Software System Engineering: A Tutorial

Dr. Richard Hall Thayer
Software Management Training LLC

Applying system-engineering principles specifically to the development of large, complex software systems provides a powerful tool for process and product management. Software engineering has its early roots in system engineering which is reflected in their many common terms. This article discusses a merger between system engineering and software engineering called software system engineering. System engineering looks at controlling the total system development including software. Software engineering looks at controlling just software development. (System engineers would call software engineering component engineering.) The application of system engineering to the development of software gives a large measure of control software development.

Applying system engineering principles specifically to the development of large, complex software systems provides a powerful tool for process and product management. This process is called software engineering. Dr. Winston Royce, father of the Waterfall chart, points out that software engineering was developed from system engineering, and he argued for calling the union software system engineering. Unfortunately, this did not stick, and software engineering and software system engineering can be viewed as separate processes. 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-kilobyte diskette 20 years ago now requires a 600-megabyte compact disc.

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. When the Federal Aviation Administration systems were upgraded to automation back in the 1960s, the much larger systems could handle many more aircraft over a larger terrain. However, these larger systems continued to use much of the earlier 1950s hardware; it was the software that enabled larger groups of hardware to work together towards the common goal of safely delivering an aircraft from takeoff to landing. Software provides the system's major technical complexity.

Because of this increase in size and 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[1]. 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.

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.

Simply tracking a development project's managerial and technical status – resources used, milestones accomplished, requirements met, and tests completed – does not provide sufficient feedback about the project's health. Instead, we must manage the *technical processes* as well as its products. System engineering provides the tools the 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 (SwSE). Many practitioners consider SwSE to be a special case of system engineering and others consider it to be part of software engineering. However, it can be argued that SwSE is a distinct and powerful tool for managing the technical development of large software projects.

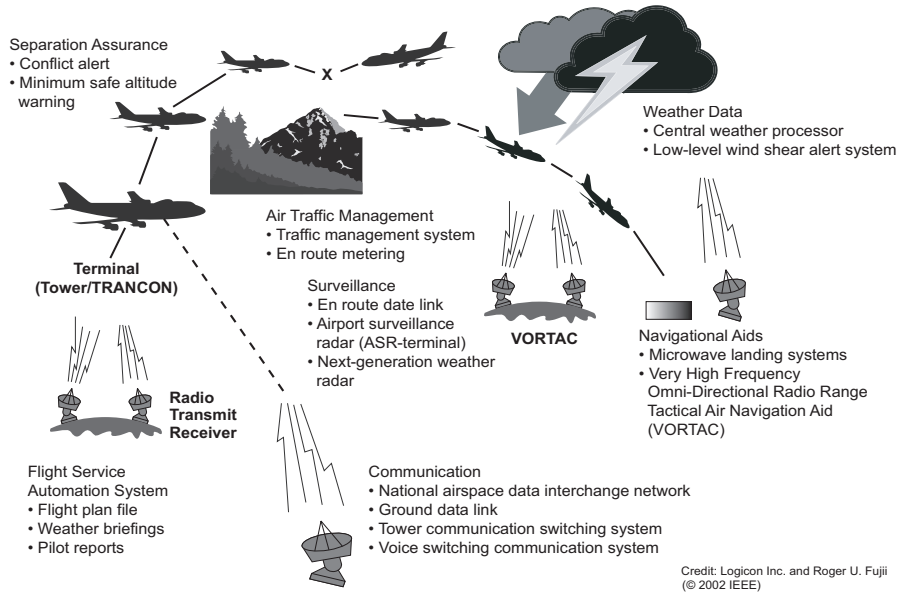
This tutorial integrates the definitions and processes from the Institute of Electrical and Electronics Engineers (IEEE) software engineering standards [2] into the SwSE process.

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 appli-

Figure 1: *Software Ties the System Together*



Credit: Logicon Inc. and Roger U. Fujii
(© 2002 IEEE)

Portions of this article appeared in Computer, Apr. 2002. © 2003 IEEE. Reprinted with permission from *Computer*, Vol.35, Issue 4, pp. 68-73, Apr. 2002.

cation 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.

IEEE Std. 1220-1998 describes the system engineering process and its application throughout the product life cycle [3]. *System engineering produces documents, not hardware.* These documents are associated with the developmental processes within the project's life-cycle 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*. 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 SwSE?

The term *software system engineering* dates from the early 1980s and is credited to Dr. Winston Royce [4], 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, which produces computer programs and user 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 software engineering, it is both a technical and a management process. The SwSE technical process is the analytical effort necessary to transform user operational needs into the following:

- A software system description.
- Software system requirements and design specifications.
- Necessary procedures to verify, test, and accept the finished software product.
- Necessary documentation 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.

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 (COTS) 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

Early in my software engineering career, I was informed that software engineering was the engineering of software copied from the hardware engineers (e.g. electrical engineers, mechanical engineers, and so forth). I was well acquainted with the mechanics of software engineering which made it different from computer science. The following are examples of what makes the mechanics of software engineering different than computer science:

- Dividing the project into phases such as life-cycle development methods.
- Managing software as a separate project.
- Using intermediate products (specifications), e.g., requirements specifications, design specifications.
- Reviewing, testing, and auditing.
- Using configuration management and quality (process) assurance.
- Prototyping and the reuse of existing components.

