriented programming in the 1970s. OO design is a bottom-up process that begins by defining a set of objects that need to be part of the system in order to achieve the system-level functionality desired. Objects are thought to be basic building blocks that can perform functions (methods) and contain information. Key properties of OO design are inheritance and information hiding. Inheritance means that a general object can be specialized by adding special characteristics; the specialized object will "inherit" all of the properties (methods and data) not overridden by the specialization. Information hiding means that an object does not need to know how another project is producing the information being sent to it, just what that information is. Systems engineers had referred to this idea as modularity for years. Besides being the basic building blocks of a system, objects are seen to promote reusability, testability, and maintainability. For more information, see Ambler [1997].

1.4 MODELING APPROACHES FOR SYSTEMS ENGINEERING

Modeling techniques for designing systems were created as early as the early 1950s. These techniques addressed the connection of system components, the decomposition of system functions, and dynamic behavior of the system. The Unified Modeling Language (UML) was created by several of the OO gurus who had developed their own approaches to modeling and decided an integrated approach was needed. The U.S. Department of Defense Architecture Framework (DoDAF) was developed within the Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) community and then extended to all of DoD. The Object

Management Group's Systems Modeling Language (OMG SysML™) was defined using UML 2.0 and moves the TTDSE process toward a goal desired by many of a model-based version of systems engineering.

1.4.1 Modeling Approaches for TTDSE

The first modeling approach of TTDSE was the block diagram. Each block represented a system component. Lines between the blocks represented the exchange of information, energy, or physical entities. Next, *N*-squared (N^2) diagrams were created to capture a high-level view of the flow of information, energy, and physical entities among the components at a given level of abstraction for the system (see Lano [1990a, 1990b]). Next, the N^2 diagram was transformed to a functional perspective, the components being exchanged for major system functions. Function flow block diagrams were then developed to capture the dynamics of the system's behavior. Meanwhile, software designers were creating data flow diagrams to model software systems. Manufacturing designers were creating the Structured Analysis and Design Technique (SADT) that was later transformed into Integrated Computer-Aided Manufacturing (ICAM) Definition or IDEF0. Data flow diagrams, N^2 charts, and IDEF0 diagrams all capture the same basic time-lapsed flow of information, energy, and physical items among functions. State transition diagrams (or state machines) were developed and enhanced by several engineering disciplines to capture dynamic behavior; these techniques have been applied for some TTDSE efforts. Finally, Petri nets have been developed to model the dynamics of systems. Many TTDSE practitioners use some subset of these modeling techniques. All of these techniques are covered in later chapters of this book.

1.4.2 UML

The Unified Modeling Language is a specification language for modeling objects that is approved by the Object Manage Group. UML 2 was adopted in 2004 and is often described as a graphical modeling language. Critical ideas underlying object-oriented modeling are multiple views at varying levels of abstraction, object, class, inheritance, and extensibility. All useful approaches to systems and software engineering use modeling approaches that enable modeling a system at multiple levels of abstraction. An object is a basic building block of OO programming that can receive messages, process data, and then send messages to other objects. An object can be viewed as a component or actor that has the resources to receive, process, and send data. A class in object-oriented terminology is a grouping of related variables or functions; this is a key to addressing a system at multiple levels of abstraction. Inheritance (now often called generalization) is the process of creating instances of a class based upon specializations of class parameters; this is often the key to software reuse. Extensibility is a way of extending the UML modeling language. For example, stereotypes permit extending elements of UML to a specific problem domain.

UML 2.0 contains 13 different diagram categories that can be aggregated into 3 diagram types (see Table 1.4). Structure diagrams address those issues or elements

TABLE 1.4 Diagram Types for UML 2.0

Structure Diagrams	Behavior Diagrams	Interaction Diagrams
Class	Activity	Collaboration – communication
Component	State machine	Interaction overview
Composite structure	Use case	Sequence diagram
Deployment		Timing
Object		
Package		

that are part of the system being modeled. Concepts for structure diagrams include actor, attribute, class, component, interface, object, and package. Behavior diagrams examine the activities that must happen in the system being modeled. Behavior diagram concepts include activity, event, message, method, operation, state, and use case. Interaction diagrams (considered by some to be a subset of behavior diagrams) address the flow of data and control among the elements in the system being modeled. Concepts for interaction diagrams include aggregation, association, composition, depends, and generalization (or inheritance).

Some important ideas in UML are the use case diagram, which is a high-level view of the use cases; the class diagram, which describes the relationship between structural elements of the system and the external domain; and a set of object diagrams, which are more definitive than the class diagram about the structural elements of the system and their relationships over time. Sequence diagrams are a representation of scenarios or use cases, something that traces back decades. The key design elements are the software objects.

The use case diagram and sequence diagrams define the requirements in a qualitative way; there are seldom any quantitative performance requirements and there are no nonfunctional requirements. Similarly, there is no top-level functional analysis; each object contains operations that can be performed and data that can be used for those operations. UML is primarily a graphical modeling language for creating abstractions or generalizations so that the resulting software system will be more flexible and adaptable.

This UML process is more of a bottom-up design process in which the components of the software are derived from more specific software objects that are designed to be adapted from existing code or coded from scratch. Useful references on UML are Ambler [2004] and Eriksson and Penker [1998]. Software engineers believe the appropriate model of their design is the code itself, so very little modeling and analysis is performed during this process.

1.4.3 DoDAF

The DoDAF provides three integrated views needed for a system architecture; each of the three views is composed of subviews using graphical, tabular, and textual descriptions. A data model is defined that defines entities and relationships among

the data elements that are part of these integrated views. This effort began in 1995, produced versions 1 and 2 of the C4ISR Architectural Framework in 1996 and 1997, respectively, and yielded versions 1 and 2 of the DoDAF in 2003 and 2009, respectively. The Ministry of Defence (MOD) of the United Kingdom and the North Atlantic Treaty Organization (NATO) have adopted similar architecture frameworks: MODAF and NAF, respectively.

In DoDAF 1, there were three top-level views: operational, systems, and technical. The operational view addresses the organizational and human context in which the system will be utilized. The systems view switches to the physical and functional world, starting outside the system and moving inside the system. The technical view addresses standards and conventions. In DoDAF 2, seven viewpoints were adopted: capability, data and information, operation, project, services, standards, and systems. These viewpoints are oriented to supporting DoD decision makers associated with the systems engineering activities. The *capability viewpoint* serves the needs of capability portfolio managers. Business activities are supported by the *data and information viewpoint*. The *operational viewpoint* is used to describe the tasks and activities, operational elements, and operational resource flows. The *project viewpoint* enables the description of contributions by programs, projects, portfolios, or initiatives. The *services viewpoint* is used to describe the satisfaction of DoD functions via services, as opposed to systems. The set of rules governing the management, interaction, and interdependence of parts or elements is defined by the *standards viewpoint*. The *systems viewpoint* describes the systems and other interconnections providing DoD functions.

Each of the viewpoints has a number of products, which capture a subset of the concepts, associations, and attributes relevant to the view. It is important to conceive of the DoDAF as containing a central database of all of the entities and relationships. Each product of each view is then a representation of a subset of that central database. The developers of the DoDAF continue to strive to make this structure useful to decision makers and systems engineers. References include Levis and Wagenhals [2000] and Dam [2006].

