What Is Systems Engineering?
A Consensus of Senior Systems Engineers

Facilitated by
A. Terry Bahill
Systems and Industrial Engineering
University of Arizona
Tucson, AZ 85721-0020
terry@sie.arizona.edu

Frank F. Dean
Sandia National Laboratories
Albuquerque, NM 87185
fdean@ktech.com

Copyright © 1994 to 2009 by Terry Bahill and Frank Dean

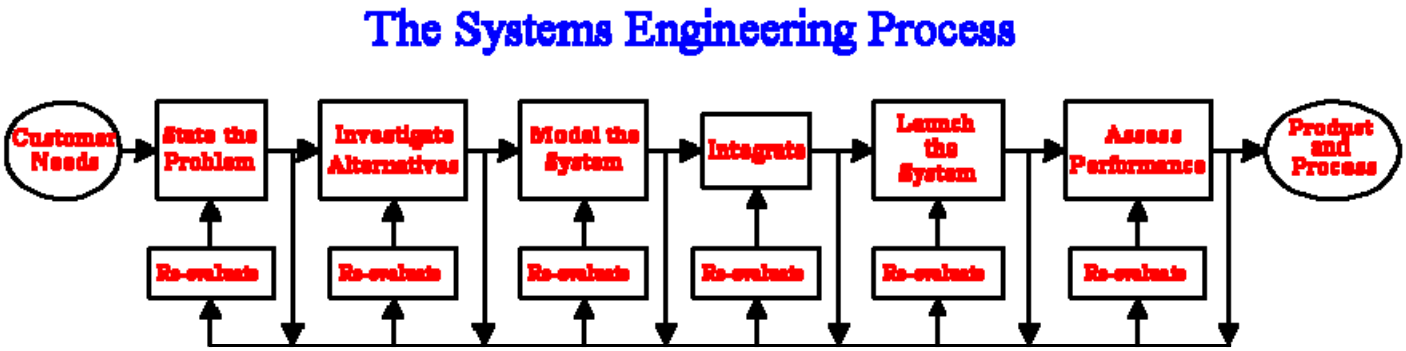
Last changed January 15, 2009.

Abstract

Systems Engineering is an interdisciplinary process that ensures that the customer's needs are satisfied throughout a system's entire life cycle. This process is comprised of the following seven tasks.

1. ***S**tate the problem.* Stating the problem is the most important systems engineering task. It entails identifying customers, understanding customer needs, establishing the need for change, discovering requirements and defining system functions.
2. ***I**nvestigate alternatives.* Alternatives are investigated and evaluated based on performance, cost and risk.
3. ***M**odel the system.* Running models clarifies requirements, reveals bottlenecks and fragmented activities, reduces cost and exposes duplication of efforts.
4. ***I**ntegrate*. Integration means designing interfaces and bringing system elements together so they work as a whole. This requires extensive communication and coordination.
5. ***L**aunch the system.* Launching the system means running the system and producing outputs -- making the system do what it was intended to do.
6. ***A**ssess performance.* Performance is assessed using evaluation criteria, technical performance measures and measures -- measurement is the key. If you cannot measure it, you cannot control it. If you cannot control it, you cannot improve it.
7. ***R**e-evaluation.* Re-evaluation should be a continual and iterative process with many parallel loops.

This process can be summarized with the acronym SIMILAR (Bahill and Gissing, 1998).



This figure is from Bahill and Gissing (1998).

The purpose of systems engineering is to produce systems that satisfy the customers' needs, increase the probability of system success, reduce risk and reduce total-life-cycle cost.

Material for this paper was gathered from senior Systems Engineers at BAE Systems, Sandia National Laboratories, Hughes Missile Systems, Lockheed Martin Tactical Defense Systems, The Boeing Company, and Idaho National Engineering and Environmental Laboratories, and also the following general references: Chapman, Bahill and Wymore (1992); Wymore (1993); IEEE P1220 (1994); Grady (1994, 1995); Hughes Aircraft Company (1994); Martin-Marietta (1994); Shishko and Chamberlain (1995); Martin (1997); Blanchard and Fabrycky (1998); EIA-632 (1999); Sage and Rouse (1999); DSMC (1999); Buede (2000); Rechtin (2000); Proceedings of the IEEE Systems, Man, and Cybernetics International Conferences; NCOSE and INCOSE Symposia and Proceedings.

