Systems Integration via Software Risk Management

Clyde G. Chittister, *Member, IEEE,* and Yacov Y. Haimes, *Fellow, IEEE*

Abstract—**This paper addresses an evolutionary process currently taking place in software engineering: the shift from hardware to software, where the role of software engineering is increasing and is becoming more central in systems integration. This shift also markedly affects the sources of risk that are introduced throughout the life cycle of a system's development—its requirements, specifications, architecture, process, testing, and end product.**

Risk is commonly defined as a measure of the probability and severity of adverse effects [19]. Software technical risk is defined as a measure of the probability and severity of adverse effects inherent in the development of software.

Consequently, risk assessment and management, as a process, will more and more assume the role of an overall cross-functional system integration agent. Evaluating the changes that ought to take place in response to this shift in the overall pattern leads to two challenges. One is the need to reassess the role of a new breed of software systems engineers/systems integrators. The other is the need to develop new and appropriate metrics for measuring software technical risk.

Effective systems integration necessitates that all functions, aspects, and components of the system must be accounted for along with an assessment of most risks associated with the system. Furthermore, for software-intensive systems, systems integration is not only the integration of components, but is also an understanding of the functionality that emerges from the integration. Indeed, when two or more software components are integrated, they often deliver more than the sum of what each was intended to deliver; this integration adds synergy and enhances functionality. In particular, the thesis advanced in this paper is that the process of risk assessment and management is an imperative requirement for successful systems integration; this is especially true for software-intensive systems.

In addition, this paper advances the premise that the process of risk assessment and management is also the *sine qua non* requirement for ensuring against unwarranted time delay in a project's completion schedule, cost overrun, and failure to meet performance criteria. To achieve the aspired goal of systems integration a hierarchical holographic modeling (HHM) framework, which builds on previous works of the authors, has been developed. This HHM framework constitutes seven major considerations, perspectives, venues, or decompositions, each of which identifies the sources of risk in systems integration from a different, albeit with some overlap, viewpoint: *software development, temporal perspective, leadership, the environment, the acquisition process, quality, and technology.*

I. The Role of Risk Assessment and Management in Systems Integration

THIS paper addresses an evolutionary process currently taking place in engineering systems: the shift from hardware to software, where the role of software engineering is increasing and is becoming more central in systems integration. This shift also markedly affects the sources of risk that are introduced throughout the system's development—its requirements, specifications, architecture, process, integration and testing, and end product.

The life-cycle developmental process of software encompasses several interconnected functions. The customer's interpretation of needs and desires as perceived by the developer is established through a set of *requirements*. To build the product, the developer projects and transforms these perceptions into step-by-step detailed instructions, known as *specifications*. (i.e., the developer does not build a product directly on the basis of the requirements per se.) These specifications are further translated into a cognitive road map, commonly known as *architecture*. The resulting architecture and specifications are then used by the developer as "blue prints" for the ultimate realization of the product; this is accomplished through the *developmental process*. The developmental process itself involves: 1) appropriate use of technology and know-how, 2) appropriate allocation of human and other resources, and 3) availability of an organization that is at the right maturity level and is endowed by and has in place an appropriate organizational and managerial infrastructure. *Integration and testing* is an important function for any product and is particularly critical for software. The *end product*, however, may or may not represent a true realization of the actual customer's needs and desires as were originally conceived by the developer.

The software engineering community accomplishes the software engineering life cycle via different models: two of the most common are the "waterfall" model consisting of system feasibility, software plans and requirements, product design, detailed design, coding, integration, implementation, and operations and maintenance [3], and the spiral model [2]. Imbedded within these phases are verification (e.g., are we building the product right?), validation (e.g., are we building the right product?), and configuration management (e.g., are we properly managing change throughout the developmental process of the system?). Sage and Palmer [23] define software configuration management as "the process of identification of software system configuration at specific points in time along the life cycle such as to enable maintenance of the traceability and integrity of the software configuration throughout the development life cycle." They also view the life cycle of software development from systems engineering perspectives and offer seven phases: requirements definitions, and specifications; feasibility analysis; program and project plan; logical and physical design; design implementation and system tests; operational deployment; and review, evaluation, and retirement.

