A HOLISTIC MANAGEMENT FRAMEWORK FOR SOFTWARE ACQUISITION

Yacov Y. Haimes, Richard M. Schooff, and Clyde G. Chittister

In the face of the federal government's recent downsizing effort and the increasing pressure to reduce government expenditures and improve oversight of the use of public funds, reform of government acquisition policies and practices has been a major initiative. While software acquisition is increasingly included in the discussion of acquisition reform, the concept of software acquisition is in fact a misnomer (even though it is included in the title of this paper). Our government acquires systems, not exclusive software per se; these systems increasingly include a major software component.

Software's critical impact on system reliability and performance makes effective software acquisition policies and strategies essential. This article considers software acquisition issues being addressed in an ongoing project conducted by the Center for Risk Management of Engineering Systems at the University of Virginia, and the Software Engineering Institute, Carnegie Mellon University.

Because preventive action is the key to successful risk management, one must plan how to avoid software system risk early in the system life cycle—before the software is purchased. The federal government's expenditures for software products and services are tremendous, and government acquisition procedures, regulations, and results are readily available for public review. This article focuses on government mission-critical software acquisition, and particularly software acquisition efforts of the Department of Defense (DoD). The results of this study will be useful to other government agencies as well as to the private sector.

Effective management of modern, complex processes such as software acquisition requires capable, mature direction. Good management of technological systems must address the holistic nature of the system in terms of its hierarchical, organizational, and functional decision making structure; various time horizons; the multiple decision makers, stakeholders, and users of the system; and the host of technical, institutional, legal, and other socioeconomic conditions that require consideration. Good management also implies the ability to identify program risks, evaluate their potential adverse impact, and effectively incorporate risk con-

siderations in the decision making management framework.

The fundamental goal of this article is to describe the development of a quantitative risk management framework for software acquisition, centered on hierarchical holographic modeling (HHM) (Haimes, 1981). This management framework can then be generalized for application to other emerging large-scale systems and processes.

Here we build on current knowledge and experience in software acquisition, software engineering, management, and decision making, and in risk analysis, and incorporate this knowledge with the principles that guide systems engineering. The proposed holistic vision of the software acquisition process and the proposed methodological framework for the management of this process are ultimately aimed at controlling the risk of projects' cost overruns and completion schedule delays. For a discussion of managing software technical risk, see Chittister and Haimes (1994a).

The Approach

Recognizing that software is only one component of a larger system (and that software is, itself, a system made up of multiple components, elements, and entities), one must manage software acquisition by considering the larger picture and take a systemic approach to resolving the complex interconnections of the multiple participants, activities, events, risks, and other process elements. With the ever-increasing importance and complexity of the software component of modern systems, it is essential that software acquisition be addressed in terms of its overall system.

We will not explicitly address the multiple aspects associated with the software acquisition process. Our objective is instead to develop a modeling framework that will enable the consideration of such complexities and interconnectedness, and then outline the approach as it applies to particular subcomponents of software acquisition: the *process* vision, composed of its multiple stages of activities; the *pro-*

Dr. Yacov Y. Haimes earned his PhD. degree in Systems Engineering from the University of California, Los Angeles. He holds the Lawrence R. Quarles professorship in the School of Engineering and Applied Science at the University of Virginia, and is a member of the Systems Engineering and Civil Engineering faculties. He is the Founding Director (1987) of the University-wide Center for Risk Management of Engineering Systems at the University of Virginia. On the faculty of Case Western Reserve University for 17 years, he was Chair of the Systems Engineering Department, and Director of the School-wide Center for Large-Scale Systems and Policy Analysis.

MAJ Richard M. Schooff is an assistant professor of mathematical sciences at the United States Air Force Academy, Colorado. He received a Ph.D. in systems engineering from the University of Virginia. He is a member of the Institute for Operations Research and the Management Sciences (INFORMS) and the International Council on Systems Engineering (INCOSE).

Clyde G. Chittister earned his BS degree in Computer Science from Pennsylvania State University and has over 29 years experience in software and systems engineering. At the Software Engineering Institute, he recently managed and led technical development of their Risk Management Program.

gram consequence vision, which includes technical performance, cost, and schedule; and the *community maturity* vision, which includes user, customer, contractor, and technology.

This work draws on a holistic representation of such complex systems and processes termed hierarchical holographic modeling (HHM). Fundamentally, HHM is grounded on the premise that complex systems and processes, such as the software acquisition process, should be studied using more than one single model, view, or perspective. HHM possesses a dual nature: it is holistic, investigative paradigm, and a mathematically sound, hierarchical, multiple-objective decision making methodology. Exploiting the inherent synergy of HHM's duality provides the necessary theoretical, methodological, and practical foundation for a risk assessment and management framework for the software acquisition (and other large-scale) process. The approach of this work (see Figure 1) is to represent software acquisition by an HHM model, enhance and extend HHM's investigative capabilities for exploring and modeling the various decompositions and submodels, and then extend the quantitative capabilities of HHM for resolving the conflict and overlap associated with the objectives of the various submodels.

HHM for Software Acquisition

Figure 2 depicts an HHM model for software acquisition. The six decompositions, or perspectives, of the software acquisition HHM indicate the multiple dimensions associated with software acquisition. The acquisition process requires the participation of numerous organizations and individuals with specific functions and responsibilities as well as requirements to coordinate their activities with the other parties. These organizations have their own goals and objectives, which are often in competition with each other. Risks and uncertainties inherent to the software acquisition process complicate the several key decisions that, in turn, affect the ultimate software product. Only by exploring the dimensions and perspectives of the overall systems acquisition and properly coordinating the objectives and requirements from each model perspective can one effectively manage the software acquisition process.

Software Acquisition Submodels. Software acquisition capability maturity implies the existence of, and adherence to, a specified, documented, and repeatable software acquisition *process* that is managed through *quantitative* strategies (Sherer and Cooper (draft) 1994). Therefore, prerequisite to a mature software acquisition customer community is the establishment, analysis, and acceptance of a software acquisition process as well as an appropriate quantitative management framework.

The multiple views the HHM provides are represented by the various hierarchical holographic submodels (HHSs), where each HHS addresses the system from one particular perspective, or dimension. As each perspective may have its own unique representation of issues, limitations, and factors, this diversity would likely lead to HHSs of different modeling topology, nature, or structure (e.g., analytical vs. descriptive models) (see Figure 3). For instance, the *process* view of the software acquisition HHM represents a progression of events or a sequence of decisions in the

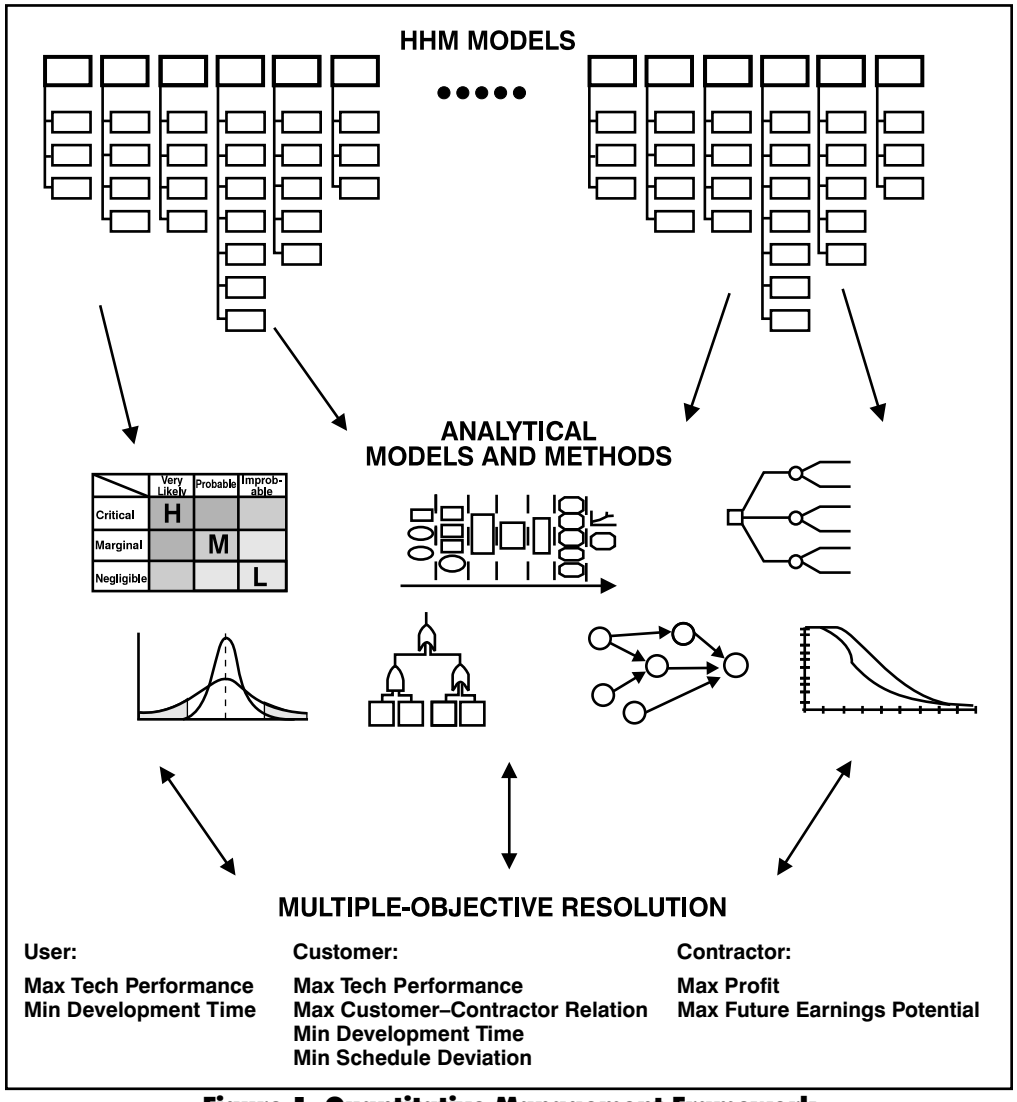


Figure 1. Quantitative Management Framework

software acquisition process that may be analyzed through process modeling (Blum, 1992), and then quantified by one of many appropriate tools, such as decision tree methods or multiple-objective decision tree methods (Haimes et al., 1990a). The *cost* element of the *program consequences* decomposition could be modeled by probability distribution analysis, supported by analytical software cost estimation models (e.g., constructive cost model (COCOMO) (Boehm, 1981)). The software *technical* element of the *program consequence* view may be quantified in terms of one of several measurable objectives (e.g., reliability, availability, maintainability) and may use fault tree analysis or Markov process models in its solu-