Earlier versions of this paper were published in the *IEEE Systems, Man, and Cybernetics Newsletter* December 1994, pp. 11-12 and in the *Proceedings of the Sixth Annual Symposium of the International Council on Systems Engineering (INCOSE)*, July 7-11, 1996, Boston, Vol. 1, pp. 503-508.

The system life cycle

The system life cycle has seven phases: (1) discovering system requirements, (2) investigating alternatives, (3) full-scale engineering design, (4) implementation, (5) integration and test, (6) operation, maintenance and evaluation and (7) retirement, disposal and replacement. However, the system life cycle is different for different industries, products and customers. Chapman, Bahill and Wymore (1992); Wymore (1993); Kerzner (1995); Shishko and Chamberlain (1995).

State the problem

The problem statement starts with a description of the top-level function that the system must perform or the deficiency that must be ameliorated. It includes system requirements stated in terms of what must be done, not how to do it. It might be composed in words or as a model. Inputs come from end users, operators, bill payers, owners, regulatory agencies, victims, sponsors, Marketing, Manufacturing, etc. These are called stakeholders. Identifying the stakeholders is an important initial task. In a modern business environment, the problem statement starts with a reason for change followed by vision and mission statements for the company. The problem statement is one of Systems Engineering's most important products. An elegant solution to the wrong problem is less than worthless.

The problem statement should not use the word optimal. The word *optimal* should not appear in the statement of the problem, because there is no single optimal solution to complex systems problems. Most system designs have several performance and cost criteria. Systems Engineering creates a set of alternative designs that satisfy

these performance and cost criteria to varying degrees. Moving from one alternative to another will usually improve at least one criterion and worsen at least one criterion, i.e. there will be trade-offs. None of the feasible alternatives is likely to optimize all the criteria. Therefore, Systems Engineers must settle for less than optimality. No complex system is likely to be optimal to all the people, all the time. It might be possible to optimize some subsystems, but when they are interconnected, the overall system may not be optimal. The best possible system may not be that made up of optimal subsystems. An all star team might have the optimal people at all positions, but is it likely that such an all star team could beat the world champions? e.g., a Pro Bowl team is not likely to beat the Super Bowl champions. If the system requirements demanded an optimal system, data could not be provided to prove that any resulting system was indeed optimal. In general, it can be proven that a system is at a local optimum, but it cannot be proven that it is at a global optimum.

Understand customer needs

Customers seldom know what they want or need. Systems Engineers must enter the customer's environment and find out how the customer will use the system. We must exceed, not merely meet, customer expectations. Flexible designs and rapid prototyping help identify aspects that might have been overlooked. Talking to your customer's customer and your supplier's supplier can be very useful. The terms *customer* and *stakeholder* include anyone who has a right to impose requirements on the system: end users, operators, owners, bill payers, regulatory agencies, beneficiaries, victims, etc. Chapman, Bahill and Wymore (1992); Wymore (1993). Frameworks, such as the Zachman framework or the DoDAF, are useful for seeing how the system fits into the customer's enterprise (Bahill, Botta and Daniels, 2006).

Discover system requirements

There are two types of system requirements: mandatory and tradeoff (Bahill and Dean, 1999). Mandatory requirements insure that the system satisfies the customer's operational need. Mandatory requirements (1) specify the necessary and sufficient conditions that a minimal system must have in order to be acceptable (They are usually written with the words *shall* or *must*.), (2) must be passed or failed, there is no middle ground (i.e. They do not use scoring functions.), and (3) must not be susceptible to trade-offs between requirements. Typical mandatory requirements might be of the following form: The system shall not violate federal, state or local laws. Mandatory requirements state the minimal requirements necessary to satisfy the customer's need.

After understanding the mandatory requirements, Systems Engineers propose alternative candidate designs, all of which satisfy the mandatory requirements. Then the tradeoff requirements are evaluated to determine the preferred designs. Tradeoff requirements (1) should state conditions that would make the customer happier (They are often written with the word *should*.), (2) should use scoring functions (Daniels, Werner and Bahill, 2001: Wymore, 1993) to evaluate the criteria, and (3) should be evaluated with multicriterion decision aiding techniques (Szidarovszky, Gershon and Duckstein, 1986; Daniels, Werner and Bahill, 2001) because there will be trade-offs between these requirements. Typical tradeoff requirements might be of the following form: Dinner should have items from each of the four major food groups.

