System Design Is an NP-Complete Problem

William L. Chapman,[1] Jerzy Rozenblit,[2] and A. Terry Bahill[3, *]

[1]*Raytheon Missile Systems, Tucson,* AZ 85739

[2]*Electrical and Computer Engineering, University of Arizona, Tucson,* AZ 85721

[3]*Systems and Industrial Engineering, University of Arizona,* 1130 E. *North Campus Drive, Tucson,* AZ 85721-0020

Received July 15, 2000; accepted March 18, 2001

ABSTRACT

The system design process translates the customers' needs into a buildable system design. It requires selecting subsystems from an allowable set and matching the interfaces between them. Designs that meet the top-level input and output requirements are tested to see how well they meet the system's performance and cost goals. This paper proves that the System Design Problem is NP-complete by reduction from the Knapsack Problem, which is known to be NP-complete. The implication of this proof is that designing optimal systems with deterministic, polynomial time procedures is not possible. This is the primary reason why engineers do not try to produce optimal systems: They merely produce designs that are good enough. © 2001 John Wiley & Sons, Inc. Syst Eng 4: 222–229, 2001

1. INTRODUCTION

The System Design Process shown in Figure 1 translates the system requirements into a buildable system design. This process can be described mathematically [Wymore, 1993] and therefore we can estimate its complexity, which will help us identify the best solution approach.

This is a theoretical systems engineering paper. Therefore, it is appropriate to link it to other theoretical papers. In the early part of the 20th century, mathematicians expected to construct a network of postulates, axioms, and interlinked theorems that would capture all true statements of mathematics. In 1931, Gödel [1990] proved that this was not possible, by proposing his Incompleteness Theorem, which is similar to "This sentence is false." Gödel was concerned with the whole field of axiomatic mathematical reasoning. In a narrower realm, later logicians tried to prove whether solutions existed for certain classes of problems. Turing [1936] showed that all solvable sequential logic problems could be solved with a Turing Machine (a state machine with a memory). Later he showed that the Halting Problem (determining whether a Turing machine will halt for a given input and set of rules) is undecidable, i.e., unprovable. Turing was concerned with the solvability of logic problems. Our present

* Author to whom all correspondence should be addressed (e-mail: terry@sie.arzona.edu).

Systems Engineernig, Vol. 4, No. 3, 2001
© 2001 John Wiley & Sons, Inc.

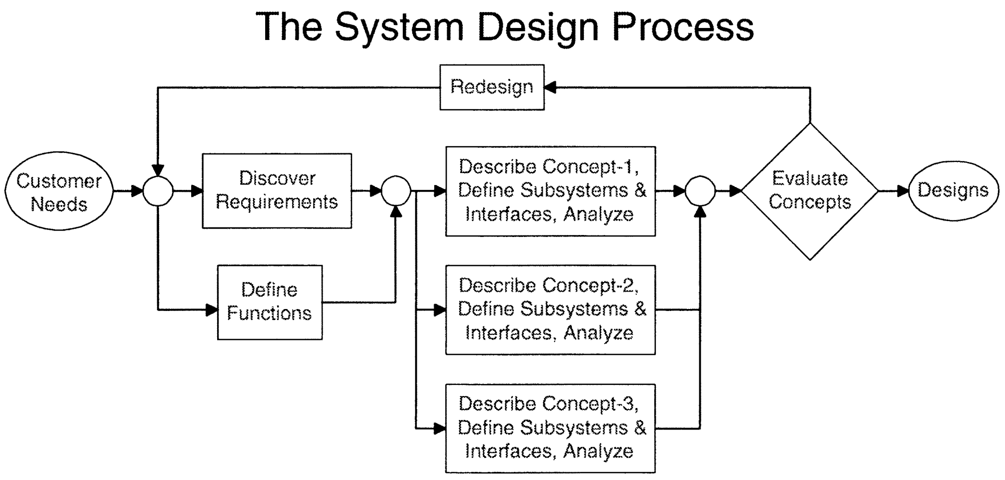

The System Design Process
Redesign
Customer Needs
Discover Requirements
Define Functions
Describe Concept-1, Define Subsystems & Interfaces, Analyze
Describe Concept-2, Define Subsystems & Interfaces, Analyze
Describe Concept-3, Define Subsystems & Interfaces, Analyze
Evaluate Concepts
Designs

Figure 1. The system design process customized for the preliminary design phase. Only three alternative concepts are shown in this figure: The best number of alternatives to be considered would, of course, depend upon the particular problem.

paper is concerned with a subset of solvable problems, namely, system design problems. In particular, it concerns the length of time required to solve these problems. The problems that take the most time are called NP-complete.

