Improving the System Software Requirements Development Process

Phillip J. Brown, Alexander E. Iwach, Donald R. Williams
Loral Vought Systems Corporation
P.O. Box 650003, M/S EM-90
Dallas, TX 75265-0003

Abstract. One of the most significant challenges currently facing the system engineering profession is devising procedures for improving the system software requirements development process. While many practitioners promote a variety of automated tools and mechanistic templates as the means to improved productivity, experience suggests the highest leverage lies in harnessing the *cognitive* processes required to produce a stable set of well defined system software requirements. System complexity and schedule constraints necessitate the use of teams of specialists working together to produce the desired software requirements database. System engineering's primary responsibilities are to foster team acceptance of a shared vision through the identification of intermediate and final products supporting the software requirements development process. This paper describes a framework for accomplishing the above, illustrates key points with actual examples, and identifies three approaches for improving management understanding of the requirements and software development processes.

INTRODUCTION

The continuing explosive growth in processor throughput capability and available memory per dollar of expenditure is turning our view of system engineering on its head (Babbitt 1993 and Rechtin 1993). In the past, and to a lesser extent today, systems were engineered based on the level of performance that could be wrung out of hardware configurations. Software was developed around the chosen hardware baseline and made to fit. However, exponential growth in computer hardware capability invites new demands for improved software performance, better reliability and previously undreamed of applications. With these new demands come increasing software complexity and an ever larger proportion of system development resources being allocated to system software development. Examples of industry software development problems abound; the General Accounting Office (Bridickas 1992) observed that “Defense's mission-critical systems continue to have significant software development problems.” The bulk of these problems are attributed to inadequate management, defective requirements definition and flawed testing approaches.

We believe many of the problems cited by the General Accounting Office can be avoided by more attention to the development of system software from specialists within a multi-disciplined team construct. Lacking such a construct, the software engineering community, bereft of a real system engineering commitment, has forged ahead developing CASE tools and procedures designed to ease their software development burden. The resulting outpouring of formal processes, metrics and automated tools has tended to mask the need for improving the efficiency of intellectual activities.

The importance of marshaling human thought in developing software is only now beginning to be studied. Recent research at Penn State (Hogg 1993) indicates time allocated to thought or intellectual (non-routine) software development tasks outweighs time allocated to clerical (routine) tasks by a margin of four to one. According to Hogg, intellectual tasks include such chores as constructing models, generating usable code, and analyzing data flow. Among the clerical tasks are redrawing diagrams, identifying rule violations, and maintaining a record of design changes.

Add the diversity of viewpoints introduced by individual problem solving techniques to a multi-person software development project, and one has the ingredients for a significant expenditure of resources. Conversely, focusing everyone's efforts through a common process, thus gaining high leverage from a productivity improvement standpoint, has been shown to reduce the total amount of time expended on thought (Cusumano 1991). This is achieved by effectively sharing, combining and aligning the specialized information each member of a system development

team brings to solving software development problems. Yourdan asserts that attention to "peopleware" can produce 10-fold productivity improvements (Yourdan 1993). Our belief, based on our experience, is consonant with Yourdan's assertions.

The use of a team approach for solving system problems is not a new idea (Churchman, Ackoff, and Arnoff, 1957). In fact, the use of multi-disciplined technical teams was successfully implemented by the military to solve strategic and tactical problems in World War II. What is new is widespread formation of teams by corporations combined with the growing recognition that the use of teams in the United States has not lived up to its promise. Two recent books by MIT's Sloan School of Management faculty contend the root cause of our inability to fully realize the benefits of teams is the emphasis in society on individualistic capitalism (Thurow, 1993) which produces traditional authoritarian "controlling organizations" (Senge, 1990). As Thurow notes, "if the system is based solely upon individual effort, there is no need to pay attention to group motivation, voluntary cooperation, or teamwork."

The solution, according to Senge, is an organizational commitment to collective learning. The elements of such a commitment are encouraging personal learning, surfacing and challenging mental models, building shared visions, fostering team learning and, most importantly, promulgating systems thinking across the breadth of the organization/project. In terms of improving the systems software development process, this translates to an early and continuing inter-disciplinary team effort focused on aligning and developing the capacity to produce the required software products.