Sometimes there is a relationship between mandatory and tradeoff requirements, e.g. a mandatory requirement might be a lower threshold value for a tradeoff requirement. The words optimize, maximize, minimize and simultaneous should not be used in stating requirements (Grady, 1993). Quality function deployment (QFD) is useful in identifying system requirements (Bahill and Chapman, 1993; Bicknell and Bicknell, 1994).

Verify and validate requirements

Each requirement should be verified by logical argument, inspection, modeling, simulation, analysis, test or demonstration.
Validating requirements means ensuring that (1) the recommended solution satisfies the actual needs of the

customer, (2) the description of the requirements is consist and complete, (3) a system model can satisfy the requirements and (4) a real-world solution can be tested to prove that it satisfies the requirements. If Systems Engineering discovers that the customer has requested a perpetual-motion machine, the project should be stopped. Requirements are often validated by reference to an existing system that meets most of the requirements.

Investigate alternatives

Alternative designs are evaluated based on performance, cost, schedule and risk criteria. No design is likely to be best on all criteria, so multicriteria decision aiding techniques should be used to reveal the preferred alternatives. This analysis should be redone whenever more data are available. For example, criteria should be computed initially based on estimates by the design engineers. Then models should be constructed and evaluated. Next simulation data should be derived. Subsequently prototypes should be measured and finally tests should be run on the real system. For the design of complex systems, study of alternative designs reduces project risk. Investigating bizarre alternatives helps clarify the problem statement. This is one of the main processes used to define system architecture.

For important decisions formal tradeoff studies should be performed. A tradeoff study is not something that you do once at the beginning of a project. Throughout a project you are continually making tradeoffs: creating team communication methods, selecting components, choosing implementation techniques, designing test programs, or maintaining schedule. Many of these tradeoffs should be formally documented. These are the components of a tradeoff study: Problem statement, Evaluation criteria, Weights of importance, Alternative solutions, Evaluation data, Scoring functions, Scores, Combining functions, Preferred alternatives, and Sensitivity analysis (Smith, Son, Piattelli-Palmarini and Bahill, 2007).

Define quantitative measures

Evaluation criteria, technical performance measures and measures are all used to quantify system performance. These terms are often used interchangeably, but we think a distinction is useful. Evaluation criteria (which are often called figures of merit) are used to quantify requirements. Technical performance measures are used to mitigate risk during design and manufacturing. Measures (or metrics) are used to help manage a company's processes. (Moody, et al., 1997).

Performance and cost criteria show how well the system satisfies its requirements, e.g., In this test the car accelerated from 0 to 60 in 6.5 seconds. Evaluation criteria are often called Measures of Effectiveness. Such measurements are made throughout the evolution of the system: based first on estimates by the design engineers, then on models, simulations, prototypes and finally on the real system. Evaluation criteria are used to help select amongst alternative designs and they are used to quantify system requirements. During concept selection, criteria are traded-off, that is, going from one alternative to another increases one criterion and decreases another. Evaluation criteria should be the basis for most requirements.

Technical performance measures (TPM's) are used to track the progress of design and manufacturing. They are measurements that are made during the design and manufacturing process to evaluate the likelihood of satisfying the system requirements. Not all requirements have TPMs. TPMs are usually associated only with high risk requirements, because they are expensive to maintain and track. Early prototypes will not meet TPM goals. Therefore the TPM values are only required to be within a tolerance band. It is hoped that as the design and manufacturing process progresses the TPM values will come closer and closer to the goals.

Measures are often related to the process, not the product (Moody et al., 1997). Therefore, they do not always relate to specific system requirements. Rather some measures relate to the company's mission statement and subsequent goals. A useful measure is the percentage of requirements that have changed after the System Requirements Review.

Model the system

Models will be developed for most alternative designs. The model for the preferred alternative will be expanded and used to help manage the system throughout its entire life cycle. Many types of system models are used, such as physical analogs, analytic equations, state machines, block diagrams, functional flow diagrams, object-oriented models, computer simulations and mental models. Systems will usually be more successful if the models have observable states (Botta, Bahill and Bahill, 2006). Systems Engineering is responsible for creating a product and also a process for producing it. So, models should be constructed for both the product and the process. Process models allow us, for example, to study scheduling changes, create dynamic PERT charts and perform sensitivity analyses to show the effects of delaying or accelerating certain subprojects. Running the process models reveals bottlenecks and fragmented activities, reduces cost and exposes duplication of effort. Product models help explain the system. These models are also used in tradeoff studies and risk management.

