Software System Engineering: A Tutorial

Applying system engineering principles specifically to the development of large, complex software systems provides a powerful tool for process and product management.

Richard H. Thayer
California State University, Sacramento

Software systems have become larger and more complex than ever. We can attribute some of this growth to advances in hardware performance—advances that have reduced the need to limit a software system's size and complexity as a primary design goal. Microsoft Word is a classic example: A product that would fit on a 360-Kbyte diskette 20 years ago now requires a 600-Mbyte CD.

But there are other reasons for increased size and complexity. Specifically, software has become the dominant technology in many if not most technical systems. It often provides the cohesiveness and data control that enable a complex system to solve problems.

Figure 1 is a prime example of this concept. In an air traffic control system, software connects the airplanes, people, radar, communications, and other equipment that successfully guide an aircraft to its destination. Software provides the system's major technical complexity.

The vast majority of large software systems do not meet their projected schedule or estimated cost, nor do they completely fulfill the system acquirer's expectations. This phenomenon has long been known as the *software crisis*.[1] In response to this crisis, software developers have introduced different engineering practices into product development.

Simply tracking a development project's managerial and technical status—resources used, milestones accomplished, requirements met, tests completed—does not provide sufficient feedback about its health. Instead, we must manage the technical processes as well as its products. System engineering provides the tools this technical management task requires.

The application of system engineering principles to the development of a computer software system produces activities, tasks, and procedures called *software system engineering*, or SwSE. Many practitioners consider SwSE to be a special case of system engineering, and others consider it to be part of software engineering. However, we can argue that SwSE is a distinct and powerful tool for managing the technical development of large software projects.

This tutorial integrates the definitions and processes from the IEEE software engineering standards[2] into the SwSE process. A longer version that includes a detailed step-by-step approach for implementing SwSE is available in *Software Engineering Volume 1: The Development Process*, part of the IEEE Computer Society's "best practices series."[3]

SYSTEMS AND SYSTEM ENGINEERING

A *system* is a collection of elements related in a way that allows a common objective to be accomplished. In computer systems, these elements include hardware, software, people, facilities, and processes.

System engineering is the practical application of scientific, engineering, and management skills necessary to transform an operational need into a description of a system configuration that best satisfies that need. It is a generic problem-solving process that applies to the overall technical management of a system development project. This process provides the mechanism for identifying and evolving a system's product and process definitions.