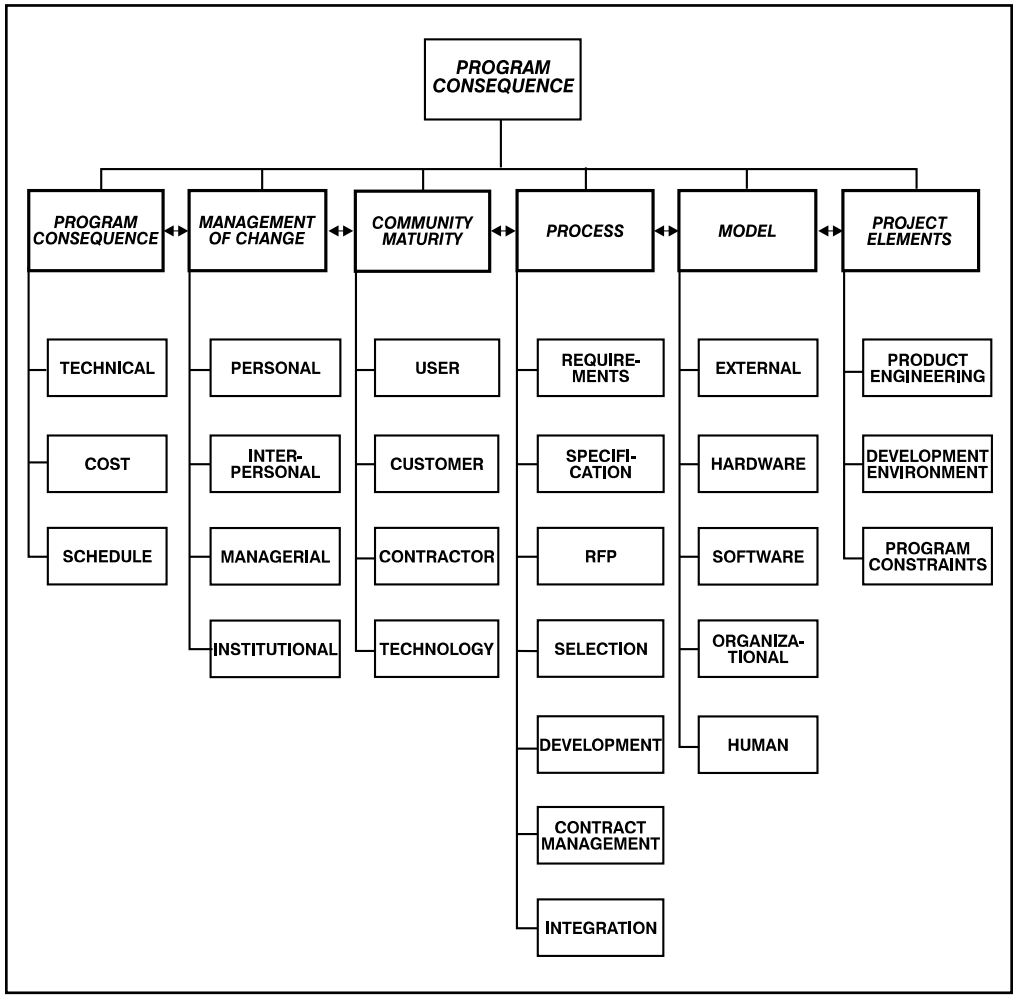


Figure 2. Hierarchical Holographic Model for Software Acquisition

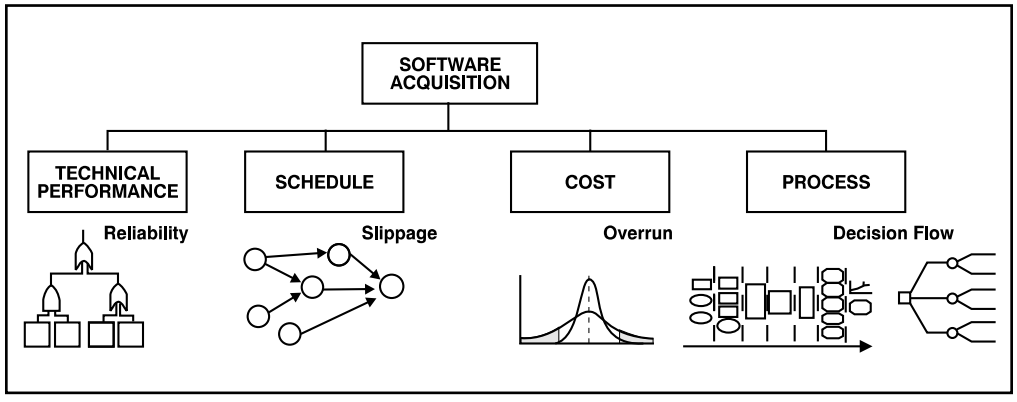


Figure 3. Demonstration of Analytic Methods for HHS Solution

tion (Johnson, 1989). Similarly, the *schedule* perspective may be analyzed through program evaluation review technique (PERT) or related methods (Boehm, 1981). While each HHS can then be solved independently, a coordinated solution to the overall problem must be resolved at the highest level of the HHM.

In this article, we focus on the *process* perspective of software acquisition, developing a modeling framework that describes the participants, inputs, activities, decisions, and interrelations of the various elements of software acquisition.

Multiple Objective Resolution. The full value of structuring the software acquisition analysis in this manner is realized by using a multiple-objective construct. Consider, for example, an over-simplification of the objectives of the end-user in a software acquisition effort. The user wants a system that achieves a high level of technical performance, and generally prefers that the system be developed and delivered as soon as possible. These statements can be formalized as F_1; maximize technical performance, and F_2; minimize development time, where F_i represents a specific objective. As Figure 1 shows, in addition to the user's multiple objectives, similar multiple statements can be expressed for the other participants in the software acquisition process.

The multiple-objective approach provides a context for achieving a "win-win-win" environment for the user, customer, and contractor. When associated with each view of the hierarchical holographic models, this approach not only provides a promising structure for resolving competition among participants, but also helps decison makers consider system trade offs such as between performance and cost, or between schedule and technology.

Background: Software Acquisition Issues

Major issues in software acquisition today include the criticality of the software component in modern systems, increasing pressure for reform initiatives, and the need for a less adversarial acquisition environment.

Software's Criticality

As computer use has become central to organizational activities and engineering system design, the software component of these systems has become increasingly important. That criticality is well docu-

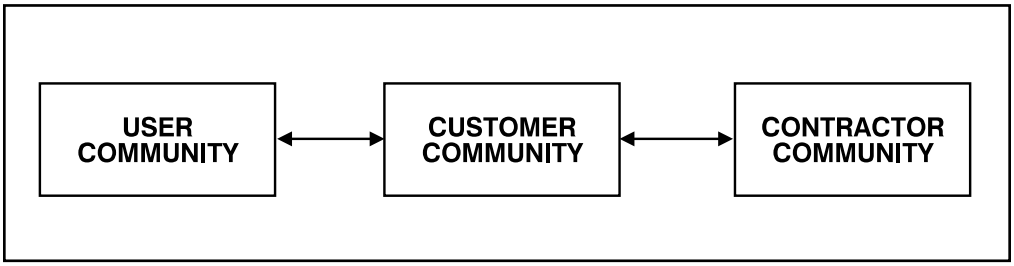


Figure 4. Hierarchical Holographic Model for Software Acquisition

mented and universally accepted (Boehm, 1984; Haimes and Chittister, 1993; Blum, 1992; GAO, 1990). Chittister and Haimes (1994a) document a shift in importance from hardware to software within modern systems. Software has become the principal system design component, as well as the principal factor affecting system quality. In fact, software has been described as the "Achilles' heel" of modern weapon systems because it is a key determinant of development schedules and because key functions such as navigation, enemy detection, and fire control depend on it (GAO, 1992b). Examples of system failures whose root was failure of the software have been well publicized (e.g., GAO, 1992a). Due to the continued expansion of software's commanding role in modern systems (and the budget for such systems), the ability to effectively acquire and integrate software into these systems will be increasingly important.

Participants in the Software Acquisition Process

The three principal participants, or groups of participants, in an acquisition endeavor are the user, the customer, and the contractor. In government acquisition, these groups rarely constitute single individuals, but each often comprises one or more organizations and their representatives. Under current practice the user and contractor communities generally communicate through the customer community (Figure 4). Re-engineering reform initiatives of the acquisition process would encourage more direct user-contractor communication.