Manuscript received February 4, 1995; revised June 18, 1995, and September 3, 1995. This work was supported by the United States Department of Defense (DoD).

C. G. Chittister is with the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Y. Y. Haimes is with the Center for Risk Management, University of Virginia, Charlottesville, VA 22903 USA (e-mail: yyh4f@virginia.edu).

Publisher Item Identifier S 1083-4427(96)04759-5.

From the perspectives of the temporal process of software development, these functions or phases do not exist in isolation from each other; rather, they are connected and exert influence on each other. This inter-and intra-functional connectedness yields three sources of increased risk: 1) internal sources (originating from within a developmental function, 2) interactive sources (originating from interactions among the various functions), and 3) external sources (originating from outside the project). The assessment and management of these three sources of software technical risk should not only result in a better understanding of their connectedness, but can also provide a systemic framework for directing, streamlining, coordinating, and synergizing these functional relationships.

Risk is defined here as a measure of the probability and severity of adverse effects. Software technical risk is defined as a measure of the probability and severity of adverse effects inherent in the development of software.

The sources of software failure are numerous [4], [18]. Putnam and Myers [22] identify six major types of what they term "defects" in software: requirements defects, design defects, algorithmic processing defects, interface defects, performance defects, and documentation defects. Johnson [14] associates possible cause of faults with problems in four areas: 1) specification mistakes (e.g., incorrect algorithms, architectures, or hardware and software design specifications), 2) implementation mistakes (e.g., faults that are introduced through poor design, component selection or construction, or software coding mistakes), 3) component defects (e.g., manufacturing imperfection, random device defects, or component wear-out), and 4) external disturbance (e.g., radiation, electromagnetic interference, operator mistakes, or environmental extremes). In subsequent discussion, we will address in more detail the centrality of assessing and managing software-based defects, and thus the risks associated with them.

In this paper *we define systems integration as the process of amalgamation and coordination among all the couplings and interactions of the system's components so that the entire system can perform its intended functions as a unit.* Indeed, we think of systems integration as a process that is not limited in its occurrence to an "after development," but to a continuous process from cradle to grave.

Note that each subsystem (a system's part or component) can achieve its limited, compartmentalized function; however, only through system integration (i.e., through an orchestrated coordination) can the entire system achieve its intended functions. This necessarily leads to the concept of a system within a system, and federated (coordinated) systems. *In systems federation, the subsystems, may be loosely integrated, may act independently, and are coordinated primarily by humans* (e.g., the pilot changes the speed of an airplane, or readjusts the position of the flaps). In integrated systems, on the other hand, the system acts as a unit and the integrating software performs all the functions without direct human interference. In other words, the more automated the system, the greater is its need for systems integration. Since humans serve as integrating agents, more integration becomes necessary with less human involvement, and the systems integration agent thus becomes increasingly valuable. With automated control (with less human intervention), there is a risk that a problem might not be detected. The centrality of software continues to evolve in systems integration, and with this evolution there is an over-riding need for the development of an effective process for assessing and managing the risks associated with systems integration for software-intensive systems.