Design the system

The overall system must be partitioned into subsystems, subsystems must be partitioned into assemblies, etc. Reusability should be considered in creating subsystems. For new designs, subsystems should be created so that they can be reused in future products. For redesign, subsystems should be created to maximize the use of existing, particularly commercially available, products. Systems engineers must also decide whether to make or buy the subsystems, first trying to use commercially available subsystems. If nothing satisfies all the requirements, then modification of an existing subsystem should be considered. If this proves unsatisfactory, then some subsystems will have to be designed in-house. Engineers designing one subsystem must understand the other subsystems that their system will interact with. Flexibility is more important than optimality. Hardware, software and bioware must be considered. Bioware (or wetware) means humans and other biological organisms that are a part of the system. For example, in designing a race track the horses or dogs are a part of the bioware. Facilities for their care and handling must be considered, as should provisions for education, human factors, and safety. These activities are called System Design for new systems and Systems Analysis for existing systems.

Create sequence diagrams

Arguably the first step in designing a system should be creating sequence diagrams. This means collecting typical sequences of events that the proposed system will go through. Sequences can be descriptions of historical behavior or can be based on mental models for future behavior. Such descriptions of input and output behaviors as functions of time are called sequence diagrams, behavioral scenarios, operational scenarios, operational concepts, operational sequences, threads, input and output trajectories, collaboration diagrams, logistics or interaction diagrams. Sequence diagrams are easy for people to describe and discuss, and it is easy to transform them into a system design. Additional sequences can be incrementally added to the collection. (Bahill and Dean, 1999; Bahill and Daniels, 2003).

Define system architecture

Defining the system architecture means choosing the high level switches that will determine the components and subsystems of the system. The following shows some choices that have to be made: (1) object-oriented design, structured analysis, or functional decomposition, (2) distributed or centralized computing, (3) commercial off the shelf (CoTS) or custom designed, and (4) Ada versus C++ or Java. Rechtin and Maier (1997) state that System Architecting is done in the first two phases of the system life cycle: discovering system requirements and concept exploration.

Functional decomposition

Systems engineers do functional decomposition on new systems (1) to map functions to physical components, thereby ensuring that each function has an acknowledged owner, (2) to map functions to system requirements, and (3) to ensure that all necessary tasks are listed and that no unnecessary tasks are requested. This list becomes the basis for the work breakdown structure.

When analyzing (or re-engineering) an existing system, Systems Engineers do functional analysis to see what the system does in order to improve its performance (often called value engineering), and they also do functional decomposition to see what the system is supposed to do. In this manner they can describe the present state of the system and the desired (or goal) state of the system. They can then suggest how the system design can be changed. Making radical dramatic changes in the system is called re-engineering. Making small incremental changes is called total quality management.

Icarus, and many flight wanna-bes after him, tried to understand how to fly by analyzing the physical components that birds used to fly: Legs, Eyes, Brain, and Wings. Using this paradigm man was not able to fly. The Wright brothers, in contrast, identified the following functions for the flight problem: Takeoff and land, Sense position and velocity, Navigate, Produce horizontal thrust, and Produce vertical lift. Once it was understood that thrust and lift were two functions, two physical components could be assigned to them. By using a propeller to produce thrust and wings to produce lift, manned flight was possible. The following Netscape specific table shows a mapping of functions to physical components.

Functional Versus Physical Decomposition

Function	Airplane Physical Component	Bird Physical Component
Takeoff and land	Wheels, skis or pontoons	Legs
Sense position and velocity	Vision or radar	Eyes
Navigate	Brain or computer	Brain
Produce horizontal thrust	Propeller or jet	Wings
Produce vertical lift	Wings	Wings

Birds use one physical component for two functions: thrust and lift. Man had to use two physical components for these two functions. As this example shows it is perfectly acceptable to assign two or more functions to one physical component. However, it probably would be a mistake to assign one function to two physical components.