In DoD acquisition activities, the user community may be a major Command from one of the services, a Unified or Joint Command, one of the military departments, or even DoD itself. Difficulties in directly identifying a single user or group of users are evident. Quite often, a representative user is designated to act on behalf of the larger user organization. In general, users are responsible for identifying and specifying operational needs, validating the criticality of those needs, and receiving the completed system.

The customer, which in DoD terms refers to the acquisition authority, and more specifically the acquisition managers and program managers, is the purchasing agent acting on behalf of the user. The customer is responsible for accurately translating the user's needs into the contractual language of systems requirements, writing the contract documents, selecting the best qualified contractor(s), monitoring system development, accomplishing contract management and negotiation functions, and conducting system testing and acceptance.

The contractor must develop a system that meets the requirements, guidelines, and limits stated in the contract. The ability of each participant to complete its task and effectively coordinate activities with the other participants is central to a successful acquisition program.

Software Acquisition Research

Software Development. Over the past three decades, software development practices and processes have been much studied. Early efforts to apply and extend the practices and principles of engineering to software led to the development of a new discipline: *software engineering*. More recent development of software life cycle models (Boehm, 1981) and software process development models (Feiler & Humphrey, 1992), have helped to bring a

Table 1. Initial Software Acquisition Maturity Model (SAMM), based on Sherer & Cooper (1994)

LEVEL	FOCUS	KEY PROCESS AREAS	RESULT
5 Optimizing	Process Optimization	Continuous Process Improvement Technology Insertion	Productivity & Quality
4 Controlled	Quantitative Management	Process Management Software Quality Management Defect Prevention Asset Management	
3 Defined	Integrated Project Management	Software Project Planning & Mgmt Process Focus Software Risk Management Project Team Coordination Software Engineering Monitoring Process Assurance Training	
2 Organized	Contract Management	Software Contracting Preparations Software Contract Initiation Requirements Management Software Contract Tracking Software Contract Oversight Acceptance, Transition, & Support	
1 Initial	Product and Resources		Risk

degree of standardization and process improvement to the software development community.

Much research has focused on improving the software development process (e.g., Humphrey & Kellner, 1989; Kellner, 1991; Heineman et al., 1994). Business realities such as strong competition, pressure for increased profits, and external regulations have spurred the momentum for an improved software development process (Austin & Paulish, 1993). Improving the software development capabilities of software vendors by improving their software development process maturity is the focus of the Software Engineering Institute's Capability Maturity Model (CMM) (Paulk, Weber, Garcia, Chrissis & Bush, 1993). This tool "provides software organizations with guidance on how to gain control of their process for developing and maintaining software and how to evolve toward a culture of software engineering excellence" (Paulk et al., 1992). Other related research including software process assessment, software metrics, CASE tools, software engineering, software quality, concurrent engineering, and software reliability engineering have been accomplished predominately on behalf of the software contractor, to aid in the actual develop-

ment of software and to provide visibility, through metrics, to the customer.

Software Acquisition Research. Unfortunately, when compared to software development research, relatively little has been studied and written for the customer's benefit, i.e., the development of guidelines and instruction of how to effectively acquire a software product and manage the acquisition effort. Recent, original work by Sherer and Cooper (1994) and parallel research (Baker, Cooper, Corson, & Stevens, 1994) have led to initial versions of a software acquisition maturity model (SAMM) for maturing the acquisition capabilities of the customer community. While the development of a SAMM is still in its infancy, and revisions of draft models are to be expected, the initial results appear promising. As with the CMM, the SAMM is both an evaluative tool as well as a way to increase a community's capability. An initial version of the SAMM (Table 1) proposes a structure of five progressive levels of maturity for software acquisition capability, along with key process areas for each level. Increasing the acquisition capability of the customer community improves productivity and program quality while reducing risk.

The maturity progression is intended as an upward flow: satisfying the requirements of one level leads to higher level functions. While a lower level organization may be practicing elements of a higher maturity level, full progression to the next higher level is contingent on all key process areas being fulfilled (Sherer & Cooper, 1994). Maturity in software acquisition capability implies a verified, repeatable, effective *process* and a *quantitative management* framework for governing that process. At Level 3 maturity, the customer employs an integrated project management, risk management, and process management strategy (Figure 5). Level 4, *quantitative management*, requires the customer to set and monitor quantitative quality goals for processes and products (Sherer & Cooper, 1994). The quantitative management framework described here establishes the necessary vision and practices required for the customer community's acquisition maturation.

Software Risk Research. As in many fields, software development has experienced its share of project disasters. Risk assessment and risk management specifically for software systems is a relatively recent area of research. Boehm's groundbreaking work (1981) introduced methods of decision making under uncertainty to this field. The Software Engineering Institute (SEI) of the Carnegie Mellon University, a federally funded research and development center with a broad charter to address software engineering technology, is a central participant in current software risk research. One of the SEI's focus areas is software risk management. The SEI risk paradigm (see Figure 5) depicts risk assessment and risk management as a process with several phases—identification, analysis, planning, tracking, and controlling functions—which parallel the general concepts of risk identification, risk quantification, risk analysis, risk management, and risk mitigation.

One of SEI's major functions has been to develop methodologies for addressing several of the software risk management phases. The Risk Taxonomy-Based

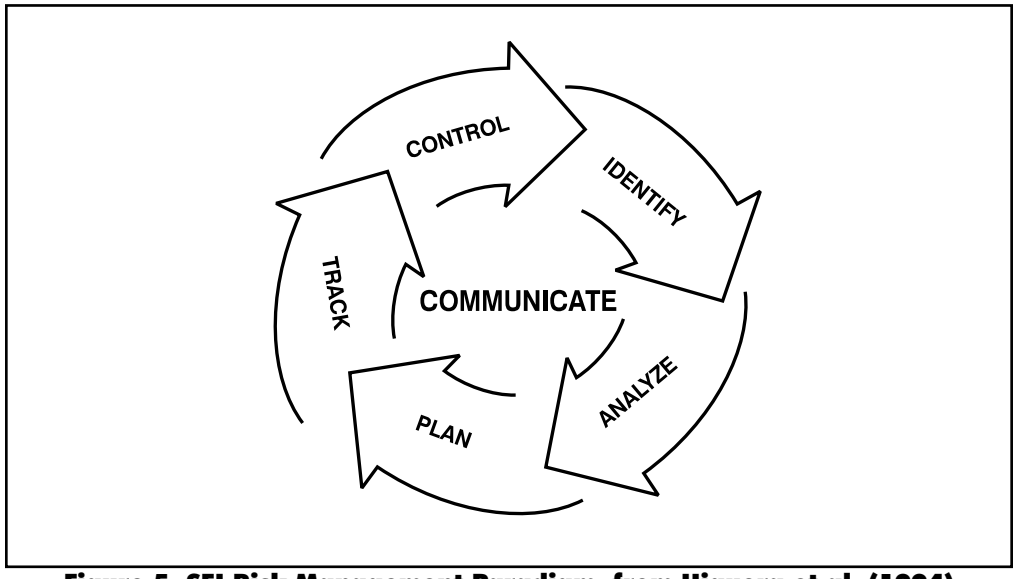


Figure 5. SEI Risk Management Paradigm, from Higuera et al. (1994)

Questionnaire (TBQ) is "a method for systematic and repeatable identification of risks associated with the development of a software-dependent project" (Carr, Konda, Monarch, Ulrich, & Walker, 1993). The Software Risk Evaluation (SRE) method (Sisti & Joseph, 1994) is a quantification method for assessing and analyzing risks for a project. A recent development, Team Risk Management (TRM) (Higuera et al., 1994), extends risk management practices to include team-oriented activities involving the customer and contractor, where these groups apply risk management methods together. TRM is a framework for cooperative risk management; it relies on strengthened risk communication between groups and incorporates the TBQ and SRE as fundamental risk analysis and assessment methods.

Acquisition Reform

Acquisition reform is currently a major initiative within government. The administration's *Plan for Economic Development in the Technology Sector* (Clinton and Gore, 1993) provides a broad plan to reinvent the federal acquisition system. The need for improvement has been expressed across government sectors. In a survey of senior executives in the federal government, a majority stated that "the procurement process frequently results in procurement decisions that are neither cost effective nor in the best interests of the government" (SAMERT, 1994). The Secretary of Defense stated that "the existing DoD acquisition system can be best characterized as an 'industrial era bureaucracy in an information age' " (Perry, 1994).

Although the defense acquisition system provides a structured, highly regulated process for systems acquisition, the regulations and restrictions imposed on the process over time have often hampered efforts toward efficiency and creativity. In

recent testimony before the Senate Committees on Governmental Affairs and Armed Services, the Undersecretary of Defense for Acquisition and Technology John Deutch, summarized the problem (1994):

> The system is too cumbersome and takes too long to satisfy customer requirements. In addition, the system adds cost to the product procured. DoD has been able to develop and acquire the best weapon and support systems, not *because* of the system, but in *spite* of it. And they did so at a price—both in terms of the sheer expense to the nation and eroded public confidence in the DoD acquisition system.

The dilemma facing any government acquisition reform is how to provide sufficient oversight for the expenditure of public funds, with the least amount of intrusive policy and regulation, yet still accomplish the acquisition goal. Acquisition reform must ensure the continued existence of important safeguards designed to ensure the integrity of the acquisition process. Reforms that reduce regulatory oversight may possibly increase the risk of mismanagement of public funds. Lawmakers often must balance the risks inherent in reducing oversight with the cost to industry and government to comply with oversight regulations.

Of particular concern regarding software acquisition are some of the key characteristics associated with software acquisition: Software evolves rapidly, it is difficult to explicitly define and specify, acquisition officials often lack software understanding, and there is difficulty in estimating project costs and time requirements. The current acquisition process requires an average of 16 years to field a new weapons system (Pages, 1994), while software and computer product life cycles are as short as 1 or 2 years. A software solution could become obsolete before it is delivered.