2. NP-COMPLETE PROBLEMS

NP-complete is the name of a class of problems for which there is no known efficient deterministic algorithmic solution [Garey and Johnson, 1979]. All known algorithms for solving these problems have the property that as the problem size increases, the number of steps necessary to solve the problem increases exponentially.

First, let us look at an easy problem that has efficient algorithms whose number of steps increases at the rate of a polynomial. For example, a simple sort of a list of numbers can be done in n^2 number of operations. Thus, if there were 10 numbers to sort, it would take, at most, 10^2 or 100 operations to perform the sort. If there were 100 numbers to sort, then it would take 100^2 or 10,000 operations. This is an easy problem.

Now let us look at a hard problem that takes an exponential number of steps to solve, because the size of the problem is in the exponent, such as e^n. For an exponential problem with n=10 there would be 22,026 operations. If n=100, there would be 2.7×10^{43} operations. As shown in Figure 2, the number of operations quickly exceeds the capability of any machine to compute a solution. For example, if there were a machine that could do 10^{12} operations per second (none yet exist) and there was a problem that required 10^{20} operations, it would take 10^{20-12} or 10^8 s or more than 3 years to solve. If the problem required 10^{23} operations, then it

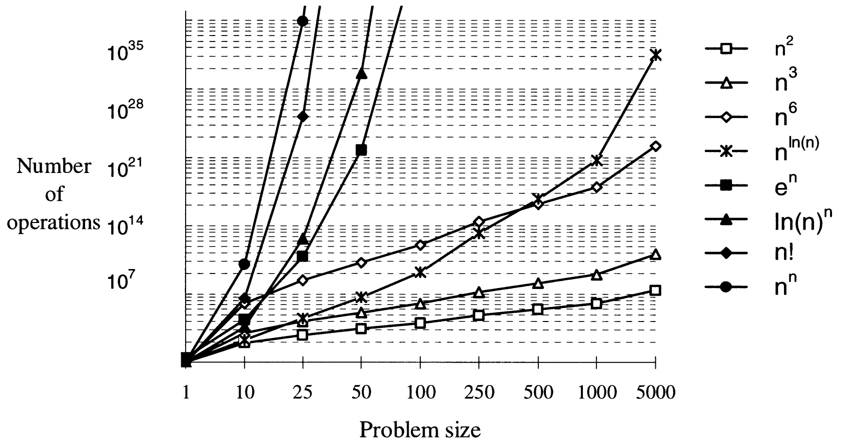


Figure 2. As the problem size n increases the number of operations needed to compute an optimal solution increases. The ordinate is logarithmic.

would take 3171 years to solve. Assume the age of the universe is 9 billion years which is 3×10^{17} s, then any problem that requires more than 10^{12+17} or 10^{29} operations could not be done in the entire age of the universe!

Another example of this explosive growth of exponential functions is the fable of the king, the peasant, and the checkerboard. The peasant did a favor for the king, and, in return, the king asked the peasant to name his reward. The humble peasant said he merely wanted a single grain of rice on the first square of a checkerboard and twice as many on each succeeding square, 2^n. The king agrees. The first few squares take very little rice. The 20th square takes a few gallons. The 25th square takes the volume of a desk. The 30th square needs the volume of a lecture room. The 64th square requires 2^{63} grains of rice, which would fill a million large cargo ships.

The set of problems solvable in Nondeterministic Polynomial time is called NP. Logicians created a conceptual device called a nondeterministic machine to solve these problems. A nondeterministic machine has an infinite number of processors and two stages: a guessing stage and a checking stage. Each processor guesses an answer and the checker verifies that it is a good answer. Both stages run in polynomial time (polynomial in the size of the input, e.g., n^6). Because an infinite number of processors exist and all guessing and checking is done in parallel, the computation time is polynomial. Of course, this is a fantasy machine, but it helps to illustrate the fact that a certain set of problems can be solved in polynomial time if one of these nondeterministic machines is used. Clearly, any deterministic polynomial algorithm could still use the nondeterministic machine, because the algorithm could be restricted to one processor.