0018-9162/02/$17.00 © 2002 IEEE

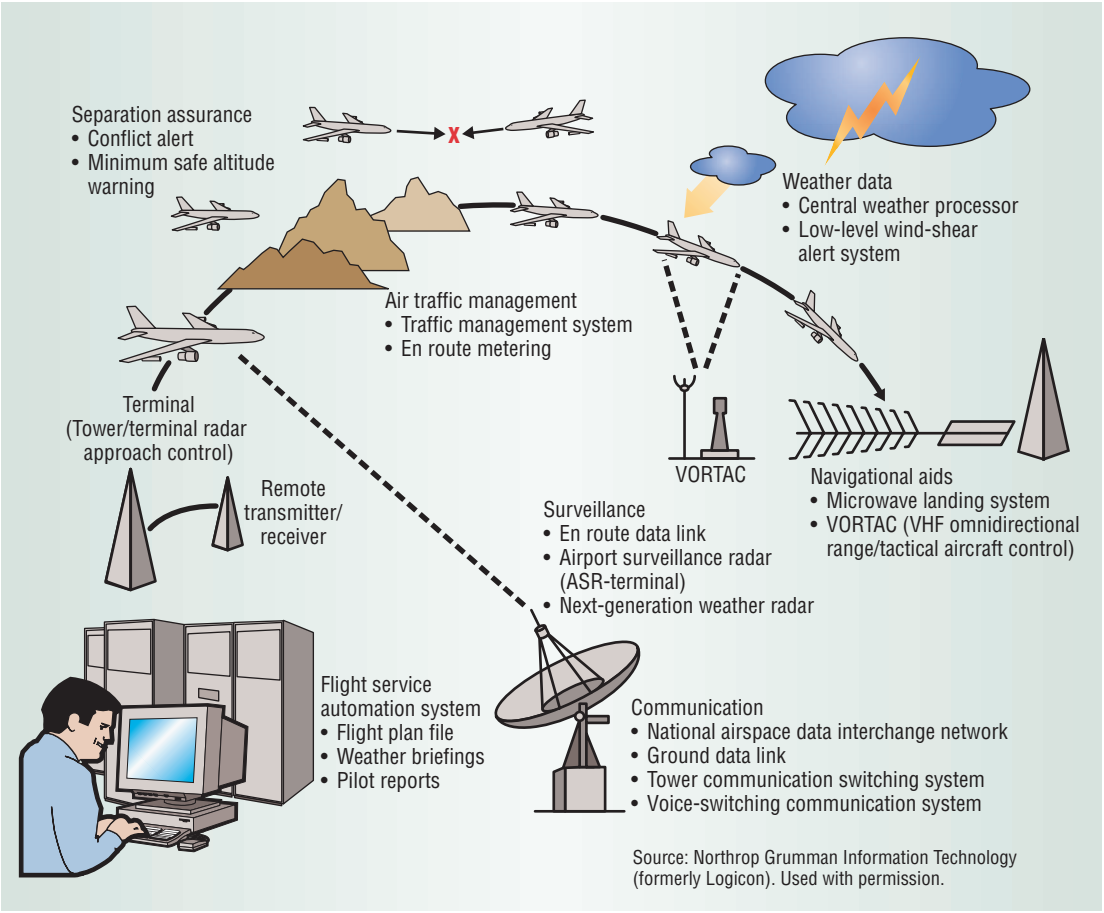

Figure 1. Air traffic control system environment. Software ties the elements of large systems together, and is frequently the most complex and technically challenging part of the system.

IEEE Std. 1220-1998 describes the system engineering process and its application throughout the product life cycle.[4] System engineering produces documents, not hardware. The documents associate developmental processes with the project's lifecycle model. They also define the expected process environments, interfaces, products, and risk management tools throughout the project.

System engineering involves five functions:

- *Problem definition* determines the needs and constraints through analyzing the requirements and interfacing with the acquirer.
- *Solution analysis* determines the set of possible ways to satisfy the requirements and constraints, analyzes the possible solutions, and selects the optimum one.
- *Process planning* determines the tasks to be done, the size and effort to develop the product, the precedence between tasks, and the potential risks to the project.
- *Process control* determines the methods for controlling the project and the process, measures progress, reviews intermediate products, and takes corrective action when necessary.
- *Product evaluation* determines the quality and quantity of the delivered product through evaluation planning, testing, demonstration, analysis, examination, and inspection.

System engineering provides the baseline for all project development, as well as a mechanism for defining the *solution space*—that is, the systems and the interfaces with outside systems. The solution space describes the product at the highest level—before the system requirements are partitioned into the hardware and software subsystems.

This approach is similar to the software engineering practice of specifying constraints as late as possible in the development process. The further into the process a project gets before defining a constraint, the more flexible the implemented solution will be.

WHAT IS SOFTWARE SYSTEM ENGINEERING?

The term software system engineering dates from the early 1980s and is credited to Winston W. Royce,[5] an early leader in software engineering. SwSE is responsible for the overall technical management of the system and the verification of the final system products. As with system engineering, SwSE produces documents, not components. This differentiates it from software engineering (SwE), which produces computer programs and users' manuals.

SwSE begins after the system requirements have been partitioned into hardware and software subsystems. SwSE establishes the baseline for all project software development. Like SwE, it is both a technical and a management process. The SwSE tech-