Fundamental to building the system/software requirements development team is an agreed-upon definition of the responsibilities of the system and software organizations. One such definition was presented at last year's National Council on Systems Engineering symposium (Brown and Cady 1993). Moreover, a common framework for focusing all affected disciplines on the system must be provided. Other affected disciplines typically include mechanical designers, electronics engineers, test personnel, and representatives from the quality organization. A frequently used medium for the team's common reference frame is a specification tree.

In devising a process for the system/software team to follow in developing system software requirements for appropriate tree branches, a critical test the process must pass is that it works for both hardware and software requirement definition. Too often one sees processes developed from either a strictly software or strictly hardware perspective. Equitable allocation of requirements between hardware and software elements mandates use of an unbiased requirements development process.

SYSTEM SOFTWARE REQUIREMENTS DEVELOPMENT PROCESS

Development of system software requirements traces from original system specifications and follows the flow-down of requirements to subordinate elements. An example of a system decomposition which follows

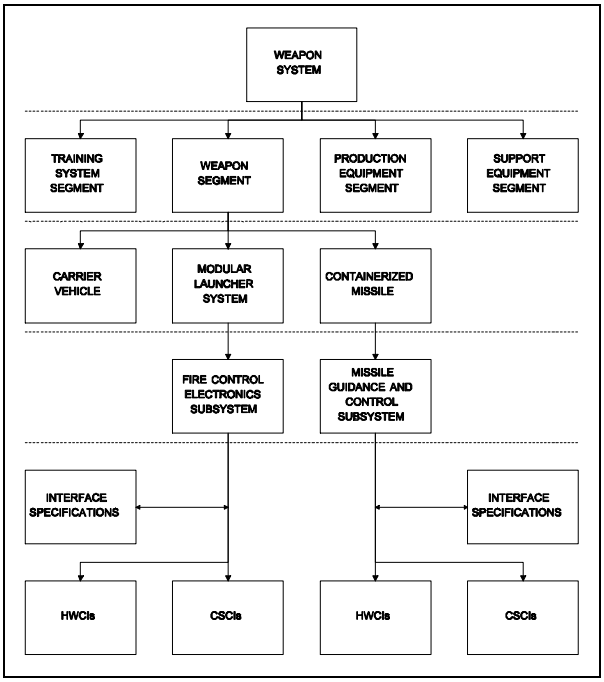

Figure 1. System Specification Tree Promotes Team Communication

the work breakdown structure (WBS) and corresponding specification tree is shown in Figure 1. For the sake of illustration, the complexity of the system requires only four levels of decomposition to arrive at a point that designers may begin the process of developing components. The process described here works for any level in the hierarchy.

The objective of the requirements development process is to amplify top-level requirements and develop an underlying design of methods and algorithms for implementation in software. For example, a weapon system which includes missile guidance components will have guidance laws

implemented in software. Guidance and/or control system engineers develop specific methods and algorithms by building models using FORTRAN and other computer-based tools. Specific software design issues are considered during this phase of the development and impacts are assessed by the requirements development team, which includes a software engineering representative. In the past, software requirements definition started at the lowest level of system definition and decomposition and, in many cases, resulting source code was turned over to software engineers to implement the algorithms in embedded software.

As the complexity of systems increased and new processes for software development were implemented, it became clear that software requirements needed to be surfaced much earlier in the system development process. Furthermore, systems have become more oriented toward software implementations and less oriented toward hardware implementations. The system engineering needed to partition requirements between hardware and software elements is evolving from a view of hardware definition followed by software development and integration to a perspective of system-level models which identify software-related requirements early in the system development process.

A top view of the process to develop system software requirements for each level of system decomposition is presented in Figure 2. Three steps to develop system software requirements for a level of system decomposition are shown. The basic process includes static requirements analysis, functional analysis, and dynamic analysis. Intermediate products of the basic process include system behavior diagrams which capture system sequences and component interaction, interface characterization, and timelines. The final products are system requirements allocations, preliminary software requirements, and preliminary interface requirements. In the case of weapon systems, the system requirements allocations are documented in a System/Segment Design Document, and preliminary software requirements are used in the development of preliminary software requirements specifications and preliminary interface requirements specifications in accordance with DOD-STD-2167A as described in Figure 2.