NP-complete is a class of problems that can be solved in polynomial time on a nondeterministic machine, but for which no deterministic polynomial time algorithm is known. It has never been proven that a polynomial algorithm does not exist—but no one has ever found one, and mathematicians think no one ever will. One critical feature of all NP-complete problems is that an instance of one can be mapped into an instance of another by using a polynomial time transformation. If even one of the problems in the class NP-complete can be shown to have a solution in polynomial time, then all of them have such a solution. Yet, none has been found in 30 years of searching by very talented people. Thus, it is generally agreed that if a problem is shown to be NP-complete, then no efficient algorithm for solving the problem will ever be found. To prove that a problem is NP-complete, we must show that it is in the intersection of NP problems and NP-hard problems. But this statement does not clarify; it just adds more complexity, so let us look at an example.

To illustrate NP-completeness, we will now discuss the Knapsack Problem, which Karp [1972] proved is NP-complete. In the Knapsack Problem, a hiker is to pack a backpack. He has a large collection of items to choose from (U). Each item (u) to be put in the backpack will have a certain value [$v(u)$] on the trip and each item has a certain size [$s(u)$]. The hiker's task is to choose items so that at least the total value of (K) is obtained, without exceeding the size of the backpack (B). It is formally stated as

Instance: A finite set U, a "size" $s(u) \in Z^+$ (where Z^+ is the set of positive integers) and a "value" $v(u) \in Z^+$ for each $u \in U$, a size constraint $B \in Z^+$, and a value goal $K \in Z^+$.

Question: Is there a subset $U' \subseteq U$ such that

$$\sum_{u \in U'} s(u) \le B \quad \text{and} \quad \sum_{u \in U'} v(u) \ge K\ ?$$

3. THE SYSTEM DESIGN PROBLEM

In terms of systems theory, the system design problem can be described as stating the input–output relationships, the design constraints, and the performance and cost figures of merit [Asimow, 1962; Chapman, Bahill, and Wymore, 1992; Wymore, 1993]. For a given set of subsystems (or components) available to build the system, a possible system is configured that satisfies the system's input-output requirements. This system is then tested using some predefined test requirement to provide an overall system performance index (*PI*). This must exceed a customer provided acceptability limit. The cost of the system in terms of time, money, or other resources is then computed into an overall cost index (*CI*). This *CI* must be less than some customer specified target value.

The systems approach to design can be characterized as follows: Define a set of subsystems (Z) that constitute the available technology to build the desired system. Each element Zi in this set has an input port Ii, and an output port Oi. The ports provide the means of connecting the different subsystems together to form a system. The output ports are connected to input ports according to a system coupling recipe, SCR, to form a candidate system, $Z@$. Define the overall input to the desired system as $I0$ and the overall output of the desired system as $O0$. In this paper, we only consider the aspect of system design related to connecting subsystems together and evaluating performance. If this task is shown to be NP-complete, then certainly the larger task that

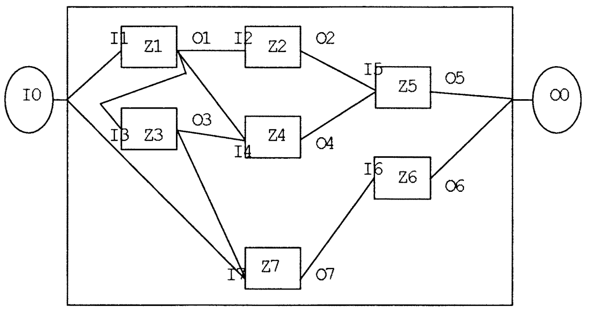


Figure 3. Potential connectivities for seven subsystems.