Studies conducted on the reliability of software and on its role in overall system operation continuously point to the failure of such systems at the initial operating capability (IOC) phase. (In this paper the following definitions are adopted from Johnson [1983], where failures are caused by errors, which are caused by faults: A *fault* is a physical defect, imperfection, or flaw that occurs within some hardware or software component. A *failure* is the nonperformance of some action that is due or expected.) These failures are some manifestation of either poor systems integration, or of situations where critical sources of risk have not been appropriately identified, assessed, and managed. In fact, all individual sub-components and components can be perceived to be "correctly" designed and constructed as specified, and the system may still fail or not function as intended—integration reveals the bugs that are otherwise not discovered in isolation. This phenomenon is also common in phased or incremental delivery, where functions are added to the product over time and where major design modifications take place between the IOC and the final operating capability (FOC) phase of the system development. These changes may be uncoordinated and thus are basic sources of risk. Furthermore, software engineering is an *intellectually intensive activity*, and is subject to human errors and to personal interpretation by the software engineer of the architecture and of the overall system's requirements. These and other issues explored in this paper are what make integration of software-intensive systems different from the integration of pure hardware systems.

The identification, assessment, and management of all sources of risk in software-intensive systems are a critical requirement for successful systems integration. While assessing and managing the risks associated with software engineering during the entire life cycle of a system are the imperative requirement for successful systems integration, it is the *sine qua non* requirement for protecting against unwarranted time delay in project completion schedule, cost overrun, failure to meet performance criteria, and lack of quality. Of course all this is done in the spirit of team collaboration and close interpersonal communication. Fig. 1 is a conceptual illustration of the life-cycle project cost with and without risk mitigation strategies, given that, at time zero, sources of risk have been identified and assessed. Although risk mitigation is not free, it pays off in later stages as suggested by Fig. 1. Consider, for example, a subsystem deployment that is on the critical path. At time $t = 0$, it was assessed that if "business continues as usual," then the likely time delay may exceed 10 weeks (for a one-year total schedule). In this case management may consider any or all of the following options: add more personnel to the project; authorize more working hours as overtime; introduce new technology; stretch the schedule and delay the completion of select functions; reassign priorities; etc. Given that the cost of risk mitigation

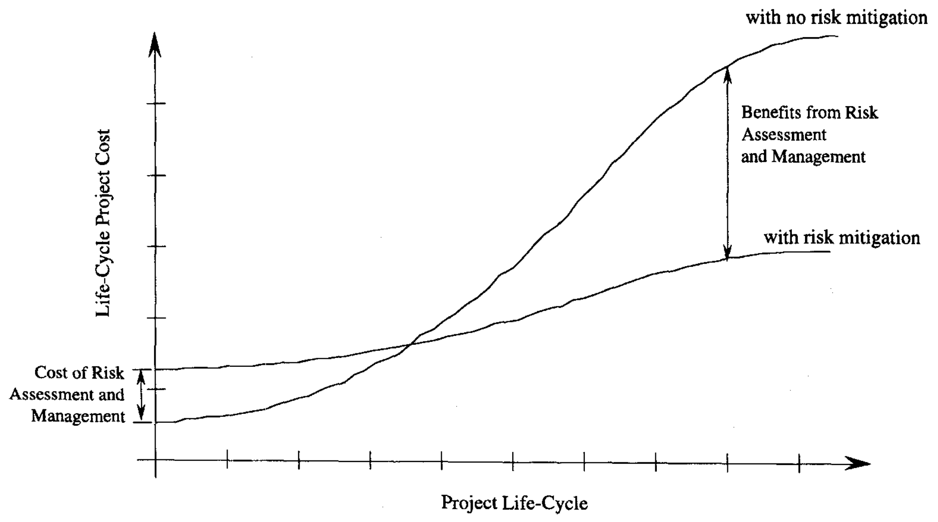

Fig. 1. The cost of no risk mitigation.

can be as high as one hundred times the cost of risk prevention (or even higher in some instances), prudence calls for an early assessment of all major risks and for the initiation of appropriate mitigating actions in order to confront, control, and manage these risks. Clearly, the savings that may accrue as a result of risk prevention and early risk mitigation can be compounded, given the adverse domino effect that a delay in the completion of one subsystem may have on the entire system. Furthermore, the consequences of "doing nothing" may manifest themselves not only in terms of dollars and time but also in a loss of good will, adverse effects on the employees and the culture of the organization as well as on the prospect of securing future contracts. In summary, the timely assessment of sources of risk associated with the life cycle of any project may be essential for successful systems integration.

