A similar distinction should be made with the functions of systems engineering and those of the software or hardware engineering disciplines. Each group performs similar functions but on different scales and breadth of application. While the overall systems engineering function is concerned with the technical processes, methods, and tasks of a project at the architectural and operational level, the software or hardware engineering functions are concerned with the technical processes, methods, and tasks needed to develop and deliver their components to the systems engineering function for evaluation and, ultimately, integration. As a result, the systems viewpoint must be carried forward into the software and hardware specialties.

For projects developing products with both hardware and software components, the software and hardware functions should have representatives on the *systems engineering team.* The role of these individuals will be both to support the systems engineering effort with their specific skills and to carry the systems viewpoint forward into the activities of their specialized functions as the components are developed. In the event that the project is developing only software items and has no other organization with suitable technical background, the software organization itself must adopt the systems role. Thus, either scenario requires that individuals or groups be available to perform the software systems engineering function.

1.4 Software Systems Engineering

As discussed above, software systems engineering, like systems engineering itself, is both a technical and management process. The *technical process* of SwSE is the analytical effort necessary to transform an operational need into a description of a software system; a software design of the proper size, configuration, and quality; its successive elaboration in requirements and design specifications; the procedures needed to verify, test, and accept the finished product; and the documentation necessary to use, operate, and maintain the system.

SwSE is not a job description—it is a process and a mind-set that should be adopted when approaching software engineering at the highest levels. Many organizations and people can and do practice SwSE: system engineers, managers, software engineers, and programmers, as well as customers and users. Many practitioners consider SwSE to be a special case of SE. Others consider SwSE to be part of software engineering. But what really matters is that it be done, not what organizational arrangements are made to do it.

SE and SwSE are often overlooked in software development projects. Systems that are composed entirely of software or run on commercial, off-the-shelf computers are often considered "just" software projects, not system projects, and no effort is expended to develop a systems engineering approach. This results in a *code-centric* approach to product development that ignores many important customer requirements, commonly known as *hacking*. The lack of a systems engineering approach to projects often results in software that will not run on the hardware selected, or will not integrate with hardware and other software systems, or that is not suitable for long-term usage. This neglect of the systems aspects of product development has contributed to the aforementioned long-running software

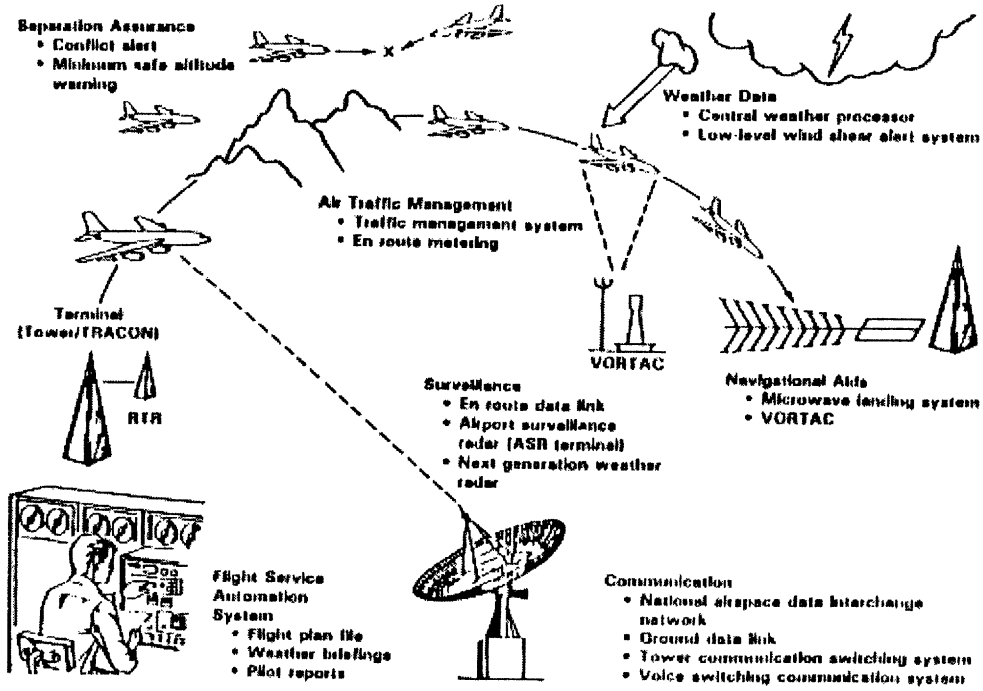

Figure 1.3: Software Integrates the System (Courtesy Northrop Grumman Information Technology, formerly Logicon, Inc.).

crisis [Gibbs 1994, Royce 1991].