1.4.4 SysML

There has been a push among some systems engineers for an approach to systems engineering that is less text based and, therefore, more model based. The arguments against text-based processing are its inefficiencies for finding errors and stress points, testing both performance and timing behavior in one or more competing designs, and providing actionable information for trade studies and design reviews. Ultimately, there is a need to examine performance issues and conduct tests before the first prototype is completed. Software engineers, for the most part, seem to have no problem with waiting until the code is written to find out that there are major timing and latency problems. Hardware has traditionally taken much longer time to redesign, so systems engineers prefer to get the bad news early. This emphasis has led to model-based systems engineering efforts, the most visible of which is SysML.

SysML is a visual modeling language that was adapted from UML 2.0 and enhances the traditional top-down systems engineering process. SysML extends the

TABLE 1.5 Diagram Types for SysML

Structure Diagrams	Behavior Diagrams	Interaction Diagrams	Requirement (New)
Class – renamed to be	Activity (modified)	~~Collaboration – communication~~	Requirement (new)
Block definition	State machine	~~Interaction overview~~	
Internal block	Use case	Sequence diagram	
~~Component~~		~~Timing~~	
~~Composite structure~~			
~~Deployment~~			
~~Object~~			
Package			
Parametric design (new)			

modeling language of traditional, top-down systems engineering; this extension should make the traditional approach to systems engineering less prone to errors and more efficiently implemented. Table 1.5 shows which UML 2.0 diagrams have been dropped (strikethrough), adopted (new), or modified (modified) for SysML. The first thing to notice in Table 1.5 is that there is a new column for requirements with a single diagram type. The column with the most changes is the first column for structure diagrams. Here the class diagram has been renamed to capture two different concepts associated with the physical architecture: block definition and internal block connectivity of parts. A new diagram was created for modeling performance, called the parametric diagram. The package diagram was kept as is from UML 2.0. Within the category of behavior diagrams, the activity diagram has been modified, while the state machine and use case diagrams have been kept as is. Finally, most of the interaction diagrams have been dropped; the only remaining interaction diagram is the sequence diagram. The implication of these changes is that SysML places much greater emphasis on behavior compared to interaction than UML does.

These diagram concepts will be introduced in later chapters of this book.

The real challenge for SysML (and every other model-based approach) is to include easily understood descriptions of the system design and the associated requirements for nonengineering stakeholders. References for SysML are Bock [2006], Friedenthal and Moore [2014], and Delligatti [2013].

1.5 INTRODUCING THE CONCEPT OF ARCHITECTURES

Levis [1993] has defined an analytical systems engineering process (for the left side of the Vee process) that begins with the system's operational concept and includes the development of three separate architectures (functional, physical, and allocated) as part of this decomposition. The functional (or logical) architecture defines what the system must do, that is, the system's functions and the data that flows between them. The physical architecture represents the partitioning of physical resources available to

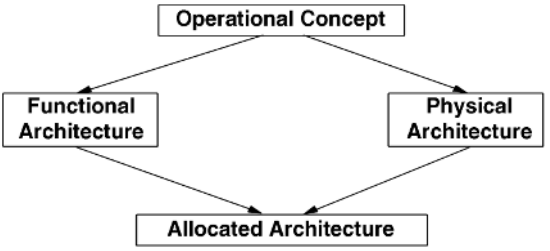

FIGURE 1.10 Architecture development in the engineering of a system. (After Levis [1993].)

perform the system's functions. The allocated architecture (see Fig. 1.10) is the mapping of functions to resources in a manner that is suitable for discrete-event simulation of the system's functions and is analogous to Alford's [1985] approach with behavior diagrams. Figure 1.10 suggests that the functional and physical architectures are developed independent of each other and then combined to form the allocated architecture. This suggestion is inaccurate, rather the two architectures are developed in parallel, but with close interaction to ensure that the allocated architecture is meaningful when the functional and physical architectures are combined. Chapters 7–9 address these three architectures and their development in detail and discuss the interactive development of them.

Critical to this multiple-architecture approach is the balancing of information among them. To be complete, three separate models must be developed: data, process, and behavior models. The functional architecture includes the first two (data and process) models and the initial behavioral model, as discussed in Chapter 7. The behavioral model should be finished and exercised as part of the allocated architecture (see Chapter 9). Each of these three models must be integrated to define the three architectures properly.

Figure 1.11 shows an organization chart representation of a physical architecture of the F-22 fighter. Note that this physical architecture includes more than the F-22; the

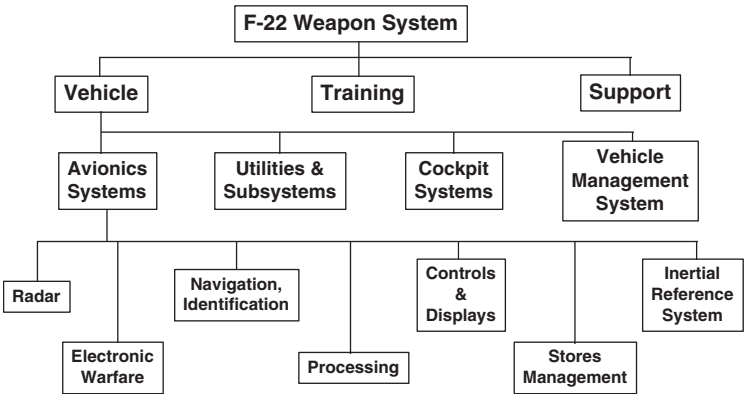

FIGURE 1.11 Sample physical architecture (F-22 Type A Spec). (From Reed [1993].)

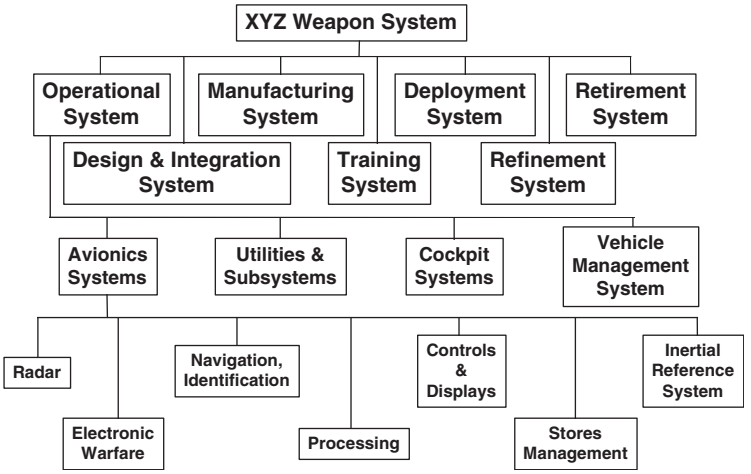

FIGURE 1.12 Life cycle physical architecture.

training and support systems are included as well. For a life-cycle-balanced (concurrent engineering) definition of the F-22, the physical architecture should have been decomposed, as shown in Figure 1.12.

Graphical techniques, such as Figures 1.11 and 1.12, are invaluable because they serve as an excellent communication medium; communication is one of the most important functions of systems engineers. A physical architecture subdivides the problem into manageable parts, permitting and encouraging an iterative process and providing excellent documentation.