The shift that favors software engineering should not be thought of as only from hardware to software; rather, it is a shift from a limited, piecemeal vision of software development to a holistic, system-wide vision. Such a visionary approach considers the entire life-cycle path of software development through its six major milestones—requirements, specifications, architecture, process, testing, and product—*where the risk assessment and management process serves as the overall cross-functional systems integration agent that connects and unites these milestones along the life-cycle path.* Boehm's spiral model [2] is a good example of a mechanism with which risk assessment can be performed at each cycle.

Chittister and Haimes [5], [6], who discussed the role of risk assessment and management in software development, introduced and analyzed the role of technical and nontechnical risks in the context of the software developmental life cycle. The increasing role of software as an overall systems integrator, however, is becoming more evident as the shift from hardware to software continues. Furthermore, as the demarcation line between hardware and software becomes fuzzier and fuzzier with the improved technology of microprocessors and target machines, the dominance of software as the overall systems coordinator and integrator becomes more visible. *This necessarily leads to the conclusion that the process of risk assessment and management itself becomes central to effective overall systems integration.* For example, the use of higher level language (e.g., Ada or C^{++}) and the subsequent conversion to machine language and downloading to target machines (e.g., microprocessors) introduce critical sources of risks and uncertainties that must be addressed methodically and systematically.

White *et al.* [28] make a forceful argument that a new cadre of systems engineers with expertise in the systems engineering of computer-based systems is needed to do the systems integration job. They argue that "in too many cases, systems engineers do not understand what information should be provided to Computer-Based Systems (CBS) implementers. Systems engineers provide information in the wrong sequence and with insufficient detail" and that "most systems engineers do not have the skills to understand the consequences, perform the necessary analysis, or establish the required risk-avoidance measures." In addition, many upper level managers are hardware oriented, and younger software-oriented engineers may face a challenging task in convincing hardware-oriented managers of the centrality or efficacy of software engineering. The thesis advanced in this paper not only supports the notion that there is a need to "perform the necessary analysis, and establish the required risk-avoidance measures," but further advocates that this need is so important that the process of risk assessment and management itself has become the quintessential instrument for systems integration. The two seemingly distinct groups mentioned earlier—*engineers as managers of risk, and risk experts as managers of engineering systems*—are a clear manifestation of this emerging need for a new breed of engineers. There is a role separation between the two and a new role for each one. These are engineers who understand how the various components of the system interact with each other and who also have the tools to investigate what can go wrong and what strategies to follow to mitigate potential risks. Today, computer-based systems

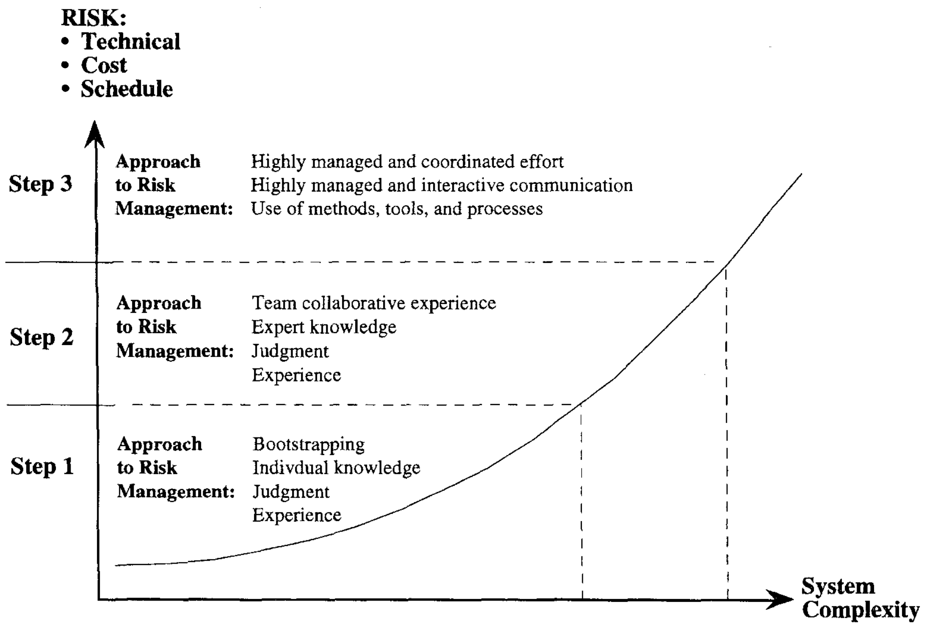
RISK:
• Technical
• Cost
• Schedule
Step 3
Approach to Risk Management:
Highly managed and coordinated effort
Highly managed and interactive communication
Use of methods, tools, and processes
Step 2
Approach to Risk Management:
Team collaborative experience
Expert knowledge
Judgment
Experience
Step 1
Approach to Risk Management:
Bootstrapping
Indivdual knowledge
Judgment
Experience
System Complexity