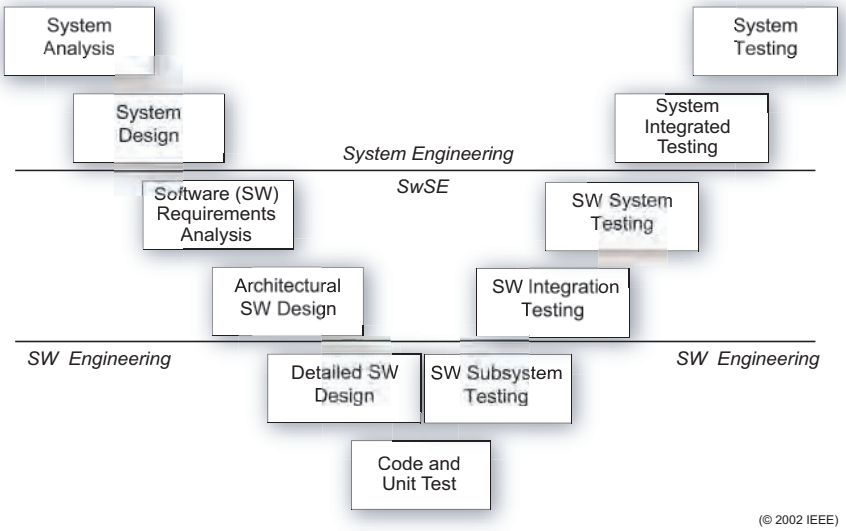
Later, I had an opportunity to mingle with a number of engineers from a conventional engineering discipline, and I asked them about some of our software engineering processes such as those listed above. Most had no idea what I was referring to.

Then by chance, I had an opportunity to work for a company that used system engineering. I then realized that I had found the source of software engineering processes: It was system engineering.

Both SwSE and software engineering are technical and management processes, but software engineering produces software components and their supporting documentation. Specifically, software engineering is the following:

- The practical application of computer science, management, and other sci-

Figure 2: *Engineering Relationships*



ences 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.
- 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 software engineering. 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 *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 the effort to ensure that the project meets costs and schedules and satisfies technical requirements [5].

Figure 3 illustrates the management relationships between project management, SwSE, and software engineering. 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 the final software product. Software engineering is responsible for developing the software design, coding the design, and developing software components.

The Functions of SwSE

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 [6].

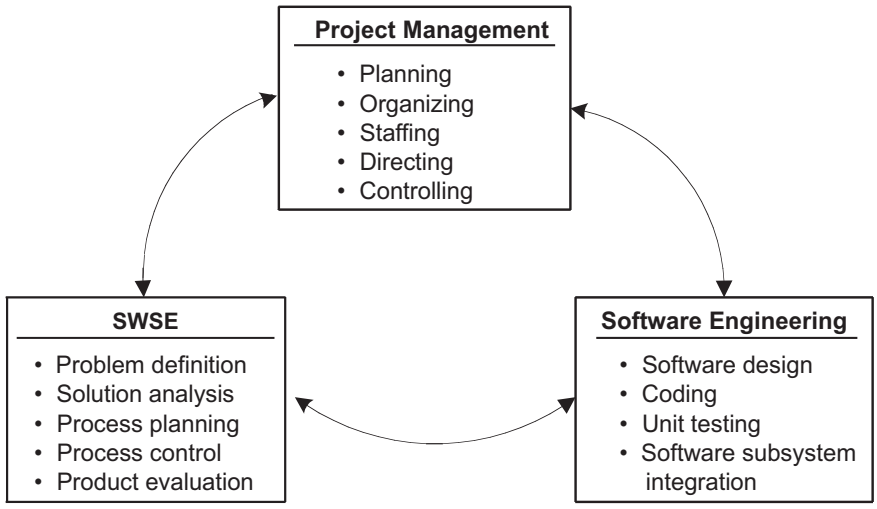


Figure 3: *Management Relationships*

We can categorize software requirements as follows [7]:
- *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.

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 [8]. The design represents a specific, logical approach to meet the software requirements.

Software design is traditionally partitioned into two components:

Table 1: *System Engineering Functions Correlated to 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 (VV&T)	Evaluate final product and documentation.

(© 2002, IEEE)

SwSE Planning Activities	Project Management Planning Activities
Determines tasks to be done.	Determines skills necessary to do the tasks.
Establishes order of precedence between tasks.	Establishes schedule for completing the project.
Determines size of the effort.	Determines cost of the effort (in staff time).
Determines technical approach to solving the problem.	Determines managerial approach to monitoring the project's status.
Selects analysis and design tools.	Selects planning tools.
Determines technical risks.	Determines management risks.
Defines process model.	Defines process model.
Updates plans when the requirements or development environment change.	Updates plans when the managerial conditions and environment change.

Table 2: *Process Planning Versus Project Planning*

- *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.
- *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 software engineering.