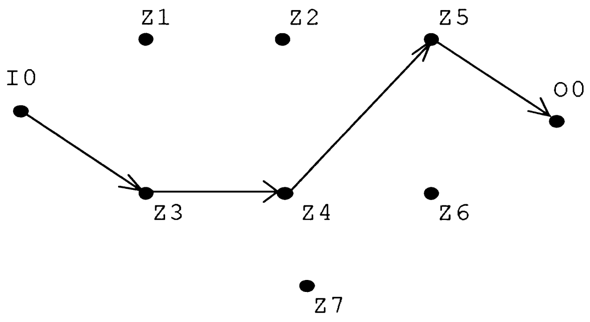


Figure 5. One possible route through the directed graph of Figure 4.

includes discovering requirements, defining system functions, and redesigning will also be NP-complete. A system, $Z@$, that could be built from seven subsystems is shown in Figure 3.

These connections can be expressed as a directed graph where the individual subsystems are nodes and the possible connections between ports are the arcs. The initial source for the directed graph is the system input port, $I0$, and the initial target (or sink) for the directed graph is the output port, $O0$. Let the length of each arc represent the cost of connecting the two subsystems (see Fig. 4). To find a potential system design, we must find a path through a directed graph.

Finding a path through a directed graph can be accomplished in polynomial time. Finding the shortest path through the graph can also be accomplished in polynomial time. The problem is that system design is not simply solving for one constraint such as the least cost. In addition, a system must be found that at least matches a threshold of performance. Having two constraints to satisfy at the same time requires tradeoffs as a search for the best value is done. This requires much more searching especially as the number of options increases.

A system is often composed of subsystems that are connected together. A System Coupling Recipe (SCR) specifies the subsystems to be connected (VSCR) and their connectivities (CSRC) [Wymore, 1993]. For the

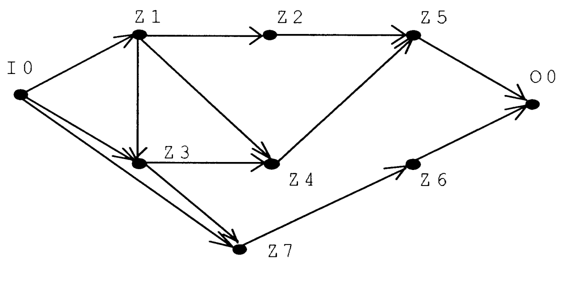


Figure 4. A directed graph of the connectivities shown in Figure 3.

route in Figure 5, the system $Z@$ is the result of the following system-coupling recipe:

$$\text{SCR}=\{\text{VSCR} = (Z3, Z4, Z5)$$

and

$$\text{CSCR} = ((I0, IZ3), (OZ3, IZ4), (OZ4, IZ5), (OZ5, O0))\}$$

The VSCR tells us that the system is comprised of three subsystems: $Z3$, $Z4$, and $Z5$. The CSCR tells us that the system input, $I0$, is connected to the input of $Z3$. The output of $Z3$ is connected to the input of $Z4$. The output of $Z4$ is connected to the input of $Z5$. Finally, the output of $Z5$ is connected to the system output, $O0$. After the system is coupled, it is tested per the requirements to see if it satisfies the *PI*. For a simple path, the *CI* of the system can be the sum of the length of the arcs.

This process is how designs are created. An engineer finds subsystems that satisfy the necessary input/output requirements, and creates an interconnection of these subsystems to satisfy the performance and cost requirements. Several different systems (concepts, alternatives, models, or prototypes) are often considered before a selection of the best possible system is determined based on tradeoff studies.

4. THE SYSTEM DESIGN PROBLEM IS NP-COMPLETE

Let us now compare this System Design Problem to the Knapsack Problem. The individual subsystems are connected together as specified by the system-coupling recipe to form the overall system called $Z@$, which has associated performance (*PI*) and cost (*CI*) measures. Let $PI \Leftrightarrow K$ and $CI \Leftrightarrow B$. Let $Z \Leftrightarrow U$, be the set of all allowable subsystems. Let $Z@ \Leftrightarrow U'$ be the subset of subsystems selected from Z by means of the SCR to form $Z@$. Each subsystem $Zi \Leftrightarrow u$ has an associated cost that contributes to the *CI*. Let cost $(Zi) \Leftrightarrow s(u)$. Each subsystem $Zi \Leftrightarrow u$ has an associated value that can be measured by the test requirement that contributes to the overall performance index, *PI*. Let perf(*Z*) $\Leftrightarrow v(u)$. We summed the performance of each item to get the overall performance index. More complicated tradeoff functions are sometimes used in the real world. This is not bothersome, because if our simple problem is NP-complete, then any design using a more complex tradeoff function would have to be at least as difficult. If we restrict the System Design Problem to performance and cost measures that combine linearly, then

$$\sum_{Zi \in Z@} \text{pref}(Zi) = PI \quad \text{and} \quad \sum_{Zi \in Z@} \text{cost}(Zi) = CI.$$

Therefore, the Knapsack Problem maps to an instance of the System Design Problem, as is shown in Table I.