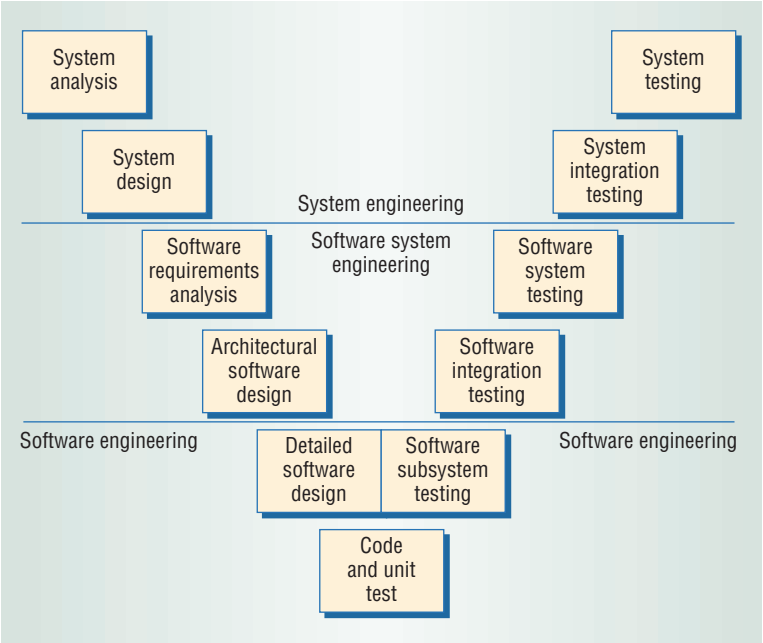

Figure 2. Engineering relationships between system engineering, software system engineering (SwSE), and software engineering. SwSE is responsible for requirements analysis, architectural design, and final software-system testing.

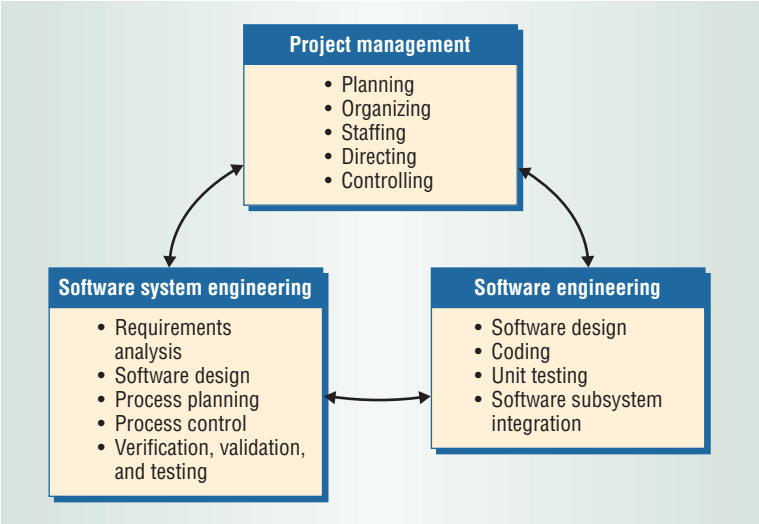

Figure 3. Management relationships between software system engineering (SwSE), software engineering, and project management. SwSE is responsible for determining the technical approach.

nical process is the analytical effort necessary to transform an operational need into

- a software system description;
- a software design of the proper size, configuration, and quality;
- software system documentation in requirements and design specifications;
- the procedures necessary to verify, test, and accept the finished software product; and
- the documentation necessary to use, operate, and maintain it.

SwSE is not a job description. It is a process that many people and organizations perform: system engineers, managers, software engineers, programmers, and—not to be ignored—acquirers and users.

As large system solutions become increasingly dependent on software, a system engineering approach to software development can help avoid the problems associated with the software crisis.

Software developers often overlook system engineering and SwSE in their projects. They consider systems that are all software or that run on commercial off-the-shelf computers to be just software projects, not system projects. Ignoring the systems aspects of software development contributes to our long-running software crisis.

SwSE and software engineering

Both SwSE and SwE are technical and management processes, but SwE produces software components and their supporting documentation. Specifically, 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 time and budget constraints; and
- the systematic application of methods, tools, and techniques that achieve a stated requirement or objective for an effective and efficient software system.

Figure 2 illustrates the engineering relationships between system engineering, SwSE, and SwE. Traditional system engineering does initial analysis and design as well as final system integration and testing.

During the initial stage of software development, SwSE is responsible for software requirements analysis and architectural design. SwSE also manages the final testing of the software system. Finally, SwE manages what system engineers call *component engineering*.

SwSE and project management

The project management process involves assessing the software system's risks and costs, establishing a schedule, integrating the various engineering specialties and design groups, maintaining configuration control, and continuously auditing

Table 1. System engineering functions correlated to software system engineering (SwSE).