Describing the functions that a system must perform is but one part of describing a system: the objects (Fowler and Scott, 2000; Jacobson, Booch and Rumbaugh, 1999; Douglas, 2000; Gomaa, 2000; Cockburn, 2001; Bahill and Daniels, 2003) and states must also be described (Bahill, et al., 1998; Wymore and Bahill, 2000).

Sensitivity analyses

In a sensitivity analysis each parameter or requirement is varied and the effects on the outputs are observed. Sensitivity analyses can be used to point out the requirements and parameters that have the biggest effects on cost, schedule and performance. They are used to help allocate resources. Karnavas, Sanchez and Bahill (1993).

Assess and manage risk

There are two types of risk: risk of project failure (due to cost overruns, time overruns or failure to meet performance specifications) and risk of harm (usually called personnel safety). A failure modes and effects analysis and risk mitigation must be performed (Bahill and Karnavas, 2000). Project risk can be reduced by supervising quality and timely delivery of purchased items. Kerzner (1995).

Reliability analysis

Major failure modes must be analyzed for probability of occurrence and severity of occurrence. Kapur and Lamberson (1977); O'Connor (1991).

Integrate system components

Integration means bringing things together so they work as a whole. System integration means bringing subsystems together to produce the desired result and ensure that the subsystems will interact to satisfy the customer's needs. End users and engineers need to be taught to use the system with courses, manuals and training on the prototypes. Grady (1994).

Design and manage interfaces

Interfaces between subsystems and interfaces between the main system and the external world must be designed. Subsystems should be defined along natural boundaries. When the same information travels back and forth among different subsystems a natural activity may have been fragmented. Subsystems should be defined to minimize the amount of information to be exchanged between the subsystems. Well-designed subsystems send finished products to other subsystems. Feedback loops around individual subsystems are easier to manage than feedback loops around interconnected subsystems. When designing subsystems and their interfaces be sure to consider reuse.

Launch the system

Launching the system means doing what the system was intended to do, e.g. running the system and producing outputs (Bahill and Gissing, 1998). Design engineers produce designs for the product and the process to make it.

Configuration management

Configuration management (also called modification management) ensures that any changes in requirements, design or implementation are controlled, carefully identified, and accurately recorded. All stakeholders should have an opportunity to comment on proposed changes. Decisions to adopt a change must be captured in a baseline database. This baseline is a time frozen design containing requirements for functions, performance, interfaces, verification, testing, cost, etc. Baselines can only be changed at specified points in the life cycle. All concerned parties must be notified of changes to ensure that they are all working on the same design. The phrase *requirements tracking* is now being used for an important subset of configuration management.

Project management

Project management is the planning, organizing, directing, and controlling of company resources to meet specific goals and objectives within time, within cost and at the desired performance level (Kerzner, 1995: Tichy and Sherman, 1993). Project management creates the work breakdown structure, which is a product-oriented tree of hardware, software, data, facilities and services. It displays and defines the products to be developed and relates the elements of work to be accomplished to each other and to the end product. It provides structure for guiding team assignments and cost and tracking control (Martin, 1997).

Documentation

All of these Systems Engineering activities must be documented in a common repository, often called the Engineering Notebook. The stored information should be location, platform, and display independent: which

means any person on any computer using any tool should be able to operate on the fundamental data. Assumptions, results of tradeoff studies and the reasons for making critical decisions should be recorded. These documents should be alive and growing. For example, at the end of the system life cycle there should be an accurate model of the existing system to help with retirement. Chapman, Bahill and Wymore (1992); Wymore (1993).

Lead teams

Complex systems cannot be designed by one person. Consequently engineers work on Integrated Product Development Teams (IPDTs). These teams are interdisciplinary with members from Business, Engineering, Manufacturing, Testing, etc. IPDTs are often led by Systems Engineers, either formally or informally. Katzenback and Smith, (1993).

Assess Performance

During the operation and maintenance phase of the system life cycle the performance of the system must be measured. Initially these measurements will be used to verify that the system is in compliance with its requirements. Later they will be used to detect deterioration and initiate maintenance.

Prescribe tests

Early in the system life cycle Systems Engineering should describe the tests that will be used to prove compliance of the final system with its requirements. However, most testing should be performed by built-in self-test equipment. These self-tests should be used for initial testing, post-installation testing, power-up diagnostics, field service and depot repair. The recipient of each test result and the action to be taken if the system passes or fails each test must be stated.