Therefore, if we can find a system *Z@* such that its *PI* and *CI* satisfy the customers' requirements, then we have solved the System Design Problem. Hence, if we can solve the System Design Problem, then we can solve the Knapsack Problem, but we know that the Knapsack Problem is NP-complete; therefore, the System Design Problem is also NP-complete.

Showing that the Knapsack Problem can be restricted to an instance of the System Design Problem is sufficient for proving NP-completeness, because if a solution for the System Design Problem were available we could use it to solve the Knapsack Problem, and hence all other NP-complete problems.

5. IMPLICATIONS

The first implication of the System Design Problem being NP-complete is that humans cannot design optimal systems for complex problems. And computers will not be able to bail us out, because computers cannot design optimal solutions for complex problems either. Therefore, the purpose of creating computerized design tools should not be to allow us to design optimal systems. Furthermore, it is unlikely that a computer can design a complex system better than a human can. Subsets of the entire design process, such as routing and checking interfaces, are done better by computers now; however, no computer algorithm exists to create even a simple automobile factory or personal computer. The creation of a system is as much art as it is science, because the combinatorics involved requires original solutions, rather than fixed algorithms for solving the problem. When the more complex issues of individual creativity and adjusting for perceived customer wants, rather than those that are accurately expressed, are considered, it becomes even more obvious that a totally automated design tool is impossible. Research must focus on the human in the loop as the only feasible approach to the System Design Problem.

If it is so difficult to obtain optimality then one might ask, "Why are there so many good systems?" The answer lies within the solution techniques of NP-complete problems. Even the most difficult problems in this class have algorithms to obtain good solutions (that is, a solution within a few percent of a theoretical optimal when it is possible to compute) with relatively simple polynomial algorithms.

The techniques used to find good solutions to NP-complete problems [Garey and Johnson, 1979] (such as the Traveling Salesman Problem [Coy et al., 1998], the Knapsack Problem [Karp, 1972], the maximum path through a network [Bernard and Graham, 1989], the minimum test collection, graph 3-colorability, etc.) can also be applied (and have been applied, knowingly or not) to the System Design Problem. We have shown that many methods of solving NP-complete problems have been used to design systems [Moody et al., 1997]. One of these techniques is decomposition. The overall system is decomposed into subsystems as shown in Figure

Table I. Mapping the Knapsack Problem to the System Design Problem

The Knapsack Problem		**The System Design Problem**	
set of all possible items	*U*	allowed technology	*Z*
an individual item	*u*	an individual subsystem	*Zi*
size of individual item	*s*(*u*)	cost of individual subsystem	cost(*Zi*)
value of individual item	*v*(*u*)	individual subsystem performance	perf(*Zi*)
selected subset of items	*U′*	system that is the result of an SCR	*Z@*
size constraint	*B*	allowable cost of system	*CI*
value goal	*K*	required system performance	*PI*
Question: Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \le B$ and $\sum_{u \in U'} v(u) \ge K$?		Question: Is there an SCR such that $\sum_{Zi \in Z@} \text{cost}(Zi) \le CI$ and $\sum_{Zi \in Z@} \text{perf}(Zi) \ge PI$?	

3. Then each of these subsystems is decomposed into subsubsystems. This process is continued until each subsubsystem is small enough that a team of engineers can design it.

One problem with the decomposition technique is that common folk wisdom says that connecting optimal subsystems will not produce an optimal supersystem (e.g., in football a Pro Bowl team would not be expected to beat the Superbowl champions). However, Wymore (http://www.sie.arizona.edu/sysengr/wymore/optimal .html) shows that for a very restricted set of tradeoff functions, connecting optimized subsystems does indeed yield an optimal supersystem.