Other ongoing issues of software acquisition include: incentive-based contracting vehicles and their appropriate application, relaxation of "mil-spec" requirements procurement, and a move to commercial off-the-shelf (COTS) software purchases. DoD's acquisition system of audit activities, which includes the possibility of criminal sanctions for violating established procedures has cultivated a climate of adversarial relationships rather than partnerships between customer and contractor (Defense Science Board, September 1987). This has also led to a risk-adverse mentality for the program manager and the customer community; there is no reward for taking risks and huge penalties for failure. There is a strong need for a "win-win-win" environment for the user, customer, and contractor communities without the institutionalized mistrust of the current system.

Holographic Modeling for Software Acquisition

The dominating attributes of modern, large-scale systems are their multidimensional nature, hierarchical competing objectives, multiple participants, a wide array of pertinent issues demanding consideration, and inherent uncertainty. The complexity of the software acquisition process and the multiplicity of the parties

involved in that process from planning, to development, to delivery, and to maintenance defy the success of any attempt to represent this process by any one single model, structure, or paradigm. In fact, representation within a single model of all the aspects of a large-scale system is so impracticable as never to be seriously attempted. However, the inability to address this critical attribute of large-scale systems is a major stumbling block. As Haimes (1981) originally stated,

> To clarify and document not only the multiple components, objectives, and constraints of a system but also its welter of societal aspects (functional, temporal, geographical, economic, political, legal, environmental, sectoral, institutional, etc.) is quite impossible with a single model analysis and interpretation. Given this assumption and the notion that even present integrated models cannot adequately cover a system's aspects per se, the concept of hierarchical holographic modeling constitutes a comprehensive theoretical framework for systems modeling.

Acceptance of the HHM

Since its origin in 1981, the HHM has provided a general framework for addressing the modeling of complicated, multiple objective problems of large scale and scope. While not receiving an abundance of direct reference in the literature, nevertheless HHM's multivisionary approach to problem definition and risk identification has been widely, although often indirectly, accepted. For example, Yeh et al. (1991) reject the single-model, step-by-step approach in managing complex software engineering development. They state:

> In most current practice, decisions are based on a one-dimensional view prescribed by waterfall-like models. This view consists of a single explicit perspective on a set of activities and their interdependencies and schedule—which form an activity structure. In the waterfall model, the activities are sequentially scheduled into phases: requirements analysis, design, codes, tests, and so on. Other models suggest adding a second perspective, a communication structure ... still models like the spiral model of software development and enhancement and models based on process-maturity levels suggest including process design and monitoring.

Such an argument, highlighting the limitations of a single model to capture the multiple aspects of a complex system, underscores the contributions of the HHM approach. Even the title of a recent text, *Software Engineering: A Holistic View* (Blum, 1992), denotes the criticality of considering the multidimensionality of complex processes such as software development. Throughout his book, *Metasystems Methodology*, Hall (1989) uses HHM to recount the history of systems methodology, and to distinguish the varied applied systems methodologies from each other. He states:

> History becomes one model needed to give a rounded view of our subject within the philosophy of *hierarchical holographic modeling*, defined as using a family of models at

several levels to seek understanding of diverse aspects of a subject and thus comprehend the whole.

Many current risk identification methods, evaluation techniques, and issue investigation schemes build on the general principles embodied by the HHM. For example, careful examination of the SEI taxonomy (Carr et al., 1993), its purpose, and methodology indicate a vision that is harmonious with HHM: the taxonomy is hierarchical in structure, is constituted by progressive levels of detail and abstraction, provides a way to address the multiple dimensions of a problem, and serves to identify areas of concern in a software acquisition endeavor. Recognizing the kinship of these methods to the HHM strengthens the parent methodology, and further demonstrates the efficacy, appropriateness, and desirability of the HHM as a framework for analyzing software acquisition and other large-scale problems.

HHM for Software Acquisition

The role of models is to represent the intrinsic and indispensable properties that serve to characterize the system: that is, good models must capture the essence of the system. Clearly, the multidimensionality of the acquisition process, and the large number of groups, organizations, and people of many disciplines that are engaged in this process defy the capability of any single model to represent the essence of the acquisition process. To overcome the shortfalls of single planar models and to identify all sources of risk associated with the software acquisition process, an HHM framework will be adopted here. HHM assumes an iterative approach to providing the structure for identifying all risks. If one fails to identify a risk source with the current views of the HHM, then expansion of the model to include a new decomposition is possible. This process, itself, will eventually capture all risk sources.

The HHM developed here constitutes the six subdivisions or perspectives shown in Figure 2. Note that each subdivision is represented within the HHM framework by a separate submodel. This also implies that, when modeled, each of the six subdivisions and the corresponding sub-subdivisions will be represented by a set of objective functions; constraints; and decision, state, exogenous, and random variables. Obviously there will be common goals and objectives, as well as separate and possibly conflicting and competing objectives.

For notational purposes, the model of a subdivision will be termed hierarchical holographic submodel (HHS); thus, there are six HHSs in the HHM of the acquisition process. Here we do not delve into the theoretical and methodological grounding of the HHM as a decision making tool (see Haimes, 1981, Haimes, Tarvainen, Shima, & Thadathil, 1990); rather, we focus on the utility of the HHM framework as a mechanism for the assessment of risk associated with the software acquisition process.

HHM for Software Acquisition Risk Identification

HHM has been used successfully for risk identification in a wide variety of applications (Haimes et al., 1994). Using the software acquisition HHM model (Figure 2), a systemic exploration of the software acquisition risk can be conducted through the multiple visions of the model. As an

example, from the *program consequence* perspective, the software acquisition process may be divided into three consequence areas: technical, cost, and schedule.

1. *Technical*. In a software context, software technical consequences are concerned with the quality, precision, accuracy, and performance over time of the software.

2. *Cost*. Refers to the programmed and unexpected expenditures for procuring the software system, along with labor, capital, and other non-monetary costs.

3. *Schedule*. Concerns the establishment of, adherence to, and changes of a temporal development plan from which systems integration schedules and operational deployment schedules are based.

Figure 6 depicts one such representation from the perspective of *program consequence* HHS, focusing on the cost risks of the software acquisition effort—in particular, the cost risks associated with each community (user, customer, contractor, and technology).

Using the program consequence perspective as the primary vision, one may then examine all such consequences that emerge from the participant communities (e.g., What effects will the customer community have on the schedule?). Robust application of the HHM involves a thorough examination from multiple combinations of perspectives the consequences and factors associated with the software acquisition effort. Such a comprehensive analysis produces a wealth of understanding of the strengths and weaknesses associated with an acquisition effort, provides a framework for devising a management plan to deal with the identified shortcomings, and maintains the holistic perspective critical to program success.

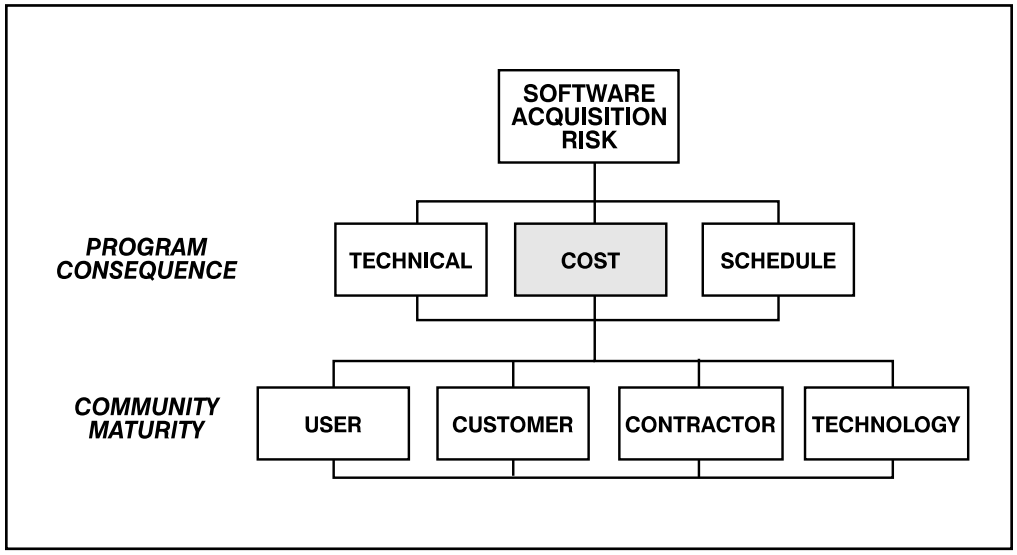


Figure 6. Program Consequence Submodel: Cost Focus

Software Acquisition Process Models

Each decomposition of the HHM depicted in Figure 2 provides a unique perspective for evaluating and describing software acquisition. In this section, we consider the *process* decomposition of the software acquisition HHM. Developing the process HHS allows one to focus on identifying, understanding, and modeling the progression of activities and interrelations associated with the software acquisition process.

Software Process Modeling

Discussion of the software process in the literature focuses primarily on software development processes and on the contractor's role in those processes. Often referred to as the software life cycle, the software development process is the collection of activities that begins with the identification of a need and concludes with the retirement of the software product that satisfies the need. Traditionally, the software process has been described in terms of the "waterfall model" (Boehm, 1976). In this model (which is an adaptation from the hardware development process model) there is a basic forward flow, or progression of activities and events. While the model presents a logical, organized approach, its inflexibility in adapting to the unique requirements of modern software development has led many to believe that this model is discredited (Blum, 1992). More recent representations of the software development process have included iterative, prototyping activities. For example, Boehm's spiral model (1988) consists of a series of learning cycles, with each iteration including the phases of identification, evaluation, planning, and testing. With each successive iteration, greater insight is gained, and system development is improved.