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 software engineering 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 an 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 the following: 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.

Table 3: *Process Control Versus Project Control*

SwSE Control Activities	Project Management Control Activites
Determines the requirements to be met.	Determines the project plan to be followed.
Selects technical standards to be followed, for example, IEEE Std. 830 [7].	Selects managerial standards to be followed, for example, IEEE Std. 1058 [5].
Establishes technical metrics to control progress, for example, requirements growth, errors reported, or rework.	Establishes management metrics to control progress, for example, cost growth schedule slippage, or staffing shortages.
Uses peer reviews, in-process reviews, software quality assurance, VV&T, and audits to determine adherence to requirements and design.	Uses joint acquirer-developer (milestone) reviews and software configuration management to determine adherence to cost, schedule, and progress.
Re-engineers the software requirements when necessary.	Replans the project plan when necessary.

VV&T

The *VV&T* effort determines whether the engineering process is correct and the products are in compliance with their requirements [9]. 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.

Verification and Validation (V&V) is a continuous process of monitoring system engineering, SwSE, software engineering, and project management activities to determine that they are following the technical and managerial plans, specifications, standards, and procedures. V&V also evaluates the software engineering project's interim and final products. Interim products include requirements specifications, design descriptions, test plans, and review results. Final products include software, user manuals, training manuals, and so forth.

Any individual or function 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 COTS computers.

Summary and Conclusions

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.

Conducting *software engineering* without conducting SwSE puts a project in jeopardy of being incomplete or having components which do not work together, and/or exceeding the project's scheduled budget.

Software engineering and SwSE are primarily disciplines used in the front end of the system life cycle for technical planning and at the very late part of the life cycle to verify if the plans have been met. A review of the emphasis in this article will show that much of the work of planning and SwSE is done during the top-level requirements analysis and top-level design phases. The other major activity of SwSE is the final validation and testing of the completed system.

Software engineering principles, activities, tasks, and procedures can be applied to software development. This article has summarized, in broad steps, what is necessary to implement SwSE on either a hardware-software system (that is primarily software) or on an almost total software system. SwSE is not cheap, but it is cost effective.◆

References

1. Gibbs, W.W. "Software's Chronic Crisis." Scientific American Sept. 1994: 86-95.
2. IEEE. Software Engineering Standards Collection. Vol. 1-4. Piscataway: IEEE Press, 1999.
3. IEEE. Standard for Application and Management of the System Engineering Process. Std. 1220-1998, Piscataway: IEEE Press, 1998.
4. Royce, W.W. "Software Systems Engineering." Management of Software Acquisition. Fort Belvoir, VA: Defense Systems Management College, 1981-1988.
5. IEEE. Standard for Software Project Management Plans. Std. 1058-1998. Piscataway: IEEE Press, 1998.
6. IEEE. Standard Glossary of Software Engineering Terminology. Std. 610. 12-1990. Piscataway: IEEE Press, 1990.
7. IEEE. Recommended Practice for Software Requirements Specifications. Std. 830-1998. Piscataway: IEEE Press, 1998.
8. IEEE. Recommended Practice for Software Design Descriptions. Std. 1016-1998. Piscataway: IEEE Press, 1998.
9. IEEE. Standard for Software Verification and Validation. Std. 1012-1998. Piscataway: IEEE Press, 1998.

Note

1. This article uses the definitions from IEEE/EIA 12207.0-1997, where acquirer is used for customer and supplier is used for developer or contractor.

About the Author

Richard Hall Thayer, Ph.D., is a senior lecturer for Software Management Training, LLC, and is a professor emeritus in software engineering at California State University, Sacramento. He is a retired Air Force colonel and managed many of the Air Force's software engineering projects. Thayer is also a consultant in software engineering and project management and a visiting researcher and lecturer at the University of Strathclyde, Glasgow, Scotland. He has written more than 50 papers and books on software engineering, including two software engineering standards. Thayer received his doctorate in electrical engineering from the University of California at Santa Barbara and a master's and bachelor's in engineering degrees from the University of Illinois at Champaign/Urbana.

Software Management Training, LLC
6540 Chiquita WY
Carmichael, CA 95608
E-mail: r.thayer@computer.org

Coming Events

October 2-3
Department of Homeland Security Department of Defense Software Assurance Forum
Tysons Corner, VA
https://buildsecurityin.us-cert.gov/daisy/bsi/events.html

November 4-7
AYE 2007
Amplifying Your Effectiveness
Phoenix, AZ
www.ayeconference.com/conference.html

November 8-9
Static Analysis Summit II
Fairfax, VA
https://buildsecurityin.us-cert.gov/daisy/bsi/events.html

November 12-16
ICSPI 2007
Orlando, FL
www.icspi.com

November 14-16
The 10th IEEE High Assurance Systems Engineering Symposium
Dallas, TX
http://hase07.utdallas.edu

November 19-21
The 11th International Conference on Software Engineering and Applications
Cambridge, MA
www.iasted.org/conferences/cfp-591.html.

May 2008

Systems and Software Technology Conference
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**