Figure 1.13 depicts the systems engineering design process in terms of requirements and architectures in a similar manner as the waterfall process, a sequential decomposition of requirements and the allocated architecture (functions mapped to

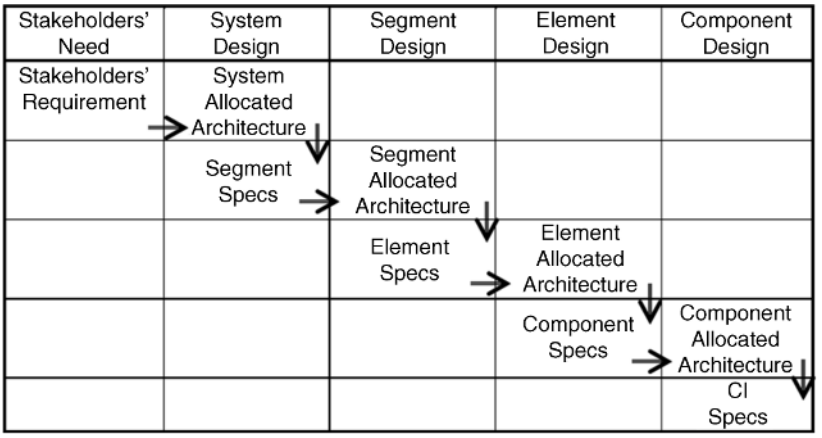

FIGURE 1.13 Design decomposition of architectures and specs.

physical resources) by moving from left to right and top to bottom. A question often asked by new students is: What is the difference between a requirement and a specification. A *requirement* is one of many statements that constrain or guide the design of the system in such a way that the system will be useful to one or more of its stakeholders. A *specification* is a collection of requirements that completely define the constraints and performance requirements for a specific physical entity that is part of the system. The systems engineering design process involves defining all of the system's requirements and then bundling them by segmenting and refining into a specification for each of the system's segments, elements, components, and CIs.

1.6 REQUIREMENTS

Requirements for a system address the needs and objectives of the stakeholders. Just as there is a hierarchy associated with the physical components of the system, there is a hierarchy of requirements. At the top of the hierarchy are mission requirements, which relate to needs associated with missions or activities that are important to one or more groups of stakeholders. These *mission requirements* typically involve the interaction of several systems, one or more of which include individuals or groups of people and are therefore stated in the context of the operation of the system in question with these other systems, called the meta-system or supersystem or system of systems. Mission requirements represent stakeholder preferences for the increased ability to perform their activities with the introduction of the system in question at a lower cost and in a faster time than the existing capability.

Stakeholders' requirements are statements by the stakeholders about the system's capabilities that define the constraints and performance parameters within which the system is to be designed. Systems engineers take these high-level, stakeholders' requirements and derive a consistent set of more detailed engineering statements of requirements as the design progresses. For the purposes of this introduction, requirements are divided into constraints and performance indices. Some constraints are simple, for example, the system must be painted with a specific shade of green. Other constraints are the minimally acceptable level associated with a performance requirement. A performance requirement defines a desired direction of performance associated with an objective of the stakeholders for the system. For an elevator system (which is used throughout this book), a performance requirement might be to minimize passengers' waiting time. For any performance requirement, there must also be a minimum acceptable performance constraint or threshold and a design goal associated with the index; this threshold dictates that no matter how wonderful a design's performance is on other objectives, performance below this threshold on this requirement makes the design unacceptable. This is a very strong statement of needs, and so minimal acceptable thresholds must be established very carefully.

Every major organization, governmental or commercial, has established its own guidelines for system or product development. The names and organizations of the several requirements documents vary somewhat but cover similar material. Table 1.6

TABLE 1.6 Typical Requirements Documents

Document Titles	Document Contents
Problem Situation or Mission Element Need Statement, and Systems Engineering Management Plan (SEMP)	• Definition of stakeholders and their relationships • Stakeholders' description of the problem and its context • Description of the current system • Definition of mission requirements • Definition of the systems engineering management structure and support tools for developing the system
Stakeholders' Need or Stakeholders' Requirements Document (StkhldrsRD)	• Definition of the problem needing solution by the system (including the context and external systems with which the system must interact) • Definition of the operational concept on which the system will be based • Creation of the structure for defining requirements • Description of the requirements in the stakeholders' language in great breadth but little depth • Trace of every requirement to a recorded statement or opinion of the stakeholders • Description of trade-offs between performance requirements, including cost and operational effectiveness
System Requirements Document (SysRD)	• Restatement of the operational concept on which the system will be based • Definition of the external systems in engineering terms • Restatement of the operational requirements in engineering language • Trace of every requirement to the previous document • Justification of engineering version of the requirements in terms of analyses, expert opinions, stakeholder meetings • Description of test plan for each requirement
System Requirements Validation Document	• Documents analyses to show that the requirements in the SysRD are consistent, complete, and correct, to the degree possible • Demonstrates that there is at least one feasible solution to the design problem as defined in the SysRD

summarizes the common major requirements documents that are produced during the beginning of the design phase. The Problem Statement (or Mission Element Need Statement in the military) gets the process rolling and identifies a problem for which a solution in the form of a system (new or improved) is needed. This document supports and documents a decision-making process to start a system development effort. The Systems Engineering Management Plan (SEMP) then defines the systems engineering development system.

Stakeholders' requirements are found in the Stakeholders' Requirements Document (StkhldrsRD). This document is produced with or by the stakeholders and is written in their language(s). Systems engineers need to be involved in a substantial way in this activity, although not all systems engineers share this view. Experience has shown that if this document is left to the stakeholders, the document will be very incomplete. The systems engineers can play a major facilitation role among the various groups of stakeholders as well as bring an assortment of tools to bear on a difficult problem, the creation of this document. These tools (a major focus of this book) ensure a greater completeness and consistency. The methods and tools presented here are equally applicable *in* the rest of the systems engineering process.

The systems engineer then begins restating and "deriving" requirements in engineering terms, called *system* requirements, so that the systems engineering design problem can be solved. This derivation of the StkhldrsRD becomes the Systems Requirements Document (SysRD).

It is critical that the requirements in all of these documents address "what" and "how well" the system must perform certain tasks. Requirements do not provide solutions but rather define the problem to be solved.

The Systems Requirements Validation Document defines requirements associated with the verification, validation, and acceptance of the system during integration. These requirements are high-level requirements that state the needs of the stakeholders for qualifying the design of the system. These requirements form the basis of the problem definition for creating the qualification system that will be used during integration. In addition to defining the high-level qualification requirements, this document should demonstrate that if the systems engineering process continues, an acceptable solution is possible. Unfortunately, this "existence proof" of a feasible solution is seldom produced in practice, leading to a major downfall of many systems engineering efforts. Namely, the realization that many months (or years) later not all of the requirements can be satisfied, and the stakeholders must relax the requirements that the engineers promised could be met.