Most current software acquisition processes still follow a waterfall approach. One major initiative associated with current acquisition reform initiatives is modifying the existing process to better meet the unique requirements of software development. Improvements allowing for adaptive design, prototyping, and other iterative development approaches are being recommended (Scientific Advisory Board, 1994).

As software acquisition encompasses a range of activities and concerns far beyond that of merely developing a product, existing process models are inadequate for fully describing the software acquisition process. The process models developed in this section are descriptive in nature: they indicate the basic activities, events, interrelations, and functions of the software acquisition process as currently practiced (or as intended to be practiced). Realizing that analyses of the process and process improvement, alone, are not sufficient to accomplish the larger goal of improving software acquisition (Feiler, 1994), these models constitute an initial vehicle for examining the multiplicity of elements associated with software acquisition.

Software acquisition process models provide a representation of the progression of interrelated activities, the interactions of the participant communities, the functions that each participant accomplishes, and the means for analyzing the impact of these actions on the actual software acquisition effort. Each stage of the model consists of several elements or activities that define the principal contribu-

tion of that phase to the overall process. Before developing a fully detailed model, we first examine the essential elements and activities of the software acquisition process. Once these fundamental elements have been identified, we add complexity and detail to the model. Again, the intent is to describe the current process and to work within that context, not to prescribe an ideal process.

Essential Software Acquisition Process Model

Essence refers to the object of concern, or inherent nature of an activity (Brooks, 1987). Using this definition, the essence of the software acquisition process is finding a solution that meets the stated need. By initially focusing on the essence of this process, we will have a context from which to address the more detailed issues that complicate the process' effectiveness.

Abstraction of the software acquisition process leads to the simplified model shown in Figure 7. This model takes the user's perceived real-world need as input and produces a solution to meet that need. The model contains three translation activities, which together represent a transformation from a stated need to a solution:

1. From the user's real-world statement of need to a requirements statement that details an intended solution for the need.

2. From the requirements specification statement to a development and specification statement. This statement includes the details of design, process, and evaluation along with the method of selecting a contractor to implement the development plan.

3. From the development statement to an actual system that satisfies the user's real-world need.

The model highlights the understanding each participant community must have of its role in the process. As requirements are developed from input originating in the user's domain, the customer must understand the user's domain and then be

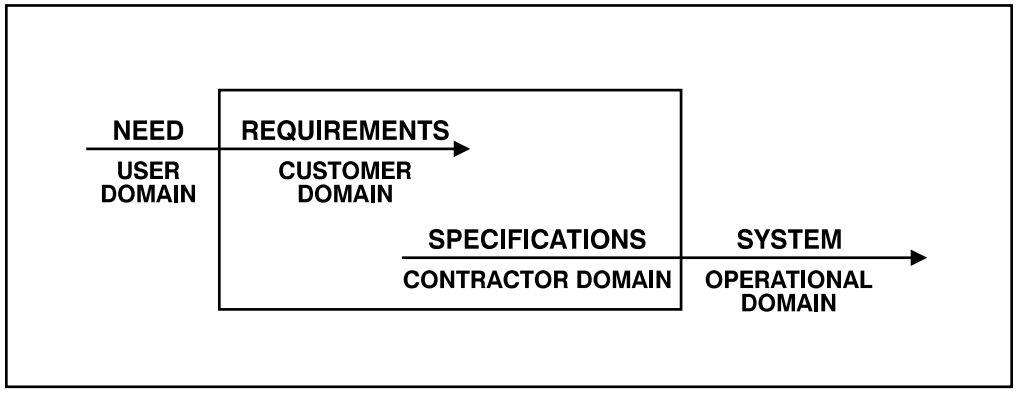


Figure 7. Essential Software Acquisition Process Model, adapted from Blum (1987)

able to generate system requirements that are usable by the contractor. These requirements, however, are not specific enough to provide sufficient detail to fully define the intended system. Therefore, requirements must be translated into more formal, detailed system design and development statements. The contractor's understanding of software tools and environments must also extend to an understanding of the implementation domain—that domain in which the developed system will operate.

The essential model explains why the software acquisition process is so complicated: It requires the coordinated activities of several participant communities, necessitates experience in several domains, and depends on a series of difficult translation activities. As software continues to assume a greater role in modern systems, the difficulties of software acquisition become the overriding difficulties in systems acquisition. Technology is changing so rapidly, and communication between the three communities is getting increasingly complicated, especially because of DoD's demanding operational needs that precipitate the requirement for state-of-the-art systems. These facts constitute the driving force of the software acquisition process.

Detailed Software Acquisition Process Model

Expanding the level of detail included in the essential model leads to the model depicted in Figure 8. The seven stages of the detailed model parallel that depicted

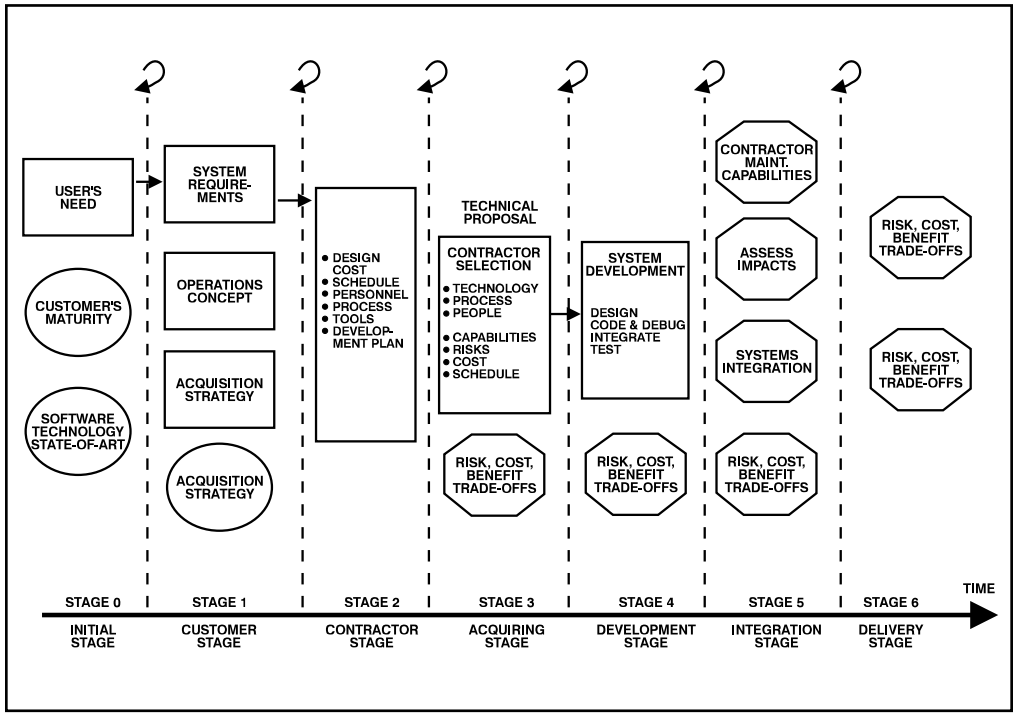


Figure 8. Detailed Software Acquisition Process Model

in Figure 7. Stage 0 corresponds to the input to the model, the need developed by the user; stage 1 parallels requirements specification; stages 2, 3 and 4 are an expanded treatment of the development activity; and stages 5 and 6 constitute the system phase of the previous model.

In Figure 8, boxes represent an activity or function to be accomplished. Circles indicate current information, state of events, or other contributing factors that cannot be manipulated by decisions made in the acquisition effort. Octagons indicate output results that require monitoring and managing and are affected by decisions made by the participants in the acquisition effort. The three arrows indicate the three translation activities. The curved arrows above each stage's boundary indicate the possibility for iteration in the process. While not institutionalized in the current process, reform initiatives and reengineering activities indicate a growing support for an iterative path through the acquisition process.

Stage 0: Initial Stage. This stage is a precursor to the actual acquisition activities; however, the initial actions and abilities that proceed from this stage affect the balance of the acquisition effort. Although this stage involves both the user and customer communities, their actions at this point are independent of one another. The user, based on operational experience and training, develops an operational need, provides a review process to formalize, validate, and prioritize the need, and forwards this need to the appropriate acquisition liaison agency.

Stage 0 also comprises the baseline levels for two key elements: state-of-the-art software technology, and customer maturity. Recent investigation has shown that "application-knowledgeable, technically skilled leaders are the military's limiting resource in acquiring today's computer technology" (SAMERT, 1994).

The continued downsizing of the federal government, and the DoD in particular, further exacerbates this problem. Many highly qualified acquisition officials are taking advantage of incentives for early retirement, realizing that their skills are in great demand outside of government service. Where will DoD's needed technical experience and expertise come from? The essential key to an acquisition program's success is the technical maturity of the customer community: their knowledge of software and software acquisition, the ability to understand and translate user needs, establish requirements, develop and manage contracts, select and monitor appropriate metrics, and select the proper contractor. Hence the need for continued development of a software acquisition maturity model (SAMM) (Sherer & Cooper, 1994) aimed at evaluating a customer organization's maturity level and providing a road map for improving its capability.

Stage 1. Customer Actions. Following the user's development and validation of an operational need, the customer agency then begins the task of managing the acquisition of a software solution. The first translation activity, transforming the operationally-based language of the user's need to the contractual language of systems requirements, is completed at this point.