Conduct reviews

Systems Engineering should ensure that the appropriate reviews are conducted and documented. The exact reviews that are appropriate depends on the size, complexity and customer of the project. The following set is common: Mission Concept Review, System Requirements Review (SRR), System Definition Review, Preliminary Design Review (PDR), Critical Design Review (CDR), Production Readiness Review (PRR), and System Test. Full-scale engineering design begins after the Preliminary Design Review. Manufacturing begins after the Critical Design Review. (Shishko and Chamberlain, 1995; Bahill and Dean, 1999).

Total system test

The system that is finally built must be tested to see (1) that it satisfies the mandatory requirements, and (2) how well it satisfies the tradeoff requirements. In order to save money, a total system test was not done before the Hubble Space Telescope was launched. As a result we paid $850,000,000 to fix the system error.

Re-evaluation

Re-evaluation is arguably the most important task of Systems Engineering. For centuries engineers have used feedback to control systems and improve performance. It is one of the most fundamental engineering tools. Re-evaluation means observing outputs and using this information to modify the system inputs, the product or the process. Re-evaluation should be a continual process with many parallel loops. Everyone should continually re-evaluate the system looking for ways to improve quality. Tools used in this process include basic systems engineering, and the quality engineering techniques presented by, for example, Deming and Taguchi. Deming (1982); Bicknell and Bicknell (1994); Latzko and Saunders (1995).

Near the end of the project, engineers should write a Lessons Learned document. These lessons learned should not be edited by management, because management could trivialize what they do not understand or omit management mistakes. Communicating these lessons learned to the troops is a hard problem.

Categories of Systems Engineers

Many companies divide their Systems Engineers into three categories according to their major workflows: requirements definition, architectural design and testing and verification.

Creating Systems Engineers

The traditional method of creating Systems Engineers was to select well-organized engineers with lots of common sense and let them acquire 30 years of diverse engineering experience. But recently these traditional Systems Engineers have written books and standards that explain what they do and how they do it. So now that the tools, concepts and procedures have been formalized, in four years of undergraduate education we can teach Systems Engineers who will have performance levels 50% that of traditional Senior Systems Engineers. (These numbers are based on national average salary data. So you may disbelieve them if you wish.) Ten years of systems engineering experience will improve performance to 80% and another ten years will increase it to 100%.

Summary

Was Victor Frankenstein a good systems engineer? Yes. He built a complex system with many components, and his system worked! In addition, he gets high marks for low cost, high performance, on schedule and great reusability. However, he gets low marks for risk analysis, total life cycle analysis and final system test. The following discussion is based on Mary Shelley's 1818 novel, not on the movies.

Stating the problem: Frankenstein was depressed when his mother died. So he wanted to conquer death by infusing life into an inanimate body.

Understanding customer needs: I think he blew it here, compassion and grief consoling might better fit the needs of his customer.

Discovering system requirements: The only requirement he considered was creating life, so he did a poor job of discovering requirements.

Validating requirements: How could he validate them if he had no requirements?

Investigating alternatives: When the monster tracked down Frankenstein, the monster said that he was lonely and he wanted Frankenstein to build him a companion. In designing this companion Frankenstein considered a lot of alternatives.

Defining quantitative measures: He had no measures of effectiveness.

Modeling the system: He had mental models and sketches in his book, but no formal models. Consequently, Frankenstein was surprised at how hideous his creature turned out: thinking he was ugly [during construction]; but when those muscles and joints were rendered capable of motion, it became a thing such that even Dante could not have conceived.

Functional decomposition: He did a physical (not a functional) decomposition.

System design: He designed a very good system. And he gets high marks for reusability!

Sensitivity analyses: He did not know about sensitivity analyses. They are modern tools.

Risk management: He gets very poor marks for risk management. Indeed his monster ended up killing his brother, his best friend and his bride. But before Frankenstein finished his second monster, he considered the risks and destroyed his work.

Reliability analysis: His monster was very robust. It overcame lack of food, freezing cold and bullet wounds all by itself.

Integrating the system,: This was his crowning achievement, he put all of the parts together and it worked.

Launching the system: Frankenstein did not launch his monster into the world. In fact Frankenstein ran away when it first came to life.

Configuration management: He couldn't do configuration management if he had no requirements.