Systems engineers have always desired to demonstrate the importance of requirements and getting the requirements right, for example, complete, consistent, and correct. In the mid-1970s, three organizations (GTE [Daly, 1977], IBM [Fagan, 1974], and TRW [Boehm, 1976]) conducted independent studies of software projects. These studies addressed the relative cost to fix a problem based upon where in the system cycle the problem was found. Boehm [1981] and Davis [1990, p. 25] compared the results of the three studies (see the first row of Table 1.7). The costs have been normalized so that the relative cost to repair an average problem found in the coding phase is 10 units. These results stood for 20 years. The next eight rows of

TABLE 1.7 Comparison of the Relative Cost to Fix Software in Various Life Cycle Phases

Source	Phase Requirements Issue Found			
	Requirements	Design	*Code*	Test
Boehm (1981)	1	5	*10*	50
Hoffman (2001)	1	3	*5*	37
Cigital (2003)	1	3	*7*	51
Rothman (2000)		5	*33*	75
Rothman_Case B (2000)			*10*	40
Rothman_ Case C			*10*	40
Rothman (2002)	1	20	*45*	250
Pavlina (2003)	1	10	*100*	1000
McGibbon (2003)		5		50
Mean	1	7.3	*25.6*	177
Median	1	5	*10*	50.5

Table 1.7 show results from recent studies, summarized in Haskins et al. [2004]. As can be seen, the results have held up well. Getting the requirements right is a very difficult task, and therefore a task that is fraught with errors. An error that is caught during requirements development can be fixed for about 10% of the cost associated with an error caught during coding. Errors caught during maintenance in the operation of the system cost about 20 times that of an error caught during coding and 200 times the cost of an error caught during requirements development. Unfortunately, many of these errors are not caught until late in the life cycle, causing the expenditure of significant money.

1.7 SYSTEM'S LIFE CYCLE

There are many ways to define a system's life cycle. However, the common phases associated with a system are development, manufacturing, deployment, training, operations and maintenance, refinement, and retirement. Systems engineers have activities in all of these phases, but the primary phases of concern to the systems engineers are development and refinement. Stakeholders use and maintain the system in the operation and maintenance phase. A common mistake is to envision these phases as distinct and separate in time. In fact, it is common (though not required) to have four distinct periods: development only, preinitial operational capability development and testing, operational use and refinement, and retirement. All but the first period have multiple phases occurring in parallel, as shown in Figures 1.14–1.17.

In the development period, the systems engineering team receives resources from the bill payer and begins the development of the system. This period involves heavy interaction with the stakeholders as the requirements process is begun, and the

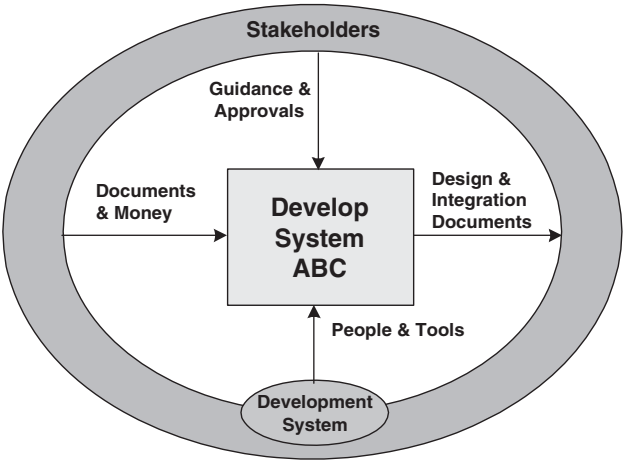

FIGURE 1.14 Development period.

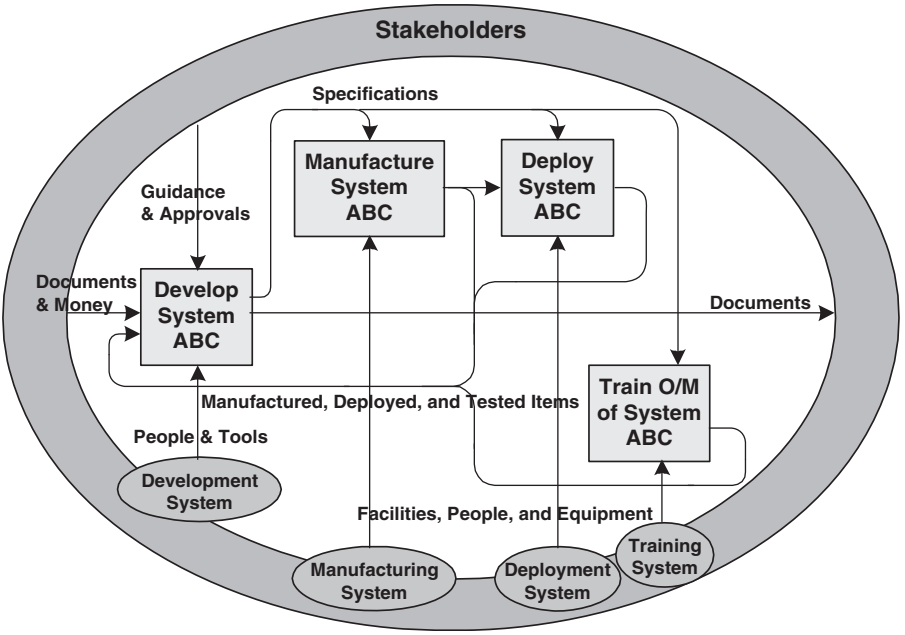

FIGURE 1.15 Period of preinitial operational capability.

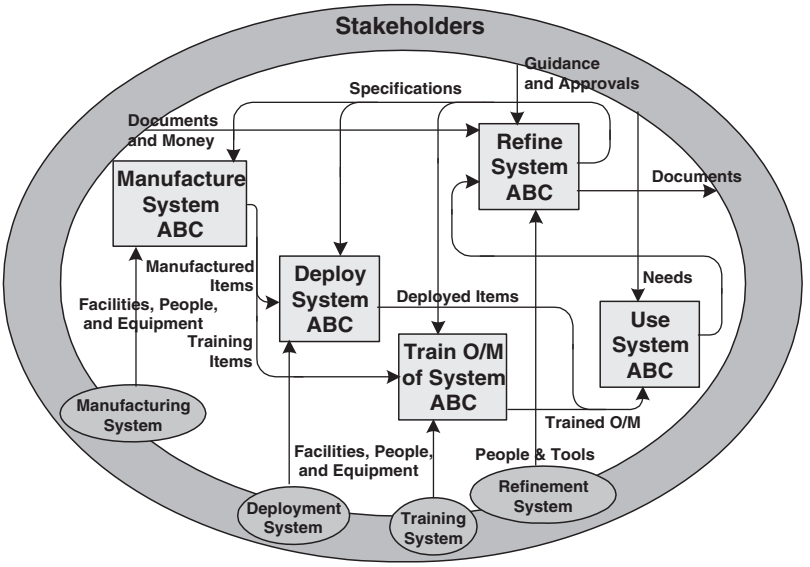

FIGURE 1.16 Period of operational use and refinement.