A requirement is a "function or characteristic of a system that is necessary; the quantifiable and verifiable behaviors that

a system must possess and constraints that a system must work within" (Christel & Kang, 1992). Requirements generally specify "what" the system requires in terms of functions and data, and "how well" the system must perform relative to the goals and objectives of the system (Ashworth, 1989).

The output of the requirements identification activity is a formal statement that captures the full intent of the user community's need and communicates this in appropriate language to the contractor community (in DoD, the result is called a request for proposal [RFP]). In this light, requirements analysis is the bridge between user needs and system specifications from which a solution can be developed (Przemieniecki, 1993). Errors in requirements definition can pass through undetected to later stages of the acquisition process, possibly not realized until a deficiency arises at system implementation. Greater discussion of the art and process of requirements elicitation and development can be found elsewhere (Christel & Kang, 1992; Southwell et al., 1987; Rzepka, 1989; and Fickas & Nagarajan, 1988).

Stage 2. Contractor Actions. While some contractor involvement may be solicited during requirements generation, this stage marks the formal introduction of the contractor to the acquisition process. Candidate contractors conduct the second translation activity associated with the acquisition effort by responding to the requirements specification with their detailed development plan. This plan typically includes a description of the design for the intended system, its size, structure, complexity, and other descriptor information. Statements of technology requirements, development environment, development tools, personnel, management, and other organizational and technical issues are also included along with cost and schedule figures.

While the user and customer may have spent considerable time in accomplishing stages 0 and 1, quite often the contractor completes stage 2 in a matter of weeks. Such a time constraint raises important questions concerning the reliability of the contractor's estimates. For instance, how accurate are a contractor's estimates concerning a project that will require technology beyond current capabilities, relying on the development of yet-to-be technology. Other important questions (Haimes & Chittister, 1993) are: do developers with little experience overestimate or underestimate cost and schedule, and do developers with experience overestimate or underestimate cost and schedule?

Stage 3. Acquiring Stage. Some time before proposals are received, the customer determines the evaluation standards upon which all proposals will be scrutinized and evaluated. Organizational capabilities, performance history, cost estimate practices, as well as metrics for evaluating other performance criteria are considered. Other key areas of interest include the contractor's statements regarding technologies, development processes, and capabilities.

Generally, during contractor selection, the customer is faced with a wealth of information—some pertinent, some not. What is needed is a method for determining what the customer needs to know, a process to synthesize and filter the data, and a structured process for using the information to choose the contractor.

Stage 4. System Development. In this stage, the selected contractor implements the development plan and actually produces the software system. Most likely, instead of a single contractor, a contractor consortium of teams of major companies collaborate in the development of a system. The majority of software acquisition and development research has focused on the activities of this stage. The well-known waterfall models of the software development life cycle (Royce, 1970; Boehm, 1981) are models of the activities of this stage. More recent research on software development are also principally concerned with this stage's events (Boehm, 1988; Sage, 1992; Feiler & Humphrey, 1992; Heineman et al., 1994).

Stage 5. Integration. As system development progresses to an initial operating capability, the contractor, customer, and user coordinate the acceptance testing of the system. The success of this stage hinges on the combined work and decisions that stem from previous stages. Systems integration is the dominant activity here; the components are integrated with other system elements. Increasingly, software is the vehicle for accomplishing systems integration and assuring system success (Chittister & Haimes, 1994b).

At this stage, previously identified technical and nontechnical risks have the greatest likelihood of materializing. These risks may have existed all along, and come to the surface during the activities of system integration. In order to most effectively plan for and manage the systems integration activity and the risks that may arise, the customer organization must make critical trade offs between costs, benefits, and risks associated with each policy option.

Stage 6. Delivery. Only when at full operating capacity and beyond can the value of risk mitigation and risk prevention efforts be fully realized. Often, selection of a risk mitigation strategy is based on prevention versus correction—that is, a proactive approach versus a "wait and see" approach: "Early defect fixes are typically two orders of magnitude cheaper than late defect fixes, and the early requirements and design defects typically have more serious operational consequences" (DoD, 1991).

Maintenance and modification initiatives with their associated costs and impact on operational capabilities are principal concerns of this stage. This stage is also the link for returning to stage 0 in the next acquisition effort. If the customer (or user, or contractor) captures the knowledge gained through this acquisition experience and uses it to increase the abilities of their organization (e.g., via training or documentation), then this community's maturity level is increased. They will be better prepared for dealing with the next acquisition effort.

Analysis and Extension through Process Models

The process models developed in this section provide a framework for a more detailed understanding of the interrelations and activities associated with the current software acquisition process. Each phase of the process can be explored in greater depth using the HHM model (Figure 2). The process models also provide the construct from which an iterative software acquisition process could be modeled. These models are an effective vehicle for understanding the required interconnections, mechanisms, information, and ac-

tivities that must be included in such an innovative paradigm.

Extended HHM for Coordinated Submodel Solution

This section describes two additional software acquisition hierarchical holographic submodels, *program consequence*, and *community maturity*, and presents an approach to resolving the inherent conflict in the analytic modeling associated with these submodels.

Program Consequence HHS

The program consequence HHS of the software acquisition HHM addresses the need to synthesize vast amounts of information concerning the suitability of a proposed system for meeting an operational need. This includes analysis of the proposed system design, estimates of the design's technical performance, development cost, and schedule estimates, and other relevant factors. The trade offs between performance, cost, schedule, and the risks in each area are considered in this activity.

The multiple factors associated with the program consequence HHS (Figure 9) may be viewed as submodels of this HHS. Each sub-submodel may be quantified and analyzed by way of an appropriate model (analytic or descriptive), with the requirement for an overall resolution at the HHS level.

Software technical performance may be quantified in terms of one of several measurable objectives (e.g., reliability, availability, maintainability) and is best analyzed through fault tree modeling or Markov process modeling (Johnson, 1989; Kanoun et al., 1993; Tai et al., 1993). While several software cost estimation models exist (e.g., Boehm, 1981; Charette, 1989; Pressman, 1987), each generally produces a single-point estimate of the projected development cost. The cost risk-mitigation approach of Haimes and

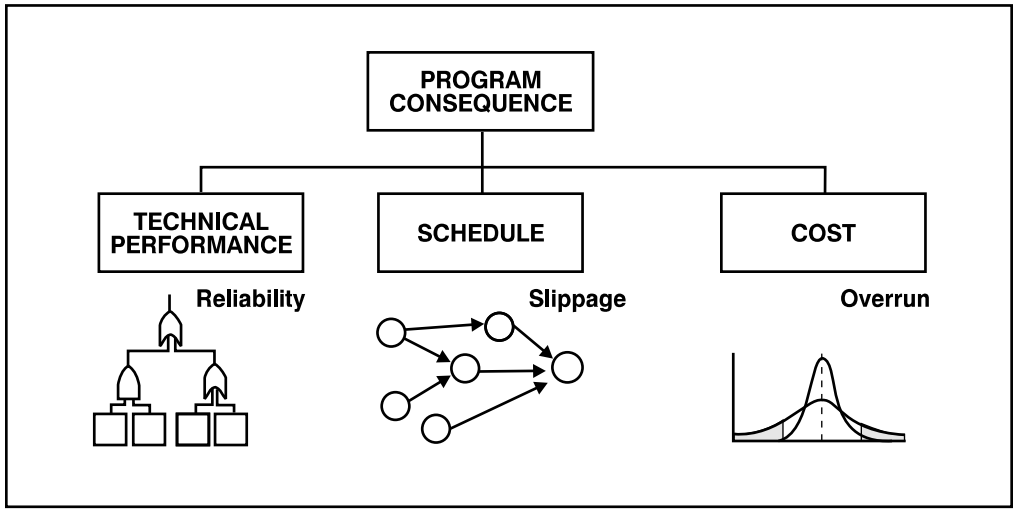


Figure 9. Program Consequence HHS Sub-submodels

Table 2. Multiple Objectives of Acquisition Process Participants

PARTICIPANT	OBJECTIVE	
USER	MAXIMIZE	TECHNICAL PERFORMANCE
	MINIMIZE	DEVELOPMENT TIME
CUSTOMER	MAXIMIZE	TECHNICAL PERFORMANCE
	MAXIMIZE	CUSTOMER–CONTRACTOR RELATION
	MINIMIZE	COST
	MINIMIZE	DEVIATION FROM TIME SCHEDULE
CONTRACTOR	MAXIMIZE	PROFIT
	MAXIMIZE	FUTURE EARNINGS POTENTIAL

Chittister (1993) is an improved approach, employing a probabilistic, extreme event cost analysis methodology that allows incorporation of cost estimation model results. Software project scheduling, most often modeled through PERT (Project Evaluation and Review Technique) or CPM (Critical Path Method) models (Charette, 1989), also has probabilistic extensions (Abdel-Hamid & Madnick 1983; Haimes et al., 1994). While each of the three sub-submodels can be analyzed and solved independently, overlapping objectives and constraints require that a managed, coordinated solution to the overall problem be resolved at the HHS level.

Community Maturity HHS

The *community maturity* HHS captures the competing, yet overlapping objectives of the three participant groups. This HHS can, in some ways, be viewed as an extension of the program consequence HHS. Each participant community is further represented by its own sub-HHS, with only those consequences applicable to each participant group considered in the submodels.

In an acquisition effort the user's primary objective is to meet all of the operational needs. This objective could be stated as acquiring a system that maximizes technical performance. Generally, this is not the only objective for the user. Getting the system as soon as possible—minimizing development time—may also be important. The customer has a similar, but different multiple-objective problem: to minimize cost, maximize technical performance, enforce a contractual time schedule, and maximize contractor–customer communication. The contractor also has a multiple-objective problem: maximize profit and maximize potential for future earnings (additional contracts). An example of these multiple objectives, at least one for each of the acquisition process participants, is represented in Table 2.