Most engineers realize that most of they time they do not try to produce optimal designs. In this paper we have shown why this is a good strategy. Engineers do not try to produce optimal designs, because for complex systems it would be impossible to do so. Simon [1957, pp. 204–05] says that the key to successful design is "the replacement of the goal of *maximization* with the goal of *satisficing*, of finding a course of action that is 'good enough.' ... Since the [designer] ... has neither the senses nor the wits to discover an 'optimal' path—even assuming the concept of optimal to be clearly defined—we are concerned only with finding a choice mechanism that will lead it to pursue a 'satisficing' path, a path that will permit satisfaction at some specified level of all its needs."

Ostrofsky [1977, p. 79] defined the optimum system to be "theoretically the most favorable for the criteria defined" and the optimal system to be "the most favorable for the criteria and the set of candidates defined." His point was that you never get the optimum solution, and you must settle for less.

As two reviewers of this paper stated, "Most systems engineers probably understand the fact that an optimal solution to a complex system design problem does not exist!" All we have done in this paper is prove mathematically that their intuitions, generated by years of experience, are indeed correct.

The real-world System Design Problem is harder than the System Design Problem described in this paper. First, in this paper, the requirements never changed; in the real world, this is seldom true. Second, in this paper, we have shown that designing *one* system is NP-complete. But do design engineers only design one system? No. They design many systems and then try to choose the best alternative for implementation. Therefore, the real-world System Design Problem is harder than what we have described in this paper. However, Klein [1998] says that designers really do not evaluate many alternative designs in parallel, but rather they develop one alternative at a time until they get one that satisfies. Therefore, the complexity of the real-world System Design Problem lies somewhere in between designing one system and designing many and selecting the best alternative. Therefore, if designing one system is NP-complete, then the real-world System Design Problem is even harder.

In this paper, we have shown that the theoretical System Design Problem is NP-complete in all aspects discussed, and we have shown that the real-world System Design Problem is harder. However, there is one exception. The performance of a system could be greater than the sum of its subsystems (cooperation). Two lions chasing a Thompkins Gazelle are more than twice as likely to catch it, than a single lion. A pair of chopsticks performs more than twice as well as an individual chopstick. However, we used linear addition for our performance figures of merit. So, if the whole is greater than the sum of its parts, we could get better performance.

6. SUMMARY

There is a need for theory in the field of Systems Design. This paper is but one baby step in that direction. Because system design is such an old field, it would be expected to have a theoretical basis. However, there are very few theoretical systems engineering papers. The notable exceptions are Wymore [1993], Wymore and Bahill [2000], and the World Wide Web papers by Wymore (http://www.sie.arizona.edu/sysengr/wymore). Although it is only a small step, this paper adds to the theoretical papers in the field.

This paper has proven that the System Design Problem is NP-complete by mapping it to the Knapsack Problem. The implications are that achieving an optimal design for a complex system is not likely. It is possible to design forever without achieving an optimal solution. Therefore, limits must be set on the design early in the process. We do not have the time to design optimal systems, but we can design systems that are good enough.

In addition, creating a computer design tool will be very difficult. The interesting aspect of NP-complete algorithms is that it is often quite easy to find near-optimal solutions. Within the context of product design, optimality is seldom an objective, but rather satisfaction of a problem statement. Therefore, a solution good enough to satisfy the customer may be within reach of a knowledge-based design tool.

In summary, philosophers have pondered the type of problems that are solvable: The System Design Problem seems to be solvable. Theoreticians have worried about how long it might take to find solutions for certain classes of problems: Because the System Design Problem is NP-complete, finding an optimal solution could take an infinite amount of time. Therefore, systems engineers should ensure that their customers do not require optimum solutions, because optimum solutions are not feasible. Behaviorists have shown that humans do not seek optimal solutions, instead they seek satisficing solutions: therefore, systems engineers must explain to their customers that we only produce good enough solutions for the System Design Problem, that is, we produce satisficing solutions, not optimal solutions. Finally, academicians have proven that the job of a systems engineer is hard, in fact, NP-hard.

ACKNOWLEDGMENTS

We thank Will Evans at the University of Arizona for thoughtful comments. An earlier version of this paper was published in the *Proceedings of the IEEE International Conference on Systems Man and Cybernetics*, San Antonio, TX, October 1994, pp. 1880–1884.