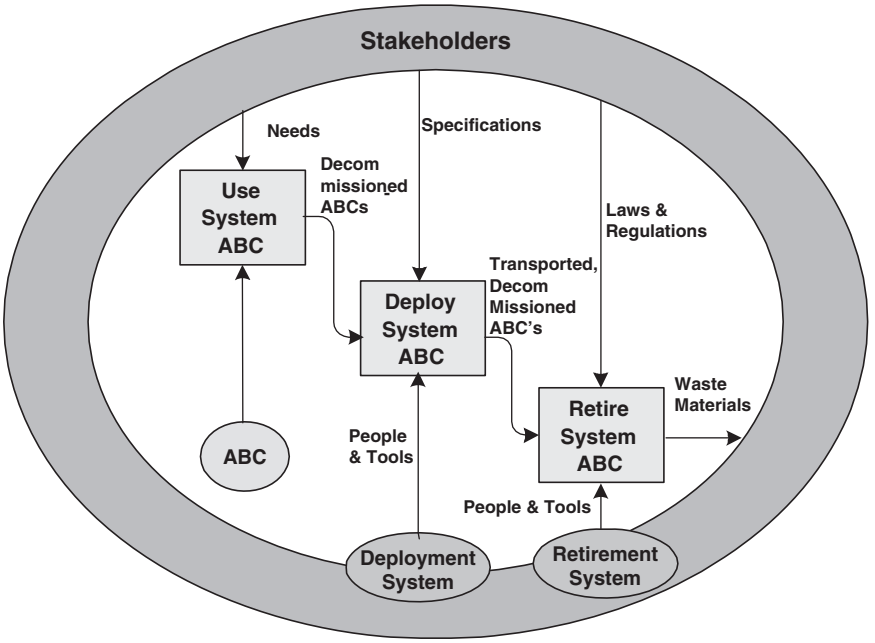

FIGURE 1.17 Retirement period.

architectures and models for simulation and analysis are initiated. However, this period ends when the manufacturing, deployment, and training teams begin preparation for the system.

Development, manufacturing, deployment, and training activities are pursued concurrently during the second period, after concurrent design occurred in the first period. Specifications flow from the development process to the other three. Manufactured, deployed, and training equipment flow to development for testing. Interaction continues with the stakeholders as final testing occurs, leading to the acceptance of the system by the stakeholders. This period ends just as the first operational systems are being delivered to the users.

The third period begins as users receive the first operational items. This period also contains continued production of the system, as well as deployment of and training on the system. Refinement of the design begins here. Manufactured items are sent to the deployment system, which delivers them to users. One of the most difficult problems to solve adequately from the perspective of the users is how to deploy upgraded items while the existing items are being phased out. Training items are sent to the training system (if needed), which produces trained operators and maintainers (O/M). Users and maintainers provide feedback about what they like and do not like, which is used during the refinement phase to make changes to the design, leading to upgrades of the system.

Finally, the bill payer of the system decides when the useful life of the system is over, beginning the initiation of the last period. The retirement phase may take considerable time. As the system is removed from service, the deployment system is used to transport the system from users to retirers. Note that this retirement process can be very orderly, as is the case with military systems. Alternatively, the retirement can be user-driven as is the case with most commercial products such as cars and computers.

Wenzel et al. [1997] describe the cycle model (see Fig. 1.18) that attempts to capture many of the issues discussed in this chapter. The cycle model stresses five cycles that include the elements of design and integration that have already been discussed as well as the management aspects of systems engineering. Table 1.8 describes these cycles in some detail. The first cycle satisfies the key elements of stakeholder satisfaction, beginning with the determination of the need and ending with the delivery of the system to satisfy those needs. The development functions on this first cycle include requirements development and creation of the system design. The second cycle (verification) addresses the modeling, prototyping, and testing, which must be part of the development process; these cycles within the verification cycle enable the requirements and the solution to be refined and verified. The third cycle enables management to insert technologies and external resources into both the development and the manufacturing processes to improve the chances of stakeholder satisfaction, subject to the constraints faced by management. The controlling cycle provides configuration management throughout development and enables product releases and updates throughout the system's life cycle. Finally, top-level management and stakeholder review and approval are included in the final cycle.

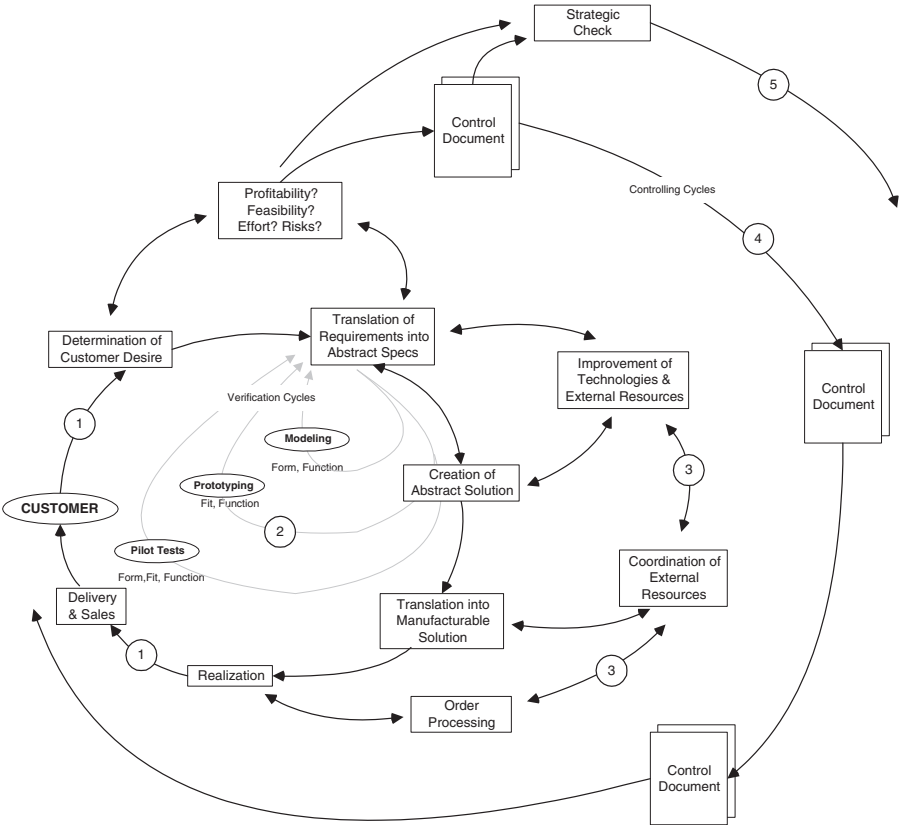

FIGURE 1.18 Cycle model of systems engineering. (After Wenzel et al. [1997].)

TABLE 1.8 The Cycles of the Cycle Model

Design and Integration Cycles	Management Cycles
1. *Core Cycle:* realization of stakeholder needs, followed by requirements development, design, manufacturing and product delivery	3. *Technologies and External Resources Cycle:* insertion of the appropriate technologies and resources into the systems engineering process
2. *Verification Cycle:* analysis, simulation, prototyping, integration, and testing	4. *Controlling Cycle:* configuration management of the design process and multiple product releases and updates
	5. *Strategic Check Cycle:* management assessment and approval of product development

1.8 DESIGN AND INTEGRATION PROCESS

Recall the design and integration Vee as identified by Forsberg and Mooz [1992]. The Vee model defines five major functions for the design or decomposition phase, as shown in Figure 1.19. Note that these functions must be repeated for each stage of the decomposition process. A modification of the more detailed design functions, as put forth by Forsberg and Mooz [1992], is shown in Figure 1.20. This figure also shows how the Forsberg and Mooz [1992] functions are grouped to be comparable to the five analytical systems engineering functions.