Project management: He did an excellent job of project management: his system had low cost, high performance and came in on schedule.

Documentation: He kept a laboratory notebook. This notebook is what enabled the monster to track Frankenstein down a year and a half later in a different country.

Leading teams: Frankenstein worked alone and never told anyone what he had done.

Assessing performance, which includes prescribing tests, conducting reviews, verifying requirements and total system test. Frankenstein did not do any of these tasks.

Re-evaluating and improving quality. The reason Frankenstein went to England was to consult with philosophers there, because he wanted his second monster to be better than the first. He never got around to describing the lessons that he learned while doing the project.

Systems Engineering is responsible for making sure that all these tasks are performed in a engineering environment. However, the Systems Engineering process must be tailored for each project. Often this means omitting certain tasks, which reduces cost but increases risk. If you choose to omit one of these tasks, you should ask yourself, Why?

References

ANSI/EIA 632, Standard, *Process for Engineering a System,* January 1999.

Bahill, A.T. and Briggs, C, The systems engineering started in the middle process: a consensus of system engineers and project managers, *Systems Engineering,* 4(2): 156-167, 2001.

Bahill, A. T., Botta, R., and Daniels, J., The Zachman framework populated with baseball models, *Journal of Enterprise Architecture,* **2**(4): 50-68, 2006.

Bahill, A.T. and Daniels, J, Using object-oriented and UML tools for hardware design: a case study, *Systems Engineering,* 6(1): 28-48, 2003.

Bahill, A.T. and Karnavas, W.J, Risk analysis of a Pinewood Derby: A case study, *Systems Engineering*, 3(3): 143-155, 2000.

Bahill, A.T., Alford, M., Bharathan, K., Clymer, J.R., Dean, D.L., Duke, J., Hill, G., LaBudde, E.V., Taipale, E.J. and Wymore, A.W., The design-methods comparison project, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **28**(1), 80-103, February 1998.

Bahill, A.T. and Dean, F.F., Discovering system requirements, in A.P. Sage and W.B. Rouse (Eds), *Handbook of Systems Engineering*, John Wiley & Sons, 175-220, 1999.

Bahill A.T. and Chapman, W.L., A tutorial on quality function deployment, *Engr Management J*, **5**(3):24-35, 1993.

Bahill, A.T. and Gissing, B., Re-evaluating systems engineering concepts using systems thinking, *IEEE Transactions on Systems, Man and Cybernetics*, Part C: Applications and Reviews, Volume 28, Number 4, pp. 516-527, November 1998.

Bicknell, K.D. and Bicknell, B.A., *The Road Map to Repeatable Success: Using QFD to Implement Changes*, CRC Press, Boca Raton, 1994.

Blanchard, B.S. and Fabrycky, W.J., *Systems Engineering and Analysis*, Prentice-Hall, 1998.

Botta, R., Bahill, Z. and Bahill, A. T., When are observable states necessary? *Systems Engineering*, **9**(3): 228-240, 2006.

Buede, D. M., *The Engineering Design of Systems*, John Wiley, New York, 2000.

Chapman, W.L., Rozenblit, J. and Bahill, A.T., Systems design is an NP-complete problem, *Systems Engineering*, 4(3): 222-229, 2001.

Chapman, W.L., Bahill, A.T. and Wymore, W., *Engineering Modeling and Design*, CRC Press, Boca Raton, 1992.

Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, 2001.

Daniels, J., Werner, P.W., and Bahill, A.T., Quantitative Methods for Tradeoff Analyses, *Systems Engineering*, 4(3), pp. 199-212, 2001.

Deming, W.E., *Out of the Crisis*, MIT Center for Advanced Engineering Study, Cambridge MA, 1982.

DSMC, *Systems Engineering Fundamentals*, Defense System Management College, 1999.

Douglas, B.P., *Real-Time UML*, Addison-Wesley, Reading, 2000.

Fowler, M. and Scott K., *UML Distilled: A brief guide to the standard object modeling language*, Addison-Wesley, 2000.

Grady, J.O., *System Requirements Analysis*, McGraw Hill Inc., 1993.

Grady, J.O., *System Integration*, CRC Press, Boca Raton, 1994.

Grady, J.O., *System Engineering Planning and Enterprise Identity*, CRC Press, Boca Raton, 1995.