Obviously, some of these objectives are competing both within a participant's individual problem (maximize technical performance versus minimize cost) and between participants (maximize contractor's profit

versus minimize customer's cost). A common DoD contracting practice is to fix profit margin; under such restrictions, the only way for a contractor to increase earnings is by spending more, thereby increasing the overall program cost. A contractor can also earn more with each deviation and modification to the contract. The more deviation, the greater the contractor's earnings. These realities are in direct competition to the customer's goals of minimizing cost and contract deviations. There are overlaps between the objectives of the three participants, yet there are certain objectives unique to each individual problem. Adequate coordination of these multiobjective problems is the key to a mutually agreeable solution.

As Figure 10 shows, analytic methodologies appropriate for analyzing each of the submodel objectives may not necessarily assume the same form. At least one objective, the customer's desire for good relations with the contractor, may be unrepresentable mathematically. Some indication concerning this objective, however, may be analyzed through a classification model based on HHM analysis of the characteristics and capabilities of the particular contractor. Each sub-HHS model is itself a multiple objective model, and the result of the three participant community solutions must be resolved at the overall HHS level.

Similar multiobjective problems can be derived for each HHS of the HHM. For instance, maximizing technical performance may include the subobjectives of maximizing reliability, number of computations per second, system availability, or some other system performance feature. To a certain extent, each of these subobjectives is in conflict with the others—for example, a system designed for

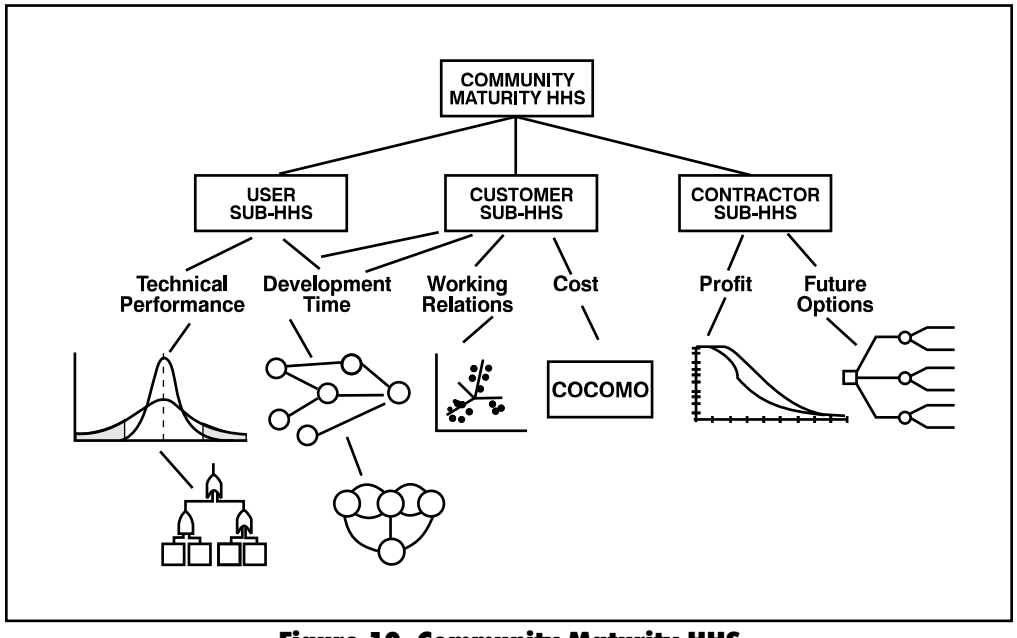


Figure 10. Community Maturity HHS

computational speed may not be extremely reliable. Including cost minimization as a subobjective may introduce additional conflicts (e.g., a highly reliable system is generally not the low-cost option). The practical reality of software acquisition demonstrates that it must be described in terms of multiple, multiobjective problems that are conflicting, possible overlapping, and exhibit a hierarchical structure.

HHM: Model Management for Submodel Conflict Resolution

The quantitative framework described in this paper is founded on the synergistic coupling of the HHM and process models, and the incorporation of other appropriate models, methods, and tools to effectively analyze and support decision making throughout the acquisition process. Figure 1 shows a representation of this framework. While each of the many methods and tools has been designed for its own unique purpose, each may give a greater contribution when coordinated with the results of other methods. A holistic vision ensures that methodologies are not employed for their own sake (sublevel optimization), but that each contributes to the overall system goals and objectives. In this manner, we achieve analytic progression through the complex acquisition process.

The field of model management, a growing area of study, recognizes that complex problems rarely can be solved by a single model encompassing all problem aspects. Solutions to such problems often require the integration of multiple models each addressing a specific aspect of the problem (Mitra & Dutta, 1994). Model management methodologies provide approaches for combining several models into an integrated model that is sufficient to solve a given problem (Basu & Blanning, 1994). While several approaches have been proposed (e.g., database management, artificial intelligence, conceptual graphs), the most promising and easily implemented are those of the graphical approach. These methods provide a framework for model composition by identifying models that may be combined into a composite model (Muhanna & Pick, 1994). The HHM can appropriately be considered as a model management methodology; resolution of the higher level model requires the coordinated solution of multiple submodels. HHM's unique handling of overlapping objectives and program constraints is particularly desirable. While the original HHM assumes that each submodel has a mathematical programming formulation, relaxing this requirement provides a means for resolving the more realistic problem where submodels have diverse analytic constructs.

HHM provides the framework for representing the software acquisition submodels in terms of hierarchical multiobjective decision models (MODM). The MODM approach provides a context for a "win-win-win" environment, as a solution that is mutually acceptable to all three participants would be found in the set of nondominated solutions to the coordinated multiobjective problem (Chankong & Haimes, 1983). The multiobjective modeling approach also provides a structure for resolving competition between issues, such as the trade off between performance and cost, or between schedule versus technology.

Formulation of a hierarchical multiobjective problem, taken from the frame-

work of the HHM, implies the formal evaluation of model elements and data requirements (random variables, decision variables, state variables, functional relationships, etc.). Such a comprehensive effort not only establishes an overall analytical framework for reaching an agreeable solution, but also provides greater insight and understanding as to the interrelationships and structure of the software acquisition process.

When a purely analytic approach for resolving HHS conflicts is not possible, it may be necessary to consider conflict resolution strategies (Fraser & Hipel, 1984; Raiffa, 1982), trade off methodologies (Chankong & Haimes, 1983), and negotiation strategies (Nierenberg, 1978). The *community maturity* and *program consequence* problems described above are not ones of simple multiple-objective resolution due to the overlapping and coordination inherent in the HHM/HHS structure—therefore application of conflict resolution strategies must consider the unique characteristics of hierarchical, overlapping, and yet conflicting problems. Other possible approaches to be explored include some variation of a weighting scheme (Chankong & Haimes, 1983), where each HHS or even each objective is assigned a weight—directly or by a pairwise comparison—to determine preference for each objective and submodel. The analytic hierarchy process (AHP) (Saaty, 1990) may be useful for such an approach. Other trade off methods, such as the surrogate worth trade off (SWT) method (Chankong & Haimes, 1983) may also prove useful.

Conclusion

The holistic approach to software acquisition management presented in this paper provides a theoretical, as well as methodological approach, for a maturing software acquisition community. This approach compliments the maturity progression described in the SAMM. Figure 11 summarizes the relationship between management maturity and quantitative risk management methods. With the SAMM, progression to higher maturity levels assumes the continuation of all lower level activities and methods, while adding new functions and processes. Similarly, with the quantitative risk management framework, holistic models build on the descriptive foundation of the process models; used together they provide even greater information and insight. As with the SAMM, lower level organizations may make use of some of the higher level methods, but have not employed all the functions required to fully progress to that higher level. The focus of each analytic method parallels the focus of the related management level.

Here we've detailed a holistic approach to the analysis of the software acquisition process and to the development of a quantitative management framework needed for the maturing of the software acquisition customer community. This framework is founded on the multiple visions of the HHM and the temporal progression of activities captured in the process models. It provides the means for incorporating appropriate analytic models and methodologies for a systemic approach to quantitative management of software acquisition. The competing interests of the participants, con-

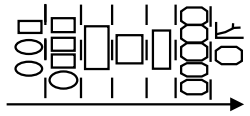
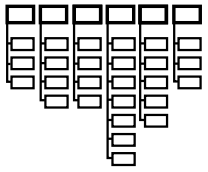
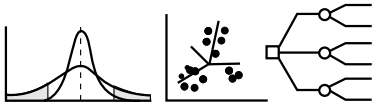
SOFTWARE ACQUISITION MATURITY MODEL		QUANTITATIVE RISK MANAGEMENT FRAMEWORK	
MATURITY LEVEL	FOCUS	ANALYTIC METHOD	METHOD FOCUS
2 - ORGANIZED	CONTRACT MANAGEMENT	PROCESS MODEL 	DESCRIPTIVE
3 - DEFINED	INTEGRATED PROJECT MANAGEMENT	HOLISTIC MODEL 	IDENTIFICATION/ QUANTIFICATION
4 - CONTROLLED	QUANTITATIVE MANAGEMENT	QUANTITATIVE MODELS 	QUANTITATIVE ANALYSIS
5 - OPTIMIZING	PROCESS OPTIMIZATION	MULTI-OBJECTIVE MODELS USER: MAX F_1 MIN F_2 CUSTOMER: MAX F_3 MIN F_4 MIN F_5	OPTIMIZATION

Figure 11. SAMM and Quantitative Management Framework Comparisons

flicting performance measures, and uncertain system requirements make a multiple-objective approach for analyzing and resolving these conflicts appropriate.