Fig. 2. The need to manage risk increases with system complexity.

can be so complex that the systems engineers cannot rely on their expertise alone and must seek the cooperation of other disciplines. This also means that systems engineers must empower other engineers in the team to do the risk analysis and the systems integration. *In particular, to be a good software engineer or a good systems engineer/integrator, one must also be a good risk analyst.*

II. Shift from Hardware to Software

As a system becomes more complex, the risks associated with its development increase. At the lowest level (Step 1) the individual (manager or engineer) can assess and manage the risks based on personal knowledge and experience (see Fig. 2). As complexity increases the individual's span of knowledge and experience soon becomes inadequate; the next level (Step 2) is to employ outside expertise to gain more specific knowledge and experience. However, as a system eventually becomes even more complex (Step 3), interaction and communication among its engineers, managers, and other experts must be facilitated and managed in a systemic manner [5], [6]. At this step a systematic and structured set of processes, methods, and tools for assessing and managing risks becomes imperative. Without such a framework people cannot effectively work as a team to manage the risks, and in fact they become part of the sources of the system's risk. In many cases this is the situation today in software-intensive systems. The systems engineer is often not intimately familiar with software and is lacking a process for effectively using his or her software expertise to assess and manage software risks. Since software is playing a more central role in providing the infrastructure for systems integration, software itself is increasingly becoming a source of risk for other components of the system.

This lack of process, methods, and tools and the lack of systems engineers with software knowledge have led some software engineers to assume some of the roles of systems integration and management.

Two recent publications have addressed the evolving role of software engineers as systems integrators. The book edited by Thome [27] focuses on computer-based systems engineering (CBSE), and the paper by White *et al.* [28] summarizes some of the work conducted by the IEEE Computer Society Task Force on Systems Engineering of Computer-Based Systems (ECBS). Both Thome [27] and White *et al.* [28] make important assertions and support the following statement by Lavi *et al.* [16]: "Currently, no engineering discipline provides the knowledge base for the necessary trade-off studies concerning software, hardware, and communication components; a new discipline is needed at the systems engineering level." This sentiment is echoed by Chittister and Haimes [5]:

> The complexity of the goods and services delivered during the last two decades or so has led to extensive sub-specializations in many fields, most notably in engineering. Although this sub-specialization stems from our need for detailed expertise in narrow fields, it has brought with it a parochial and often limited vision of broader, overall system perspectives. Indeed, few engineers have either the opportunity or the expertise to appreciate their own contribution (within a project) to the entire system.