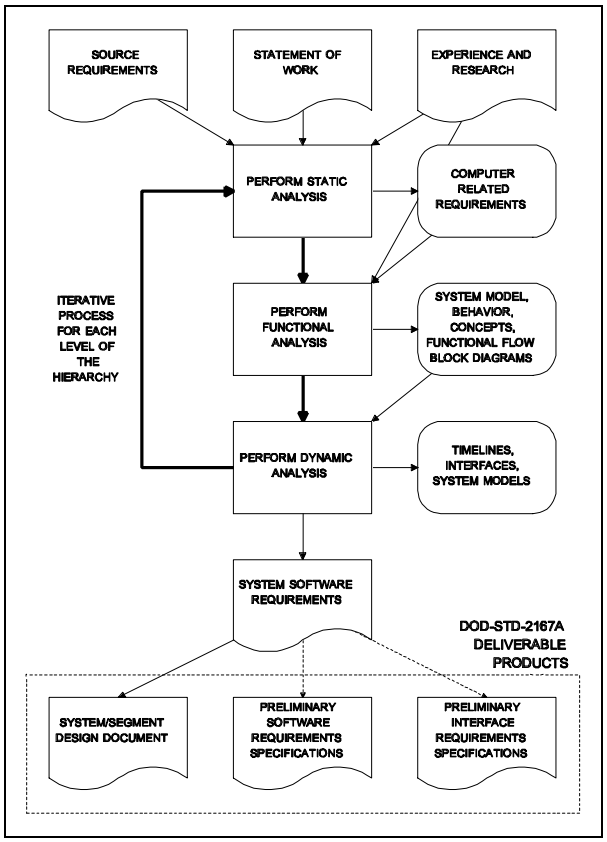

Figure 2. Product Focused System Software Requirements Development Process

Static Analysis. The first step in static analysis consists of a review of source requirements, statement of work, and documentation of preliminary development activities by system and software engineers working together to identify computer-related requirements. Computer-related requirements are requirements for functions which control the system, provide outputs to users, or monitor sensors or other input devices. In the case of weapon systems, key requirements that must be developed for implementation by system computers are target engagement activities, system sequences and displays, and operator inputs. Documentation of design activities and trade studies are essential for defining requirements for subsystems and components in complex systems. Guidance and control systems which require extensive development work must be generalized as a part of the system requirements development process. Legacy systems and preliminary point designs are included as inputs to the development team. An example of this step is portrayed in Case Study 1.

The second step in static analysis is for the team to resolve ambiguities, establish common vocabulary, amplify and clarify requirements, and identify issues. The result of this step is an update to the list of terms

and common vocabulary, and action items to be resolved between the system engineering team and the customer. For example, a requirement that merely states that general purpose processors shall have floating point instructions implemented as part of their instruction set architecture does not prohibit the use of fixed point arithmetic. This requirement may be met by processors that have floating point instructions implemented in their architecture, but are not executed directly. Generally, amplifying and clarifying a requirement means that the objective or intent of a requirement must be understood by all affected disciplines in order to meet the requirement. An example of establishing a common vocabulary is depicted in Case Study 2.

The third step for the development team is to verify achievability by comparing requirements with baseline concepts. Requirements allocated to subordinate elements must be achievable. The experience and perspective brought to the team by individuals from different specialty areas contributes significantly to the determination of achieveability. These concepts are demonstrated in Case Study 3.

Functional Analysis. Functional analysis activities drive to improving the baseline concept definition. The system/software requirements development team folds computer-related requirements into existing functional flow block diagrams and then updates and expands behavior diagrams as described in Case Study 4. State transition diagrams are used when available. The requirements development team then assesses current design concepts to determine if they are adequate to meet system requirements.