Gomaa, H., *Designing Concurrent, Distributed, and Real-Time Applications with UML,* Addison-Wesley, Reading, 2000.

Hooks, I. F. and Farry, K. A., *Customer Centered Products: Creating Successful Products Through Smart Requirements Management,* American Management Association, 2001.

Hughes Aircraft Co., *Systems Engineering Handbook,* 1994.

IEEE 1220 Standard, *Application and Management of the Systems Engineering Process,* IEEE Standards Dept., NY, December 1998.

ISO/IEC 15288, *System Life Cycle Processes*, October 2002.

Jacobson, I., Ericsson, M. and Jacobson, A., *The Object Advantage: Business Process Reengineering with Object Technology,* Addison-Wesley, New York, 1995.

Jacobson, I. Booch, G. and Rumbaugh J., *The Unified Software Development Process*, Addison-Wesley, 1999.

Kapur, K.C. and L.R. Lamberson, *Reliability in Engineering Design*, John Wiley and Sons, New York, 1977.

Karnavas, W.J., Sanchez, P. and Bahill A.T., Sensitivity analyses of continuous and discrete systems in the time and frequency domains, *IEEE Trans Syst Man Cybernetics,* **SMC-23:**488-501, 1993.

Katzenbach, J.R. and Smith, D.K., *The Wisdom of Teams,* HarperCollins Publishers, 1993.

Kerzner, H., *Project Management: a Systems Approach to Planning, Scheduling, and Controlling,* Van Nostrand Reinhold, New York, 1995.

Latzko, W.J. and Saunders, D.M., *Four Days with Dr. Deming*, Addison-Wesley, Reading, Mass, 1995.

Martin, J., *Systems Engineering Guidebook: A Process for Developing Systems and Products,* CRC Press, Boca Raton, 1997.

Martin-Marietta, *Systems Engineering Methodology Handbook* EPI 270-01, 1994.

MIL-STD-499B, Draft Military Standard for Systems Engineering, AFSC/EN, 1992. This standard was not signed by the Department of Defense. Spokesmen said that the government should not be in the business of writing standards and therefore they would adopt standards written by professional societies.

Moody, J.A., Chapman, W.L., Van Voorhees, F.D. and Bahill, A.T., *Metrics and Case Studies for Evaluating Engineering Designs,* Prentice Hall PTR, Upper Saddle River, NJ, 1997.

O'Connor, P.D.T., *Practical Reliability Engineering*, 3rd Edition, John Wiley and Sons, 1991.

Rechtin, E., *Systems Architecting of Organizations: Why Eagles Can't Swim*, CRC Press, Boca Raton, 2000.

Rechtin, E. and Maier, M.W., *The Art of Systems Architecting*, CRC Press, Boca Raton, 1997.

Rumbaugh J., Jacobson, I. and Booch, G., *The Unified Modeling Language Reference Manual,* Addison-Wesley, 1999.

Sage, A.P., *Systems Engineering,* John Wiley and Sons Inc., 1992.

Sage, A.P. and Rouse, W.B. (Eds), *Handbook of Systems Engineering,* John Wiley & Sons, 1999.

Shelley, M.W., *Frankenstein,* Barnes and Nobel Books, New York, 1993.

Shishko, R. and Chamberlain, R.G., *NASA Systems Engineering Handbook,* SP-6105, 1995.

Smith, E. D., Son, Y. J., Piattelli-Palmarini, M. and Bahill, A. T., Ameliorating Mental Mistakes in Tradeoff Studies, *Systems Engineering,* 10(3): 222-240, 2007.

Szidarovszky, F., Gershon, M. and Duckstein, L., *Techniques for Multiobjective Decision Making in Systems Management,* Elsevier Science Publishers, Amsterdam, 1986.

Tichy, M. and Sherman, S. *Control Your Destiny or Someone Else Will,* Currency Doubleday, New York, 1993.

Wymore, W., *Model-Based Systems Engineering,* CRC Press, Boca Raton, 1993.

Wymore, A.W., and Bahill, A.T., When can we safely reuse systems, upgrade systems, or use COTS components? *Systems Engineering,* 3(2): 82-95, 2000.

A Portuguese language (Brazilian) translation of this paper is available at http://br.geocities.com/rcapetti/Engenharia_sistemas.htm

Click here to go to Bahill's main Systems Engineering page.