This evolution in software engineering—a shift from hardware-based decisionmaking in engineering to software-based decisionmaking—is further explored in a subsequent paper by Chittister and Haimes [6]:

> The increased influence of software in decisionmaking has introduced a new dimension to the way business is done in engineering quarters: many of what used

to be engineering decisions have been or soon will be transferred and transformed, albeit in a limited and controlled manner, to the *software* function. This powershift in software functionality, the explicit responsibility and accountability of software engineers, and the expertise required of technical professionals on the job have interesting *manifestations, implications*, and *challenges to adapt to new realities and to change*—all of which affect the assessment and management of risk associated with software development.

White *et al.* [28] focus their work on identifying the emerging gap in the engineering of computer-based systems discipline. In distinguishing between software engineers as managers of risk and risk experts as managers of engineering systems, Chittister and Haimes [6] state:

> The intricacy and complexity of the risk assessment and management process (when applied to complex engineering systems) and the need for quantitative analysis (which requires knowledge in probability and statistics) have contributed to the emergence of the subspecialization of risk management in engineering. Thus, seeds for two seemingly distinct groups—*engineers as managers of risk*, and *risk experts as managers of engineering systems*—have been sown. In a parallel way, one may trace the distinction between a) the "engineer" as a technical expert, one primarily concerned with the technical aspects of a project and to a lesser degree with managerial issues, and b) the "manager," one primarily concerned with management (in the broader and more encompassing sense of the term) and to a lesser degree with technical aspects. Here again, on the one hand the "*engineer*" *(as a local manager)* and the "*manager*" (*as a more global manager with a broader vision and perspective*) share responsibilities, tools, and methodologies, yet at the same time each performs distinct functions, matures in different professional cultures, often uses a different jargon, and communicates with a different language. Understanding this emerging powershift surrounding the three entities—software engineering, management, and risk analysis—is at the heart of understanding the emergence of software technical risk management.

Software engineers have been increasingly assuming the bridging role between the customer and the product. The degree of *closeness* between the customer's needs and desires and the attributes and quality of the product (that were supposed to meet these needs and desires) depends to a large extent on the correctness of the intermediate transformation stage from requirements to product, and especially, on the detailed requirements that are given to the software designer. And this intermediate transformation stage is now more and more being performed by software engineers. It is here where risk assessment and management can and should play the central role of ensuring that this degree of closeness is acceptable to the *customer*. By its systemic approach in assessing all conceivable risks, the risk assessment process seeks answers to the following three questions [15]:

- What can go wrong?
- What is the likelihood that it would go wrong?
- What might be the consequences?

Clearly, the complex cognitive path (that involves many individuals) of understanding and then translating the customer's needs and desires through *requirements, specifications, architecture*, and *process* has an inherent likelihood of being diverted from its intended path into a chaotic one. This likelihood, however, can be markedly reduced if the progress along this path is continuously subjected to the scrutiny of the risk assessment process and ultimately to the prevention, abatement, and control of such risks, i.e., through risk management.

III. Software as a Systems Integrator

The displacement of hardware by software is evident. Purely mechanical and electrical devices are continuously being replaced by software-based systems, the operation of parking meters and public telephones with software-driven electronic cards is a case in point. The functioning of software as a systems integrator, i.e., as a coordinator of all a system's activities without which the system will not operate or will not function correctly or effectively, adds a new dimension to software risk management. In fact, software can assume the control role in a system and provide a mechanism that drives all of the elements to work as a unit. Thus, the emerging dominance of software as a systems integrator, whereby the various system components are organized, directed, and controlled to perform a specific set of functions will necessarily bring with it new sources of risk of failure of the system. As software assumes the role of systems integrator, risk assessment should provide a systemic approach to identifying, quantifying, and evaluating most, if not all, sources of software risk—both technical and nontechnical risks—and risk management will ensure the reliable performance of software and thus that of the overall integrated system. (Software technical risk was defined earlier. *Software nontechnical risk is defined as a measure of the probability and severity of adverse effects that are inherent in the development of software and are associated with the programmatic aspects in the development process.*) White *et al.* [28] also argue for the need for systems integration: "For example, we do not know how to monitor subsystem design decisions to ensure that they are not in conflict with system-level design decisions."