The five detailed functions that comprise the design phase must address up to five different dimensions of data (see van den Hamer and Lepoeter [1996]): (1) system variants when the system is a member of a product family (e.g., personal computers and automobiles), (2) system versions when the system is a product that evolves over time (e.g., operating systems), (3) views of the system (e.g., data and process), (4) hierarchical details or onion peels (e.g., system and subsystem), and (5) status of the data (e.g., stable and approved versus tentative or draft).

For many systems, five modeling views [Karangelen and Hoang, 1994] are critical for capturing the totality of a system: environment, data or information, process, behavior, and implementation. The *environmental* view captures the system boundary, the operational concept, and the objectives of the system's performance. The *data* or *information* view addresses the relationships among the data elements that cross the system's boundary and those that are internal to the system; this view can be critical for information and software systems, but

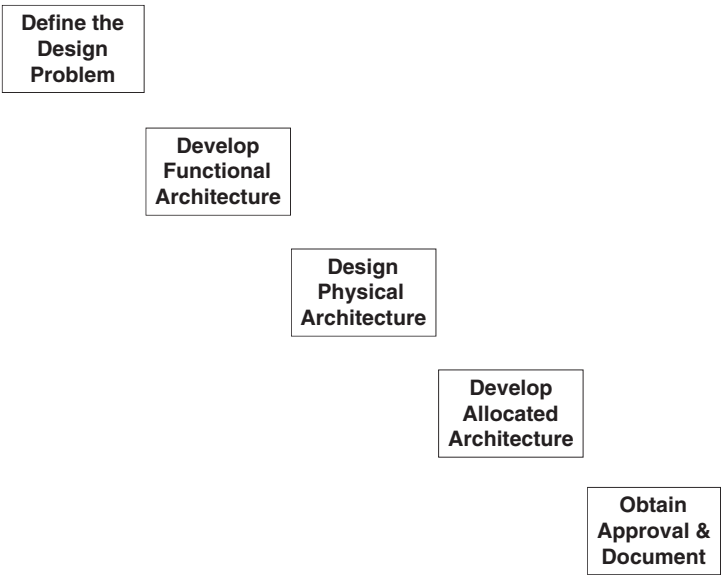

FIGURE 1.19 Five major functions of the engineering design of a system.

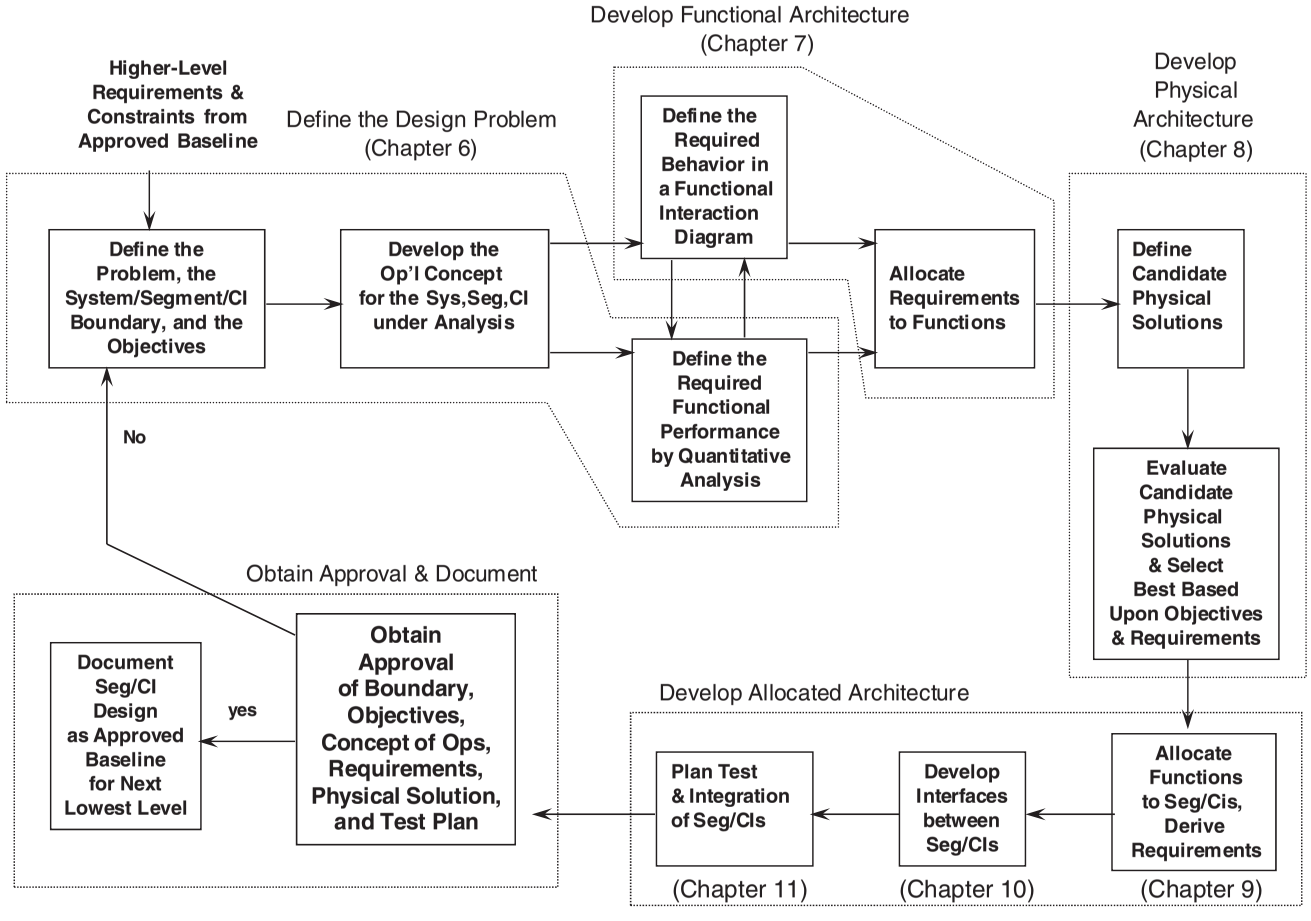

FIGURE 1.20 Detailed functions of systems engineering design.

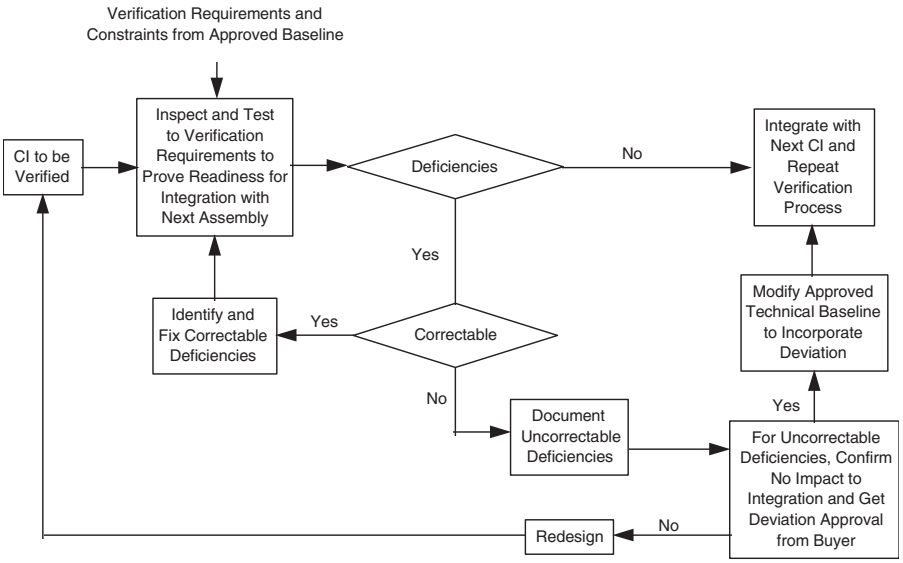

FIGURE 1.21 Functions of the systems engineering integration process.

incidental to mechanical systems. The *process* view examines the functionality of the system and is used to create the functional architecture. The *behavior* view addresses the control structures in which the system's functions are embedded. The *implementation* view examines the marriage of the physical architecture with the process and behavior views; the allocated architecture represents the implementation view. In later chapters, these views and the tools that are used to execute them will be addressed.