System engineering function	SwSE function	SwSE function description
Problem definition	Requirements analysis	Determine needs and constraints by analyzing system requirements allocated to software
Solution analysis	Software design	Determine ways to satisfy requirements and constraints, analyze possible solutions, and select the optimum one
Process planning	Process planning	Determine product development tasks, precedence, and potential risks to the project
Process control	Process control	Determine methods for controlling project and process, measure progress, and take corrective action where necessary
Product evaluation	Verification, validation, and testing	Evaluate final product and documentation

the effort to ensure that the project meets costs and schedules and satisfies technical requirements.[6]

Figure 3 illustrates the management relationships between project management, SwSE, and SwE. Project management has overall management responsibility for the project and the authority to commit resources. SwSE determines the technical approach, makes technical decisions, interfaces with the technical acquirer, and approves and accepts the final software product. SwE is responsible for developing the software design, coding the design, and developing software components.

THE FUNCTIONS OF SOFTWARE SYSTEM ENGINEERING

Table 1 lists the five main functions of system engineering correlated to SwSE, along with a brief general description of each SwSE function.

Requirements analysis

The first step in any software development activity is to determine and document the system-level requirements in either a system requirements specification (SRS) or a software requirements specification or both. Software requirements include capabilities that a user needs to solve a problem or achieve an objective as well as capabilities that a system or component needs to satisfy a contract, standard, or other formally imposed document.[7]

We can categorize software requirements as follows:[8]

- *Functional requirements* specify 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, and frequency.
- *External interface requirements* specify hardware, software, or database elements with which a system or component must interface, or set forth constraints on formats, timing, or other factors caused by such an interface.
- *Design constraints* affect or constrain 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 degree to which software possesses attributes that affect quality, such as correctness, reliability, maintainability, and portability.

Software requirements analysis begins after system engineering has defined the acquirer and user system requirements. Its functions include identification of all—or as many as possible—software system requirements, and its conclusion marks the established requirements baseline, sometimes called the allocated baseline.[2]

Software design

Software design is the process of selecting and documenting the most effective and efficient system elements that together will implement the software system requirements.[9] The design represents a specific, logical approach to meet the software requirements.

Software design is traditionally partitioned into two components:

- *Architectural design* is equivalent to system design, during which the developer selects the system-level structure and allocates the software requirements to the structure's components. Architectural design—sometimes called top-level design or preliminary design—typically defines and structures computer program components and data, defines the interfaces, and prepares timing and sizing estimates. It includes information such as the overall processing architecture, function allocations (but not detailed descriptions), data flows, system utilities, operating system interfaces, and storage throughput.

Table 2. Process planning versus project planning.

Software system engineering planning activities	Project management planning activities
Determine tasks to be done	Determine skills necessary to do the tasks
Establish order of precedence between tasks	Establish schedule for completing the project
Determine size of the effort (in staff time)	Determine cost of the effort
Determine technical approach to solving the problem	Determine managerial approach to monitoring the project's status
Select analysis and design tools	Select planning tools
Determine technical risks	Determine management risks
Define process model	Define process model
Update plans when the requirements or development environment change	Update plans when the managerial conditions and environment change

Table 3. Process control versus project control.

Software system engineering control activities	Project management control activities
Determine the requirements to be met	Determine the project plan to be followed
Select technical standards to be followed, for example, IEEE Std. 830	Select managerial standards to be followed, for example, IEEE Std. 1058
Establish technical metrics to control progress, for example, requirements growth, errors reported, rework	Establish management metrics to control progress, for example, cost growth, schedule slippage, staffing shortage
Use peer reviews, in-process reviews, software quality assurance, VV&T, and audits to determine adherence to requirements and design	Use joint acquirer-developer (milestone) reviews and SCM to determine adherence to cost, schedule, and progress
Reengineer the software requirements when necessary	Replan the project plan when necessary

- *Detailed design* is equivalent to component engineering. The components in this case are independent software modules and artifacts.

The methodology proposed here allocates architectural design to SwSE and detailed design to SwE.

Process planning

Planning specifies the project goals and objectives and the strategies, policies, plans, and procedures for achieving them. It defines in advance what to do, how to do it, when to do it, and who will do it.

Planning a SwE project consists of SwSE management activities that lead to selecting a course of action from alternative possibilities and defining a program for completing those actions.

There is an erroneous assumption that project management performs all project planning. In reality, project planning has two components—one accomplished by project management and the other by SwSE—and the bulk of project planning is a SwSE function. (This is not to say that project managers might not perform both functions.)

Table 2 shows an example partitioning of planning functions for a software system project.