REFERENCES

M. Asimow, Introduction to design, Prentice-Hall, Englewood Cliffs, 1962.

M.W. Bern and R.L. Graham, The shortest network problem, Sci Am (January 1989), Vol. 260, no. 1, 84–89.

W.L. Chapman, A.T. Bahill, and A.W. Wymore, Engineering modeling and design, CRC Press, Boca Raton, FL, 1992.

S.P. Coy, B.L. Golden, G.C. Runger, and E.A. Wasil, See the forest before the trees: Fine-tuned learning and its application to the traveling salesman problem, IEEE Trans Syst Man Cybernet Part A Syst Hum 28 (1998), 454–464.

M. Garey and D. Johnson, Computers and intractability: A guide to the theory of NP-completeness, Freeman, New York, 1979.

K. Gödel, Kurt Gödel, Collected works, S. Feferman (Editor), Oxford University Press, Oxford, 1990.

R.M. Karp, "Reducibility among combinatorial problems," Complexity of computer computations, R.E. Miller and J.W. Thatcher (Editors), Plenum, New York, 1972, pp. 85–103.

G. Klein, Sources of power: How people make decisions, MIT Press, Cambridge, MA, 1998.

J.A. Moody, W.L. Chapman, F.D. Van Voorhees, and A.T. Bahill, Metrics and case studies for evaluating engineering designs, Prentice Hall PTR, Upper Saddle River, NJ, 1997.

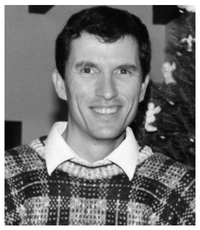
B. Ostrofsky, Design, planning, and development methodology, Prentice-Hall, Englewood Cliffs, NJ, 1977.

H. A. Simon, Models of man: Social and rational, Wiley, New York, 1957.

A. Turing, On computable numbers with an application to the Entscheidungsproblem, Proc London Math Soc, XLII, 1936, pp. 239–265 ["Correction," Proc London Math Soc, XLIII, 1937, pp. 544–546].

A.W. Wymore, Model-based systems engineering, CRC Press, Boca Raton, FL, 1993.

A.W. Wymore and A.T. Bahill, When can we safely reuse systems, upgrade systems or use COTS components? Syst Eng 3(2) (2000), pp. 82–95.



William L. Chapman is an Engineering Fellow at Raytheon Corporation in Tucson. He has worked on many parts of tactical missile design and is currently responsible for the research tasks of the Land Combat Product Line. He earned a Ph.D. in Systems Engineering from the University of Arizona in 1994.

Jerzy Rozenblit is Professor of Electrical and Computer Engineering at the University of Arizona, Tucson. He holds the Ph.D. and M.S. degrees in Computer Science from Wayne State University, Michigan and the MSc in Computer Engineering from the Technical University of Wroclaw, Poland. His research and teaching are in the areas of complex systems design and simulation modeling. His research in design has been supported by the National Science Foundation, Siemens AG, Semiconductor Research Corporation, McDonnell Douglas, and the U.S. Army Research Laboratories, where he was a Research Fellow. Dr. Rozenblit serves as Associate Editor of ACM Transactions on Modeling and Computer Simulation, Associate Editor of IEEE Transactions on Systems, Man and Cybernetics, and a reviewer for a number of national and international funding agencies. In 1994 and 1995, he was Fulbright Senior Scholar and Visiting Professor at the Institute of Systems Science, Johannes Kepler University, Austria. He has also held visiting scientist appointments at the Central Research Laboratories of Siemens AG, in Munich.

A. Terry Bahill is a Professor of Systems Engineering at the University of Arizona and an Engineering Fellow with Raytheon Missile Systems in Tucson. He received his Ph.D. in electrical engineering and computer science from the University of California, Berkeley, in 1975. He holds a U.S. patent for the Bat Chooser™, a system that computes the Ideal Bat Weight™ for individual baseball and softball batters. He is Editor of the CRC Press Series on Systems Engineering. He is a Fellow of the Institute of Electrical and Electronic Engineers (IEEE) and of the INCOSE. He is the chair of the INCOSE Fellows Selection Committee.