If alternate concepts are needed, then a concurrent engineering product design team consisting of domain specialists develops alternative design concepts to meet system or subsystem requirements. The product design team defines trade studies, selection criteria, and analyses to evaluate the concepts. The product design team then performs the trade studies and analyses, and evaluates the candidate concepts to determine if any of the concepts meet the selection criteria. If none of the concepts meets the criteria, then the process for developing alternate concepts is repeated until at least one concept is acceptable. After an acceptable concept is selected by the product design team, the requirements development team repeats the process of updating the functional flow and behavior diagrams to make them consistent with baseline design concepts.

Following selection of an acceptable concept, the functional flow and behavior diagrams are passed to the next step in the process.

Dynamic Analysis. The purpose of dynamic analysis is to develop rationale and correctly allocate requirements to hardware and software components. The system/software requirements development team defines and performs trade studies and analyses, identifies and resolves interfaces, and develops derived requirements for subordinate components based on system timelines and component characteristics. A representative example of dynamic analysis is presented in Case Study 5.

System simulation plays an important role in computer-related trade studies and analyses because it provides a way to identify and correct defects in requirements while the costs for correction are lowest. Requirements that are missing, inaccurate, incomplete, infeasible, or conflicting are identified during system level simulation (Roetzheim 1991).

The first step in performing dynamic analysis is for the requirements development team to allocate current level requirements to subordinate elements and expand system simulation models. If the requirements are allocated to more than one subordinate element, then the second step in performing dynamic analysis is for the requirements development team to define computer-related trade studies and analyses to define subsystem characteristics and identify interfaces. The third step, illustrated in Figure 3, is for the requirements development team to perform trade studies and analyses. In particular, evaluation of system timelines and performance characteristics provides the rationale for definition of derived requirements and interfaces.

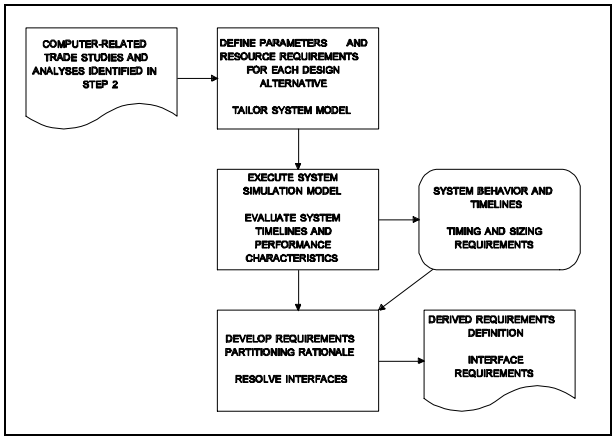

Figure 3. Computer-Related Trade Studies and Analyses Critical to Understanding Complex Systems

After performing trade studies and analyses, the

requirements development team documents the allocated and derived subordinate component requirements, interface requirements, system behavior, and timelines for the current level of system decomposition. If additional levels of decomposition are needed, then the requirements development process is repeated, beginning with static analysis as shown in Figure 2.

CASE STUDIES

The authors have experience on several programs in which system software requirements development problems were resolved successfully by small teams of specialists. Each multi-discipline team was comprised of a cadre of members from system engineering, software engineering, and quality assurance. The membership of each team was tailored to ensure proper discipline coverage by including representatives from affected disciplines (e.g., electronics systems, mechanical systems, specialty engineering, test operations). These system/software requirements development teams demonstrated that specific process improvements could be realized in solving software development problems by effectively integrating the insights and contributions of each specialist on the team. These process improvements are detailed in the following case studies.

Case Study 1: Static Analysis – Defining Requirements Based on Upgrading an Existing Design. One of the many difficult challenges facing requirements development teams involves the definition of requirements to modify and upgrade existing systems. Sometimes the familiarity with an existing system provides a team a great deal of information of the single point design while potentially obscuring the definition of the requirements.

This situation was illustrated on a program wherein an existing external interface unit was to be integrated into the weapon system. Since the original interface unit was built as a piece of test equipment for a concept validation program, no formal requirements were written. However, to upgrade the weapon system, the interface unit requirements were now required in order to allocate functions properly within the new system architecture.