Dr. Winston W. Royce, an early leader in software engineering, is credited with introducing the term *software systems engineering*, in the early 1980s [Royce 1988].

1.4.1 Role of Software in Modern Systems

The dominant technology in many modern technical products is software. Software often provides the cohesiveness, control, and functionality that enable products to deliver solutions to customers. Figure 1.3 [Fujii 1989] illustrates the concept that software is the unifying element of modern systems. Software also provides the flexibility needed to work around limitations or problems encountered when integrating other items into the system. This is especially true for problems discovered late in the development cycle, such as in integration. Fixing a hardware/software interface anomaly in software can save, literally, hundreds of thousands or even millions of dollars compared to a solution implemented in hardware. "Software is easy to change, but hard to change correctly" (an anonymous project manager).

As a result, software is usually the most complex part of any system and its development often becomes the greatest technical challenge as the system is created. In the air traffic control system, for example, software integrates people, radar, airplanes, communi-

cations, and other equipment to provide an effective air traffic control system. Software also provides the functionality needed to integrate modern business in the form of *enterprise information systems* or *enterprise resource planning* systems, as well as supporting the creation of virtual communities and interest groups on the Internet.

1.4.2 Application of Systems Engineering to Software Systems

As described earlier, systems engineering defines the product at the highest level. The systems engineering approach focuses on the operation and use of the product throughout its life cycle and applies this viewpoint to the specification of the essential features that all solutions to the problem should possess. This is done prior to decomposing the solution into specific hardware and software subsystems. Applying this same philosophy to software subsystems, or to systems composed purely of software, results in the description of the software in terms of its essential features and how those features relate to the use and operation of the system throughout its life cycle. Later, as the software is decomposed into specific software subsystems, explicit design decisions are applied to the design. These, in turn, become constraints during the next steps of the implementation process.

The concept of deferring design decisions until the proper time is a generalization of the practice of avoiding the application of constraints for as long as possible in the development process. The further into the development process a project gets before a constraint is defined, the more flexible the implemented solution will be. The designer, whether a systems or software designer, must, of course, exercise good judgment in applying this rule, because design decisions cannot be delayed to the point where the implementation is affected. In addition, some constraints are so fundamental that they can virtually dictate some aspects of the solution. The trick is to know the difference: the motto "Write no code before its time" applies equally well to design.

1.4.3 The Necessity of Software Systems Engineering

The rate at which new, complex products are being developed is continuously accelerating, and the proper operation of many of these new systems is highly dependent on software systems and subsystems. *Thus, software systems are larger and more complex now than at any time in history.* This trend is caused by a number of factors, acting separately and together:

- Inexpensive computer hardware is available, which allows users to specify more sophisticated and complex requirements.
- Software is providing an ever-increasing percentage of many systems' functionality.
- Increased software complexity is driven by increased system complexity.
- Customers are demanding more reliable and usable software systems.

- Customers want the flexibility offered by software-based solutions.

As a result, software development costs are growing in both absolute and relative terms; it is not uncommon for the cost of the software of a system to be several times that of the hardware. As the cost of developing the software grows, so does the schedule. If the project is conducted properly, the schedule growth is not as severe as that of the cost,[1] but if the project is not managed properly the schedule can more than double. As software projects grow in size and complexity, so does the risk that problems will occur. Such large and complex systems require the technical system management provided by the systems engineering approach. Without such a systems engineering approach the following problems often result:

- Complex software systems development projects become unmanageable.
- Costs are overrun and schedules are missed.
- Unacceptable risks are not recognized and, as a result, the project takes the risk unknowingly.
- Unacceptable engineering procedures (or none at all) are used.
- Erroneous decisions are made early in the life cycle that are often not detected until late in the project when they are very costly to correct.
- Subsystems and components are developed without proper coordination. The result is that they will not integrate readily—or at all.
- Some parts and components are never specified or built. Requirements, even if captured in the software requirements specification, are ignored and, as a result, not met.
- The delivered system fails to work properly.
- Parts of the system must be reworked after delivery— repeatedly.

A consistently applied software systems engineering approach is necessary if the software community is to build and deliver the "new order" of computer-dependent systems now being sought by governments, industry, and the public.

1.4.4 Role of Software Systems Engineering

For software only projects, the software systems engineering function is responsible for the overall technical management of the system and the verification of the final products. It is responsible for the activities and tasks listed in Table 1.1. This is not meant to be an all-inclusive list but to give insight into the types of tasks and responsibilities comprising SwSE.