As a software-intensive system's complexity increases, it becomes more difficult to comprehend all the positive (synergistic) and negative (adverse) interactions of its components. Furthermore, it also becomes more difficult to plan for the integration, testing, and evaluation of the system. The difficulties transcend a) grouping and ordering of components for integration purposes; b) selecting the type, sequence, and methods of testing; and c) determining the best method and execution of regression testing as new components are added. Thus, it is important that the systems integrator be able to answer not only the set of questions posed by Kaplan and Garrick [15], but also the set of questions posed by Haimes [1992]: a) What can go wrong?, b) What is the likelihood that it would go wrong?, c) What are the consequences?, d) What can be done?, e) What options are available and what are the costs, benefits, and risks associated with these tradeoffs?, and f) What are the impacts of current policy decisions on future options? Furthermore, these systems integrators should address these questions from the perspective of decisionmaking under risk and uncertainty and with the appropriate tools, methodologies, and practices available in the field of risk-based decisionmaking.

In software-intensive systems as well as in many other systems, the systems integration process for delivery may be executed in a "big bang" or a single act, with no intention of adding further functionality in the future, or the integration may be performed in an incremental fashion. Each case has its own, albeit partially overlapping, sources of risk. The systems integrator must be able to understand these two sources of risk and select, in each case, the appropriate and most effective risk-mitigating integration strategy. Since the systems integration process is commonly performed by more than one person, it is imperative that the systems integration process itself facilitates effective communication within the team through a common set of risk-based decisionmaking methods and tools.

IV. The Interface Between Users and Buyers

Software-intensive systems are commonly initiated, commissioned, designed, developed, deployed, and ultimately used and maintained by individuals and organizational entities. The opportunities for miscommunication, adversarial communication, or no communication are unfortunately numerous. It is not always that a win-win situation is advanced and promoted among the involved parties. The common parochial mentality of some individuals or organizations is to care only about their limited role, scope, and responsibility and, with this attitude, to sacrifice opportunities for synergism. This state of mind not only prevents the achievement of higher performance of the parties involved, it is also the source of major risks—both technical and nontechnical risks. Understanding this vital interface among the involved parties in the life-cycle development of the system (and the software that serves as its overall integrator) is a key to effective risk assessment and management.

Furthermore, in large-scale systems, the user and the buyer are often two separate entities. As a matter of fact, there is often a chain of individuals and organizations that separate the buyer (e.g., the Department of Defense's Acquisition Unit) and the ultimate user (e.g., a pilot, a submariner, or a tank commander). The necessary involvement of these intermediate parties constitutes yet another source of risk and uncertainty that must be understood and addressed. *The fact that software is becoming more and more the hub of the integration process of large-scale systems makes it imperative (in terms of sound management) for software risk to be assessed and managed.* This central role of software engineering in systems integration also implies and requires that the assessment and management of risk associated with each subsystem and its software must also be integrated within the overall system risk management.

V. System Integration: Software Engineering and the Software Engineer/Integrator

Since software engineering is an intellectually intensive activity, where both human intelligence and fallibility play a role, human errors are a major source of both software technical and nontechnical risks. One can identify multidimensional sources of risk within each of the functional perspectives (i.e., requirements, specifications, architecture, etc.) in the software life-cycle developmental process. The customer's perception of product needs and of the utility and efficacy of the final product is one example. Other examples of human errors that constitute potential sources of risk include lack of common technical language, misinterpretation of requirements, wrong perception of what constitutes an acceptable response time, and the level of training and education of the personnel involved. Therefore, the risk of human failure can be of paramount importance throughout the software development life cycle.