A team of system, software and electronics engineers was formed to solve this problem. The team convened a meeting with approximately a dozen engineers from the predecessor program to determine the interface unit's requirements. These specialists were very familiar with the interface unit implementation and were able to elaborate on what functions the unit performed and how the unit performed these functions.

However, the specialists were not always sure why certain features had been incorporated into the design of the unit. Lacking the underlying rationale for these features, it was not clear whether these features should be retained in the upgraded unit. The absence of interface requirements necessitated the team to perform a bottom-up analysis of the design to derive these requirements for the unit. Equipped with these requirements, the team was able to productively develop the software for the upgraded configuration.

This case study demonstrates the downstream value of developing and maintaining accurate requirements documentation and traceability for a system so that system improvements can be made quickly and efficiently.

Case Study 2: Static Analysis – Establishing a Common Vocabulary and Set of Terminology. A continuing challenge faced by requirement analysis teams is a lack of a common frame of reference. One team of specialists was able to solve this problem by analyzing and discussing the source requirements for a missile prior to developing the missile software requirements. The team first identified a set of system level requirements which pertained to the system behavior of the missile. This task consisted of listing the requirements which were applicable to the missile software, eliminating the requirements that only applied to the missile hardware, and analyzing the requirements which applied to both missile hardware and software. The team then organized these requirements in a logical manner by developing a table of contents for the missile requirements document.

By following these steps, it became evident that agreement on concepts and terminology was essential. For instance, the definition of “missile algorithm requirements” was not interpreted in the same manner by each team member; some considered an algorithm a set of mathematical equations while others regarded it as program language code. Since the purpose of defining these algorithm requirements was to facilitate missile software design, coding and testing in Ada, the team recognized the importance of establishing agreement on the terminology “missile software algorithms” prior to defining requirements. To decide this issue, the team conducted a lively debate examining the advantages and disadvantages of each point of view. Some team members argued that a substantial amount of risk was reduced by developing and testing the missile algorithms using an engineering computer system programming language (e.g., FORTRAN) and subsequently providing this

software code to software engineers for conversion into a programming language suitable for embedded real-time systems (e.g., Ada, Pascal, JOVIAL, assembly language). This approach, which has been successfully implemented on other weapon system programs, was held in high regard by these team members.

Other team members asserted that, although this approach has been implemented successfully on other programs, it has several serious shortcomings. First, these team members felt this approach was inconsistent with the fundamental purpose for requirements, namely, to define the problem statement without specifying the method of implementation. In contrast, a source code listing is the implementation of a point solution and, therefore, not a set of requirements. Secondly, if the requirements have been defined as lines of code, then each interim step which is computed by the algorithm is regarded with the same importance as the algorithm result. These team members contended that the level of detailed testing of the interim steps could add hundreds of hours of labor to the cost of the program which can be avoided if the basic underlying requirements for the missile algorithms were identified whenever possible. Finally, these team members noted that any changes to the missile algorithms would necessitate changing the source code. Since the source code was viewed as the missile algorithm requirements, a change to the source code would precipitate a formal change to the contract. These team members proposed that "missile algorithm requirements" define the performance of the algorithms in terms of the internal missile states and the missile environment.

After much discussion, the team agreed to define the terminology "missile software requirements" as the description of missile algorithm performance and to defer the development of the corresponding program language code during the design phase. As a result of this exchange of ideas during the early phase of the program, the team members were able to evaluate the consequences of these alternatives from the outset, and select a solution which both considered the affected disciplines and the elements of the product life cycle.

Case Study 3: Static Analysis – Correcting System Software Requirement Deficiencies in System Specifications. System requirements analysis has been typically conducted from a predominantly hardware perspective. While this perspective may be entirely appropriate for certain hardware oriented implementations, it limits early visibility of software-related requirements in systems which are comprised of both hardware and software elements. Developing insight into these software related requirements during the system analysis phase can reveal requirements which are ambiguous or which are in error. This early evaluation of system software requirements allows engineers to clarify requirements and to negotiate correction of requirement deficiencies with the customer prior to the start of subsequent subsystem level requirements analysis.