[1] For an explanation of this phenomena, see the discussion of parametric models in the chapter on cost and schedule estimation, Chapter 12.

Interface technically with the customer.
Support systems engineering at the product level for hybrid hardware/software systems.
Determine project software effort and schedule.
Determine and manage software technical risk.
Define and document software requirements.
Perform functional analysis of requirements and flowdown.
Determine system data throughput and storage.
Perform trade-off studies and time-line analyses.
Develop prototypes.
Design and document top-level software system architecture.
Define and document software technical interfaces.
Manage software interface control working group.
Develop verification and validation procedures and plans.
Develop software systems test plans, procedures, and cases.
Define standards, practices, and methodologies.
Conduct external and internal reviews.
Verify and audit technical products.
Manage software system configuration and change.
Identify technology needs.
Provide subcontractor software technical direction and oversight.

Table 1.1: Activities and Tasks of Software Systems Engineering

In projects developing both hardware and software items, the software systems engineering effort will either perform these functions entirely or will play a significant role in performing them as part of an overarching systems engineering activity.

1.4.5 Relationship to Software Engineering

Some areas of overlap are seen with the preceding discussion of software systems engineering and the activities of software engineering as usually conceived. Significant differences do, however, exist. In particular, as usually defined, software engineering is principally concerned with the implementation of software requirements. More fully, software engineering is:

- The practical application of computer science, management, and other sciences to the analysis, design, construction, and maintenance of software and its associated documentation;

- An engineering science that applies the concepts of analysis, design, coding, testing, documentation, and management to the successful completion of large, custom-built computer programs under constraints of time and budget; and

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

These definitions would imply that software systems engineering is partly a subset of software engineering. However, the above definitions do not focus on the needs of users, nor do they explicitly encompass the full life cycle of support that is the dominant feature in the definitions of systems engineering presented earlier.

Figure 1.4 illustrates the relationships between systems engineering, software systems engineering, and software engineering functions. In this view, the systems engineering function performs initial analysis and design and final system integration and testing. During the initial stages of software development, the software systems engineering function is responsible for software requirements analysis and architectural design. Software systems engineering is also responsible for the final testing of the software system and its delivery to the systems function. Actual component engineering, implementation, and testing are the dominion of software engineering in this view. A similar diagram can be drawn for any hardware items of a system, if such items are being developed or procured. This diagram should be contrasted with Figure 1.1, which shows the notional distribution of effort for systems engineering during the development process.

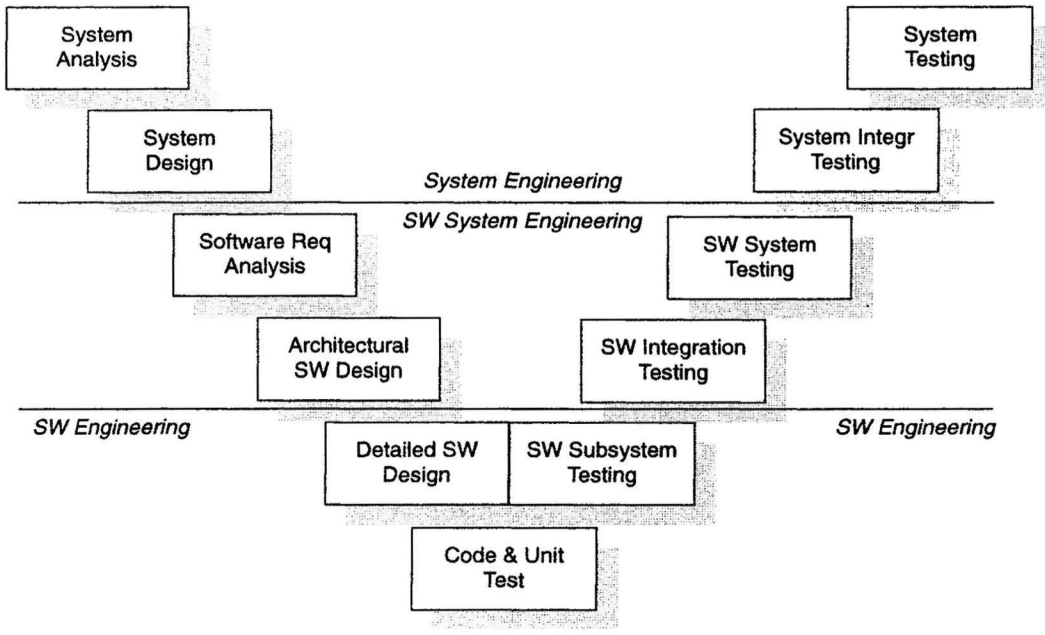

Figure 1.4: Engineering Activities and Product Flow

1.4.6 Relationship to Project Management

The *project management (PM) process* assesses the risks and costs of the software system, establishes the program master schedule, managerially integrates the various engineering specialties and design groups, maintains configuration control, and continuously audits the effort to ensure that cost and schedule are met and technical requirements objectives are satisfied. Many of these functions are performed by other organizations operating under direction from project management. For example, the configuration control function is typically performed by a group of that name, not by PM.