Process control

Control is the collection of management activities used to ensure that the project goes according to plan. Process control measures performance and results against plans, notes deviations, and takes corrective actions to ensure conformance between plans and actual results.

Process control is a feedback system for how well the project is going. Process control asks questions such as: Are there any potential problems that will cause delays in meeting a particular requirement within the budget and schedule? Have any risks turned into problems? Is the design approach still doable?

Control must lead to corrective action—either bringing the status back into conformance with the plan, changing the plan, or terminating the project.

Project control also has two separate components: control that project management accomplishes and control that software systems engineering accomplishes. Table 3 shows an example partitioning of control functions for a software system project.

Verification, validation, and testing

The verification, validation, and testing (VV&T) effort determines whether the engineering process is correct and the products are in compliance with their requirements.[10] The following critical definitions apply:

- *Verification* determines whether the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. Verification answers the question, "Am I building the product right?"
- *Validation* determines the correctness of the final program or software with respect to the user's needs and requirements. Validation answers the question, "Am I building the right product?"
- *Testing* is the execution of a program or partial program, with known inputs and outputs that are both predicted and observed, for the purpose of finding errors. Testing is frequently considered part of validation.

V&V is a continuous process of monitoring system engineering, SwSE, SwE, and project management activities to determine that they are following the technical and managerial plans, specifications,

standards, and procedures. V&V also evaluates the SwE project's interim and final products. Interim products include requirements specifications, design descriptions, test plans, and review results. Final products include software, user's manuals, training manuals, and so forth.

Any individual or functions within a software development project can do V&V. SwSE uses V&V techniques and tools to evaluate requirements specifications, design descriptions, and other interim products of the SwSE process. It uses testing to determine if the final product meets the project requirements specifications.

The last step in any software development activity is to validate and test the final software product against the software requirements specification and to validate and test the final system product against the SRS.

System engineering and SwSE are disciplines used primarily for technical planning in the front end of the system life cycle and for verifying that the plans were met at the project's end. Unfortunately, a project often overlooks these disciplines, especially if it consists entirely of software or runs on commercial off-the-shelf computers.

Ignoring the systems aspects of any software project can result in software that will not run on the hardware selected or will not integrate with other software systems. ■

References

1. W.W. Gibbs, "Software's Chronic Crisis," *Scientific Am.*, Sept. 1994, pp. 86-95.
2. IEEE, *Software Engineering Standards Collection*, vols. 1-4, IEEE Press, Piscataway, N.J., 1999.
3. R.H. Thayer, "Software System Engineering: A Tutorial," *Software Engineering Volume 1: The Development Process,* 2nd ed., R.H. Thayer and M. Dorfman, eds., IEEE CS Press, Los Alamitos, Calif., 2002, pp. 97-116.
4. *IEEE Std. 1220-1998, Standard for Application and Management of the System Engineering Process*, IEEE Press, Piscataway, N.J., 1998.
5. W.W. Royce, "Software Systems Engineering," seminar presented as part of the course titled Management of Software Acquisition, Defense Systems Management College, Fort Belvoir, Va., 1981-1988.
6. *IEEE Std. 1058-1998*, *Standard for Software Project Management Plans*, IEEE Press, Piscataway, N.J., 1998.
7. *IEEE Std. 610.12-1990*, *Standard Glossary of Software Engineering Terminology*, IEEE Press, Piscataway, N.J., 1990.
8. *IEEE Std. 830-1998*, *Recommended Practice for Software Requirements Specifications*, IEEE Press, Piscataway, N.J., 1998.
9. *IEEE Std. 1016-1998, Recommended Practice for Software Design Descriptions*, IEEE Press, Piscataway, N.J., 1998.
10. *IEEE Std. 1012-1998, Standard for Software Verification and Validation*, IEEE Press, Piscataway, N.J., 1998.

Richard H. Thayer *is an emeritus professor in software engineering at California State University, Sacramento. He is also a consultant in software engineering and project management and a visiting researcher and lecturer at the University of Strathclyde, Glasgow, Scotland. He received a PhD in electrical engineering from the University of California at Santa Barbara. Thayer is a Fellow of the IEEE, an Associate Fellow of the American Institute of Aeronautics and Astronautics, and a member of the IEEE Computer Society and the ACM. Contact him at r.thayer@computer.org.*