Figure 1.21 shows a modification of the Forsberg and Mooz [1992] integration functions. Most of this activity is dedicated to the verification that the integrated components, elements, and segments meet the derived requirements (specifications) of the systems engineering process. The final iteration of the integration functions is devoted to the validation of the system – Is this system the system the stakeholders wanted? Will they accept the system? The answer to this question is substantially determined by the extent to which the systems engineers have kept the stakeholders involved throughout the process. The greater the involvement, the more the stakeholders understand what trade-offs were made and why.

There are four primary methods for testing the system to complete the verification and validation process: instrumented test using calibrated equipment, analysis and simulation using equations and computers, demonstration or functional test using human judgment, and examination of documentation using human judgment. As integration moves from CIs and approaches the system level, human judgment must be relied upon more and more because the cost of instrumented testing on the system as a whole is prohibitive.

1.9 TYPES OF SYSTEMS

There are many possible ways to categorize systems:

natural vs. man-made
closed vs. open
static vs. dynamic
simple vs. complex
reactive vs. nonreactive
precedented vs. unprecedented
safety-critical vs. not safety-critical
high reliability vs. not high reliability
high precision vs. not high precision
human-centric vs. non-human centric
high durability vs. not high durability

However, the process described in this book should work for all "man-made" systems, with some tailoring. Clearly, a great deal of more engineering and systems engineering is required for an unprecedented system (the Shuttle) than for a precedented one (new automobile).

Magee and de Weck [2004] propose a two-dimensional classification structure of systems that was derived from the work of several other authors. The two dimensions include the character (energy, matter, etc.) of the major output of the system and the type of operation or process being employed to produce this major output. The major outputs of Magee and de Weck were broadened to include the following:

- *Matter (M):* physical objects, including organisms that exist unconditionally
- *Energy (E):* stored work that can be used to power a process in the future
- *Information (I):* anything that can be considered an informational object
- *Value (Monetary) (V):* monetary and intrinsic value object used for exchange

Magee and de Weck [2004] also broadened the list of operands or process manipulators to include the following:

- *Transformation Systems:* transform objects into new objects
- *Distribution Systems:* provide transportation, that is, change the location of objects
- *Storage Systems:* act as buffers in the network and hold/house objects over time
- *Market Systems:* allow for the exchange of objects mainly via the Value layer
- *Control Systems:* seek to drive objects from some actual state to a desired state

Table 1.9 provides an example for each of the 20 combinations in the Magee and de Weck structure.

TABLE 1.9 System Classification by Magee and de Weck [2004]

Major Process and Operand	Major Output			
	Matter	Energy	Information	Value
Transform or process	Manufacturing plant	Power plant	Computer chip	Mint
Transport or distribute	Package delivery company	Power grid system	Telecommunication network	Banking network
Store or house	Dam	Dam	Public library	Bank
Exchange or trade	Internet auction company	Energy market	News agency	Stock trading market
Control or regulate	Health care company	Energy agency	International Standards Organization	Monetary regulator

1.10 SUMMARY

Engineering involves the practice of applying scientific theories to the development, production, deployment, training, operation and maintenance, refinement, and retirement of a system or product and its parts. The engineering discipline that addresses the creation of a system that meets the needs of defined stakeholders is systems engineering. The engineering of a system involves both the design of the system's components and configuration items and the integration of those CIs and components into a qualified system acceptable to the stakeholders across the life cycle of the system.

The Vee model of the engineering of a system defines the design and integration processes of TTDSE and form the basis for this book. These processes are iterative. As illustrated in Figure 1.22, design starts as a top-down process and is analogous to peeling an onion to uncover the specifications associated with increasingly detailed components of the system. However, the trade-offs and decisions associated with the design process are so complex and intertwined that there is significant movement between low- and high-level design issues. The key to successful design is the isolation of design decisions using sound engineering principles so that this movement between low- and high-level design issues is consistent with the needs of the development process. There are logical arguments for decreasing development costs by spending the money to conduct a reasonable, systematic engineering effort of the total system.

Multiple types of architectures are introduced to differentiate between what the system does (its functions) and what the system is (its resources) and how the functions are allocated to the resources to enhance the cost-effectiveness of the system in the eyes of the stakeholders. The functional and physical architectures are developed in parallel to enhance their integration into the allocated architecture.

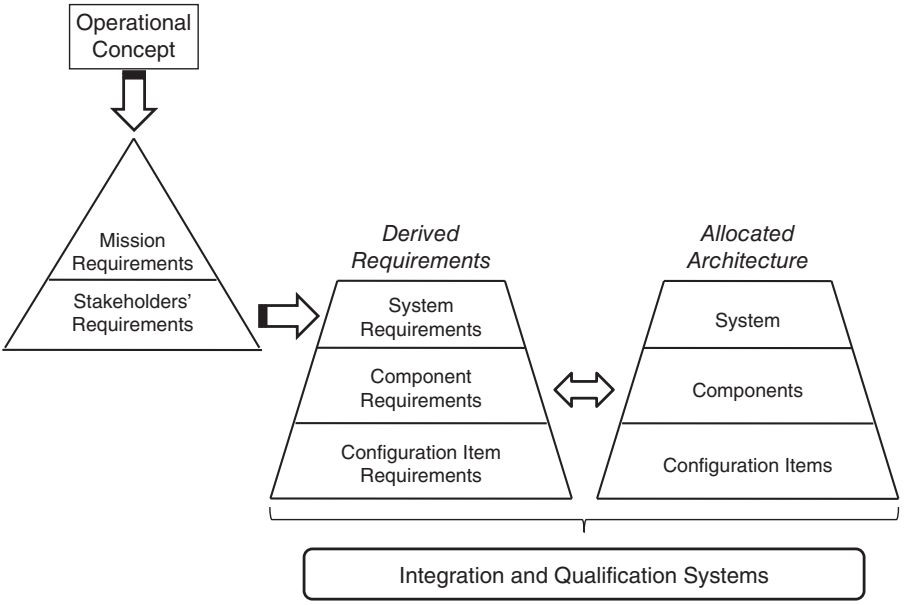

FIGURE 1.22 Summary of TTDSE.