In one instance, software engineering team members participated in an evaluation of a system specification. Their contribution resulted in a lengthy and productive dialogue with the customer which amplified software related requirements issues concerning program loading, reprogrammable memory, resistance to inadvertent operation, abnormal processing, safety, and fault recovery. In retrospect, it was agreed that determining the achievability of the requirements at the system level provided the necessary insight to develop consistent and clear lower level software requirements.

Case Study 4: Functional Analysis - Driving to a Baseline Concept Definition. One requirements development team experienced difficulty in defining the system software requirements at the interface between the fire control system and the missile. This team solved this problem by developing scenarios which described the behavior of fire control system and missile interactions. Domain experts were consulted to identify concurrent or near simultaneous tasks within these scenarios and to characterize their durations. These task sequences were organized into timelines and evaluated for completeness and closure of data involving parallel tasks and analyzed for staleness of information to assure the stability of the data. As a result of these analyses, the team was able to specify the system software requirements for this interface in terms of the internal states and environment of the fire control system and the missile.

Case Study 5: Dynamic Analysis – Allocating and Deriving Hardware and Software Requirements. A small team of system, software and mechanical engineers demonstrated the leverage available to a program when the same members of an engineering team participate throughout all phases of a development program. Their contributions were especially noteworthy in solving a control systems problem for a stabilized weapon platform. In this problem, the weapon platform was required to rotate so that, during its excursion, interference with vehicle hardware components (e.g., antennas, sensors) was avoided.

The team's first step in the dynamic analysis process was to determine the mechanical interferences between the movement of the platform assembly and the vehicle. The team then allocated the implementation of control to software (algorithms) and hardware (guides, stops, brakes).

To define software control, the team expanded the system simulation models to develop a guidance and control algorithm that met these requirements. After additional analysis, it was determined that the algorithm would monitor platform azimuth motion and command platform motion in elevation to ensure clearance. The team defined a trade study to examine three alternatives to determine elevation as a function of azimuth: (1) elevation provided by resolvers, (2) elevation provided by resolvers and computed angular rates, and (3) elevation provided by resolvers and angular rates provided by rate gyros. The third alternative identified a new requirement for rate gyros.

To perform this trade study, the team identified three parameter requirements to evaluate the design alternatives. The parameter requirements were determined by consulting the appropriate specialists who provided the platform mass properties, the torque capability provided by the azimuth drive motor and elevation drive characteristics provided by hydraulics. Upon receipt of these data, the team tailored the simulation model by refining the transfer function gains based on inertial moments and characteristics of the elevation and azimuth motors. The tailored model was then executed to verify that the algorithm met the clearance requirements and determine the required sampling rate for the algorithm.

Based on this analysis, the implementation of the algorithm was allocated to the appropriate hardware and software component in the weapon system. The design concept selected provided the algorithm position data from resolvers and angular rate from rate gyros. Subsequently, this team implemented and tested this solution successfully on the weapon system during integration and test.

As this case study attests, the effectiveness of a multi-discipline team developing the software related products of a system can be significantly increased by the formation of this team during the system requirements phase of a program and continuing this team's effort through the design, integration and test of these products.

Environment Maturation

Too few managers are aware that the continuing dramatic growth in demand for software products has been accompanied by a commensurate increase in software technologies. Pressed for time and often lacking software development experience, most managers are unaware of the software world's recurring problems and “frontier” solutions (Cusumano 1991). Significant improvements in software development productivity, which includes the requirements development process, will be very difficult without concurrently maturing the management environment.

There are three useful approaches for improving management understanding of software and requirement development issues. First is through a sophisticated customer who is able to ask critical process development questions and who places performance criteria into the contract for monitoring progress in developing the software product. In the defense industry, government project offices, many with personnel trained by the Defense System Management College, are making headway in compelling contractor management to deal realistically with software development issues.

The second, or do it yourself, approach is for the system/software requirements development team to make the time to articulate the recurring challenges software development projects typically experience and to explain what can be done to mitigate those effects. Early development and periodic review of informative metrics, risk assessment, critical path analysis, and an easy to understand process all serve to keep management in touch with the software development team's accomplishments and problems.