ACKNOWLEDGMENT

We would like to thank Archie Andrews, Lisa Brownsword, Mike DeRiso, Dave Gulch and Fred Hueber for their valuable comments and suggestions.

REFERENCES

Abdel-Hamid, T. K., & Madnick, S. E. (1991). Software project dynamics: An integrated approach. Englewood Cliffs, NJ: Prentice-Hall.

Ashworth, C. M. (1989). Using SSADM to specify requirements. IEEE Colloquium on Requirements Capture and Specification for Critical Systems, 138 (November).

Austin, R. D., & Paulish, D. J. (1993). A survey of commonly applied methods for software process improvement (Tech. Rep. CMU/SEI-93-TR-27). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Baker, E. R., Cooper, L., Corson, B. A., & Stevens, A. E. (1994). Software acquisition management maturity model (SAM3). Program Manager (July-August), 43-49.

Basu, A., & Blanning, R. W. (1994). Model integration using metagraphs. Information Systems Research, 5(3), 195-218.

Blum, B. I. (1987). Evaluating alternative paradigms: A case study. Large Scale Systems, 12(3), 189-199.

Blum, B. I. (1992). Software engineering: A holistic view. New York: Oxford University Press.

Boehm, B. W. (YEAR?) Software engineering. IEEE Transactions on Computers, 25, 1226-1241.

Boehm, B. W. (1981). Software engineering economics. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B. W. (1984). Verifying and validating software requirements and design specification. Software, (January), 75-88.

Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61-72.

Brooks, F. P. Jr. (1987). No silver bullet: Essence and accidents of software engineering. IEEE Computer, (April), 10-19.

Carr, M. J., Konda, S.L., Monarch, I., Ulrich, F. C., & Walker, C. F. (1993). Taxonomy-based risk identification (Tech. Rep. CMU/SEI-93-TR-6). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Chankong, V., & Haimes, Y. Y. (1983). Multiobjective decision making. New York: North-Holland.

Charette, R. N. (1989). Software engineering risk analysis and management. New York: McGraw-Hill.

Chittister, C. & Haimes, Y. Y. (1994). Assessment and management of software technical risk. IEEE Transactions on Systems, Man, and Cybernetics, SMC-24(2), (February).

Chittister, C., & Haimes, Y. Y. (1994, August). Systems integration via software risk management. Oral presentation, Software Engineering Symposium, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Christel, M. G., & Kang, K. C. (1992). Issues in requirements elicitation (Tech. Rep. CMU/SEI-92-TR-12). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Deutch, J. M. (Feb. 24, 1994). Statement on the Federal Acquisition Strategy Act of 1994. Testimony before a Joint Hearing of the Senate Committee on Armed Services and Senate Committee on Governmental Affairs, Washington, DC.

Department of Defense. (1991). Software technology plan: Volume II plan of action. Washington, DC: DoD.

Feiler, P. H., & Humphrey, W. (1992). Software process development and enactment: Concepts and definitions (Tech. Rep. CMU/SEI-92-TR-4). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Feiler, P.H. (1994, August). Disciplined engineering of software intensive systems. Oral presentation (slides) at the Software Engineering Symposium, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Fickas, S., & Nagarajan, P. (1988). Critiquing software specifications. IEEE Software, 5 (November).

Fraser, N., & Hipel, K. W. (1984). Conflict analysis models and resolutions. New York: North Holland.

General Accounting Office. (1990, April). DoD embedded computers. Washington, DC: Government Printing Office.

General Accounting Office. (1992, May). Embedded computer systems: Significant software problems on C-17 must be addressed. Washington, DC: Government Printing Office.

General Accounting Office. (1992, December). Weapons acquisition, a rare opportunity for lasting change (GAO/NSIAD-93-15). Washington, DC: Government Printing Office.

Haimes, Y. Y. (1981). Hierarchical holographic modeling. IEEE Transactions on Systems, Man, and Cybernetics, SMC-11(9), 606-617.

Haimes, Y. Y., Li, D., & Tulsiani, V. (1990). Multiobjective decision-tree analysis. Risk Analysis, 10(1), 111-129.

Haimes, Y. Y., Tarvainen, K., Shima, T., & Thadathil, J. (1990). Hierarchical multiobjective analysis of large-scale systems. New York: Hemisphere.

Haimes, Y. Y., & Chittister, C. An acquisition process for the management of risks of cost overrun and time delay associated with software development (Tech. Rep. CMU/SEI-93-TR-28). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Haimes, Y. Y., Li, D., Lambert, J. H., Schoof, R. M., Eisle, S., & Schneider, C. (1994). Improving risk management for the criminal justice information services (FBI) (Tech. Rep.). Charlottesville: University of Virginia, Center for Risk Management of Engineering Systems.

Hall, A. D. III. (1989). Metasystems methodology; A new synthesis and unification. New York: Pergamon Press.

Heineman, G. T., Botsford, J. E., Caldiera, G., Kaiser, G. E., Kellner, M. I., & Madhavji, N. H. (1994). Emerging technologies that support a software process life cycle. IBM Systems Journal, 33(3), 501-529.

Higuera, R. P., Gluch, D. P., Dorofee, A. J., Murphy, R., Walker, J. A., & Williams, R. C. (1994). An Introduction to Team Risk Management (Version 1.0) [Computer software] (CMU/SEI-94-SR-1). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Humphrey, W. S., & Kellner, M. I. (1989, May). Software process modeling: principles of entity process models. Proceedings of the 11th International Conference on Software Engineering. Washington, DC: IEEE Computer Society Press. 175-188.

Johnson, B. W. (1989). Design and analysis of fault tolerant digital systems. Reading, MA: Addison-Wesley.

Kanoun, K., Kaaniche, M., Beounes, C., Laprie, J-C., & Arlat, J. (1993). Reliability growth of fault-tolerant software. IEEE Trans. on Reliability, 42(2, June), 205-219.

Kellner, M. I. (1991). Software process modeling support for management planning and control. In M. Dowson (Ed.), Proceedings of the 1st International Conference on the Software Process: Manufacturing Complex Systems (pp. 8-28). Washington, DC: IEEE Computer Society Press.

Mitra, S., & Dutta, A. (1994). Integrating optimization models and human expertise in decision-support tools. Expert Systems with Applications, 7(1), (January), 93-107.

Muhanna, W. A., & Pick, R. A. (1994). Metamodeling concepts and tools for model management. Management Science, 40(9), (September), 1093-1123.

Neirenberg, G. (1978). The art of negotiating. New York: Negotiation Institute.

Office of the President of the United States of America. (1993). Technology for America's economic growth, a new direction to build economic strength. Washington, DC.

Pages, E. R. (1994, March 10). Testimony before a Joint Hearing of the Senate Committee on Governmental Affairs. Washington, DC: Government Printing Office.

Paulk, M.C., Curtis, B., Chrissis, M. B., Averill, E., Bamberger, J., Kasse, T., Konrad, M., Perdue, J., Weber, C., & Withey, J. (1992). The capability maturity model for software. 1992 SEI technical review. Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Paulk, M.C., Weber, C. V., Garcia, S. M., Chrissis, M. B., & Bush, M. (1993). Key practices of the capability maturity model, version 1.1 (Tech. Rep. CMU/SEI-93-TR-25). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Perry, W. J. (1994, Feb. 9). Acquisition reform, a mandate for change. Testimony before a Joint Hearing of the Senate Committee on Armed Services and Senate Committee on Governmental Affairs. Washington, DC: Government Printing Office.

Pressman, R. S. (1987). Software engineering: A practitioner's approach (2nd ed.). New York: McGraw-Hill.

Przemieniecvki, J. S. (1993). Acquisition of defense systems. Washington, DC: AIAA.

Raiffa, H. (1982). The art and science of negotiation. Cambridge, MA: Belknap Press of Harvard University Press.

Royce, B. W. (1970). Managing the development of large software systems. IEEE WESCON, 1(9).

Rzepka, W. E. (1989). A requirements engineering testbed: Concepts, status, and first results. In B. D. Shiver (Ed.) Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, Washington, DC: IEEE Computer Society.

Saaty, T. L. (1990). Multicriteria decision making—the analytic hierarchy process. Pittsburgh, PA: RWS Publications.

Scientific Advisory Board, U.S. Air Force. (1994, February). Information architectures that enhance operational capability in peacetime and wartime. Washington, DC: Department of the Air Force, AF/SB.

Sage, A. B. (1992). Systems Engineering. New York: John Wiley.

Software Acquisition Management Education Review Team (SAMERT). (1994, March). Report of the SAMERT. Washington, DC: U.S. DoD, Office of the Undersecretary of Defense (Acquisition Reform).

Sherer, S. W., & Cooper, J. (1994, September). Software acquisition maturity model (SAMM) draft version 4.0 (Special Rep.). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Sisti, F. J., & Joseph, S. (1994). Software risk evaluation method (version 0.2) (Report CMU/SEI-94-SREv0.2). Pittsburgh, PA: Carnegie Mellon University, Software Research Institute.

Southwell, K., Clarke, B. A., Andrews, B., Ashworth, C., Norris, M., & Patel, V. (1987). Requirements definition and design. The STARTS Guide: Vol. 1 (2nd ed.). Washington, DC: National Computing Center.

Tai, A. T., Meyer, J. F., & Algirdas, A. (1993). Performability enhancement of fault tolerant software. IEEE Trans on Reliability, 42(2), (June).

Yeh, R. T., Naumann, D. A., Mittermeir, R. T., Schielmmer, R. A., Gilmore, W. S., Sumral, G. E., & Lebaron, J. T. (1991). A commonsense management model. IEEE Software, 8, 23-33.