Requirements are used to define the design problem being solved at various levels of detail. Mission requirements define the problem in terms most meaningful to the stakeholders, terms that relate to enabling the stakeholders to accomplish tasks better, faster, and cheaper. Stakeholders' requirements are the next level of details that constrain specific characteristics of the system so as to achieve the mission requirements. Derived requirements relating to the system and specific components are even more detailed constraints upon the system. In addition to the requirements related to the system, qualification system requirements must be developed to address the verification, validation, and acceptance of the system during integration.

The integration process receives less attention than the design process and is often viewed as the yin (weaker, passive side) of development, design being the yang (stronger, active side). However, integration cannot be passive after an active design process. Rather, design and integration must proceed in harmony; integration, if done well, actually improves as well as checks the design process.

There are at least five ways that good systems engineering adds value. First is defining the problem clearly and well and then finding a good solution that balances the needs of varying segments of stakeholders and the multiple engineering disciplines. Second, systems engineers serve as a communication interface among stakeholders and engineers. Finding showstoppers that are present in the design and getting them fixed is the third value adding element. Finding design errors early when these errors are still relatively cheap to fix is the fourth. Fifth, systems engineers help identify high-risk elements of the design and develop risk mitigation strategies.

CASE STUDY: HUBBLE TELESCOPE TESTING DECISIONS

Lyman Spitzer of Princeton University (1946) suggested that a telescope in space would eliminate the atmospheric effects that blurred images seen on Earth. The National Academy of Sciences proposed launching a telescope into space in 1972. NASA began the Hubble Space Telescope in 1977. After many project mishaps, the Hubble was ready to be launched in 1986. However, the explosion of the shuttle *Challenger* delayed the launch by 4 years. In April of 1990, the Hubble was launched; on May 20, the moment of truth arrived. At first the scientists were thrilled with the data that were arriving from space; after further work the scientists however noticed a spherical aberration. The Hubble was providing a resolution of 3 times that available with telescopes on the ground, but the originating requirement for Hubble had been 10 times Earth-based telescopes. In June 1993, the shuttle *Endeavor* carried a repair team to the hobbled Hubble. The astronauts spent 3 days of painstaking efforts to install a corrective "contact lens," replace the original Wide-Field and Planetary Camera, and replace the original solar panels to eliminate jitter twice each orbit as the satellite crossed from daylight to darkness. These repairs cost over $50 million.

When the first images from Hubble were examined, the scientists knew that Hubble needed some adjustment. Several focusing tests were proposed. The telescope was taken completely out of focus and then brought slowly back into focus; this is a common approach to check for errors in any optical device. Meanwhile, another scientist wrote a software program to simulate the images from a telescope with a spherical aberration in its mirrors. The test images were amazingly similar to the simulated images, leading to a devastating conclusion.

The Hubble telescope is a two-mirror reflecting telescope, a special type of Cassegrain telescope called a Ritchey–Chretien telescope. The primary mirror (96 inches) and secondary mirror were to be hyperbolic in shape; the manufacturing process is to grind the mirror as close as possible to this shape and then polish the mirror to remove all possible aberrations within the specified tolerances. During the grinding and polishing processes, tests were conducted with a computer-controlled optical device, a reflective null corrector consisting of two small mirrors and a tiny lens. Unfortunately, the spacing between the lens and the mirrors was off by 1.3 millimeters. The aberration, 0.001 arcseconds from the design specification, resulted in an error 100,000 times the size of the desired 1/50 wavelength of light.

Why was a mistake this large not detected? Photos taken during the manufacturing process in 1981 showed the flaw, but the flaw was not noticed in the photos or other testing. A knife-edge test was conducted on the main mirror. This sophisticated and complex test produced results showing that the null corrector results were incorrect. Either Perkin-Elmer (the prime contractor) thought these results invalid and did not report them to NASA, or NASA managers ignored them on the grounds that the knife-edge test results were not correct. Two other tests could have been conducted but were not. Eastman-Kodak was a competing contractor and had built an identical primary mirror.

The primary mirrors could have been swapped and the null corrector tests rerun. The second test was an end-to-end test conducted on the assembled mirrors and other components. This test was deemed too expensive; NASA claimed the test would have cost more than $100 million, but soon had to back down when independent estimates were 10 times lower, and the Air Force could possibly have conducted tests using existing equipment.

This testing situation was aggravated and explained by management conflicts and mistakes within NASA and by cost overruns. NASA devised a management structure that included two centers, Goddard and Marshall. Marshall was given primary responsibility even though Goddard had more experience in systems of this type. Lockheed Aerospace was awarded the prime contract. Eastman-Kodak and Perkin-Elmer competed for the job of the primary mirror. Eastman-Kodak had more experience, but Perkin-Elmer provided a lower bid. Eastman-Kodak was given a contract to produce a backup primary minor, a risk mitigation strategy that could have been proven very insightful if the flaw in the Perkin-Elmer mirror had been detected [Feinberg, 1990; Petersen and Brandt, 1995; Sinnott, 1990].

PROBLEMS

1.1 Compare and contrast the waterfall, spiral, cycle, and Vee models of the systems engineering process. In particular, what (e.g., functions performed, time sequence of functions, outputs produced, interaction with stakeholders) is the same in each of these processes and what is different? Are there some categories of systems for which one process would be better than the others? Use outside references to gain more information on the waterfall and spiral models.

1.2 Describe your personal experience with a system whose capability disappointed you. In your opinion, was this disappointment a design mistake made by the system's designers or the result of a trade-off decision that had to be made during the system's design. For example, a keyboard that is too small to be as usable as you would like on a laptop computer is the result of a trade-off decision. However, a keyboard with a poor touch for typing is a design mistake. Consider the following examples:

Example	Design Mistake	Trade-Off Decision
Alarm Clock/Radio – The requirements are to show the time, to provide radio reception and listening capabilities, and to serve as an alarm clock. On this particular unit, there are two	This is a design flaw. Requirements development should have established this as a design issue. Testing should have identified the problem	

(*Continued*)

Example	Design Mistake	Trade-Off Decision
buttons on the top to adjust the time. The buttons can be depressed easily, both on purpose and accidentally. Accidental depressions will cause the alarm to activate at the wrong time		
Alarm Clock/Radio – The sleep timer, timed play and record, and clock display are only available via the remote control. If the remote is lost, these features cannot be changed	This may have been a design flaw if not consciously addressed	This may have been a trade-off decision if placing controls on the unit was too costly
Digital Audio System – The user wants a repeat button that causes the repetition of a track from a CD; the current repeat button replays the entire CD. The user also wants a means to fast forward or rewind a few seconds of a track on a CD	A repeat button for a track of a CD is quite common, so this was probably a design mistake	Fast forwarding or rewinding a few seconds could have been a trade-off decision
Stereo System – The components of the system can be turned on separately, but there is only one power off button that controls the entire system	This is a design flaw; if the components can be turned on separately, they should be able to be turned off separately	

1.3 More often than desired, engineers are required to estimate quantities related to some aspect of a system because the necessary data are not available. Systems engineers often have to estimate quantities related to the meta-system. There has been quite a bit of attention to estimation in K-12; a common example is to estimate the number of gas stations in the 48 continental states of the United States.

(a) How would you go about this? What are several ways to estimate this quantity? Besides information about how many people or how many cars there are in the United States, what other information do you know that might be related to the number of gasoline stations?

(b) Search the web and make a list of ways that other people have tackled this problem. Does this list give you any new ideas? What are they?