A third source of pragmatic information available to those firms with a matrix management structure, is the expertise resident in functional organizations. Providing this knowledge to project management can be done through sponsoring management training courses, participating in special reviews of the software product development process, promulgating procedures for best practices, and informing project management of significant process improvements made by other organizations.

Maturing the management environment requires steadfast focus on describing practical solutions to real problems. System engineering practitioners must consider it their charter to lead the way in developing a supportive organizational culture.

CONCLUSION

Improving the system software requirements development process requires a commitment by all elements of management to nourish an environment conducive to multi-discipline team communication, learning and product development.

Primary attention is given to devising a process harnessing and aligning the formidable cognitive skills of all team members. The vehicle for starting this activity is provided by system engineering in the form of a conceptual framework for attacking the system requirements development problem. System engineering also provides guidance in working through the inevitable conflicts. Selection of tools to further improve team productivity follows process definition. As requirement development team members grow to trust one another and engage in a continuing dialogue, success in the form of quality products delivered within reasonable schedules will become the norm rather than the exception.

REFERENCES

Babbitt, Albert E., "The Changing Role of the Systems Engineer." Keynote speaker at NCOSE's Third Annual International Symposium, Arlington, Virginia, July 26, 1993.

Bridickas, Paula, and Obsenski, Sally M., "Software Challenges in Mission-Critical DOD Systems." *General Accounting Office, GAO/IMTEC-93-13,* December, 1992

Brown, P. J., and Cady, K. A., "Functional Analysis Vs. Object Oriented Analysis: A View From the Trenches," *Proceedings of the Third Annual International Symposium, National Council on Systems Engineering (NCOSE),* Arlington, Virginia, July 26-28, 1993.

Churchman, C. W., Ackoff, R. L., and Arnoff, E. L., *Introduction to Operations Research*, John Wiley & Sons, New York, 1957.

Cusumano, Michael A., *Japan's Software Factories,* Oxford University Press, New York, 1991.

Hogg, Allen, "Developing Software Not So Mechanical, New Study Says." *Engineering Times*, Volume 15, Number 7, p. 10, July, 1993.

Rechtin, Eberhardt, "Frontiers of Systems Engineering: Increasingly Smart and Complex Systems." Banquet speaker at NCOSE's Third Annual International Symposium, Arlington, Virginia, July 26, 1993.

Roetzheim, Williams H., *Developing Software to Government Standards*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

Senge, Peter M., *The Fifth Discipline*, Doubleday Currency, New York, 1990.

Thurow, Lester, *Head to Head*, Warner Books, New York, 1993.

Yourdan, Edward, *Decline & Fall of the American Programmer*, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1993.

BIOGRAPHIES

Phillip J. Brown is a Technical Project Manager, System Engineering, for Loral Vought Systems Corporation. He has a Bachelor of Civil Engineering from the Georgia Institute of Technology and a Master of Science in Industrial and System Engineering from the Ohio State University. His 27 years of experience include system engineering work on aircraft, cruise missiles, direct fire missiles, indirect fire rockets, air defense missiles, and satellite systems. He is a registered professional engineer, a member of the National Society of Professional Engineers and a member of NCOSE.

Alexander E. Iwach is a Senior Engineering Specialist, System Engineering, for Loral Vought Systems Corporation. He has a Bachelor of Science in Electrical Engineering from the University of California, Los Angeles. His 14 years of engineering experience in DOD programs for missile, aircraft, satellite, shipboard and submarine applications includes design engineering, integration and test, and system requirements analysis. He is leading a system/software requirements development team and has three years experience in resolving multi-discipline technical issues during the development and evaluation of system software requirements.

Donald R. Williams is manager of the Requirements Definition group for Loral Vought Systems Corporation. He has been developing system and software requirements for five years on two major weapon systems and has been in the aerospace business for twenty-three years. He is leading the System Engineering effort to implement improvements to the requirements development process and to acquire computer-aided system engineering tools for tracing requirements, developing functional analysis diagrams, and performing system analyses.