Figure 1.5 illustrates the relationships between PM, SwSE, and software engineering. PM has overall management responsibility for the project and the authority to commit resources. SwSE determines the "big picture" technical approach, interfaces with the technical customer, and approves and accepts the final software product. Software engineering is responsible for developing the software design, coding, and testing the design and, in general, developing the software configuration items (subsystems).

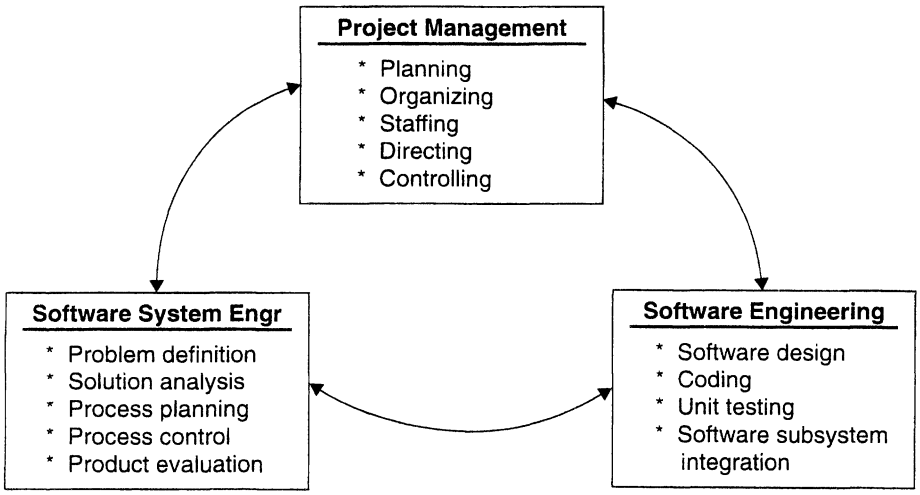

Figure 1.5: Management Relationships

1.4.7 Activities of Software Systems Engineering

The major activities of software systems engineering are listed next. Notice the parallels between these tasks and those performed by systems engineering in hybrid hardware/software developments (see Section 1.3.3). The terms in parentheses are the names commonly used when these functions are performed as part of an overall systems engineering effort.

- *Requirements analysis (problem definition)*—Determine the needs and constraints by analyzing the system requirements that have been allocated to software.

- *Architectural software design (solution analysis)*—Determine the set of solutions to the software requirements and constraints, conduct trade studies, perform analysis of possible solutions, and select the optimum one.
- *Process planning*—Determine the tasks to be done, the scope and the effort necessary to perform the tasks, the precedence between tasks, and the potential risks to the project.
- *Process control*—Determine the methods to be used in technically controlling the project and the processes, measure progress, review intermediate products, and take corrective action when necessary.
- *Verification, validation, and testing (product evaluation)*— Evaluate the final product and documentation through testing, demonstrations, analysis, and inspections. This includes any necessary software system integration activities.

The following sections of this chapter discuss these activities on a summary basis in order to illustrate their relationships. They are also discussed in more detail in separate chapters of this volume.

1.5 Software Requirements Analysis

Software requirements are (1) software capabilities needed by a user to solve a problem or achieve an objective and/or (2) capabilities that must be provided or possessed by a software system or software system component to satisfy a contract, standard, specification, or other formally imposed document [IEEE 610.12-1990].

The IEEE standard for software requirements specifications [IEEE 830-1998] partitions software requirements into the following categories:

- *Functional requirements*—Specify the functions that a system or system component must be capable of performing.
- *Performance requirements*—Specify performance characteristics that a system or system component must possess, such as speed, accuracy, frequency.
- *External interface requirements*—Specify hardware, software, or database elements with which a system or system component must interface; sets forth constraints on formats, timing, or other factors caused by such an interface.
- *Design constraints*—Affect or limit the design of a software system or software system component, for example, language requirements, physical hardware requirements, software development standards, and software quality assurance standards.
- *Quality attributes*—Specify the quality features, such as correctness, reliability, maintainability, and portability, that the product must possess.