We have so far focused our discussion on the emerging role of software as an overall systems integrator. The questions we pose in this section are 1) Who is the Software Engineer/Integrator? and 2) Are the functions performed by the software engineer different from those performed by the software engineering integrator? Our premise is that these roles are still *evolving* as the powershift from hardware to software system centrality evolves [6]. What is clear is that software is continually assuming a more *pivotal* role in systems integration, not because software must play this role, but because software engineering is *de facto* playing this role. This rapid shift in the functionality, responsibility, and influence of the individuals involved in the development of software engineering has created a new paradigm. This is a paradigm where software engineers no longer limit their vision solely to the architectural design of the component; rather, they incorporate within their vision a more holistic and systemic perspective of the overall system's model. This new paradigm is also necessarily forcing the introduction of a new cadre of software engineers who must be familiar with systems modeling and integration, and not limited to software architecture design, algorithms, or coding in higher languages. Indeed, the evolving role of software engineering is also forcing the creation of a software engineer/integrator, but the ultimate attributes and characteristics of the software engineer/integrator have not yet matured. It is worth noting that there is no career profession for software engineers in government as opposed to positions for computer scientists [21]. The powershift apparently has not been institutionally recognized, as yet, by the Federal government.

To gain an insight into this evolving role of the software engineer/integrator, we observe the following sample of activities that are currently being performed by the software engineer/integrator:

- The display format of information and functions of most systems, including all their decisionmaking ramifications, are being determined and orchestrated by the software engineer/integrator, because software is the implementation mechanism. Even though the applications engineer plays a role in coordinating these efforts, many technical decisions, such as performance, error detection, and conflict resolution, are left to the software engineer, who frequently does not have the information necessary to make the proper trade-offs and decisions.

- The control system, which is commonly modeled by the designer, is interpreted (and in some sense redesigned) and implemented by the software engineer/integrator. Although this interpretation, redesign, and implementation of the control system may not affect the *functions* of the control, they certainly do affect the *performance* of the overall system.
- Increasingly, the systems modeler or designer has either to know the capability of the software pertaining to that design (which is rarely the case), or he or she must ask the software engineer whether the implementation of a specific design can be realized through software engineering. In either case, the systems designer is increasingly delegating an integrating role and responsibility to the software engineer.

To further explore the evolving role of the software engineer/integrator, we will use as an example the implementation of the systems requirements of a combat airplane via software engineering:

- Through sensors, the software provides data on threats, and informs a pilot, for example, through the display. This display was, of course, redesigned and coded by the software engineer/integrator. Weapons data and other vital information must also be displayed and communicated, and once again the role of the systems integrator is indispensable in providing the partnership between the hardware and software engineers.
- The airplane may be forced to be in various flight regimes. For example, in air-to-air combat, the priority of the functions performed by the airplane must have been modified from the previous flight regime, and the integration of the various systems components must be translated by the software engineer/integrator into command and control statements. These statements must be in congruence with the designated functions of all involved systems components. Furthermore, this task also requires that the pilot receive on-line feedback on the weapon and enemy status—involving yet another essential systems integration function that should be deployed by the software engineer/integrator. Finally, there are also the default, on-line, pre-programmed decisions that are automatically executed when the situation is too complex or too fast for the pilot to get involved. Although these critical controls are modeled and designed by others, it is the software engineer/integrator who ultimately translates and implements them into the system. For example, in planes such as the F-22, V-22, and other commercial aircraft such as Boeing 777, critical functions can no longer be designed into the hardware because of the implementation of the system software. One might argue that this interfacing role of the software engineer has always existed; however, the latter is functioning more and more as the designer and deployer of vital systems decisions—decisions that not only transcend any single component's functionality, but also involve the entire system.

Thus, the software engineer/integrator is now deciding not only on the implementation and deployment of specific functions of the system, but is also deciding on the functional interface among the system's components and is often making decisions in support of the client/user. Although this is also true for hardware engineers, the shift from hardware to software predominance is resulting in an ever-increasing importance of software engineering within system integration.

VI. Hierarchical Holographic Modeling

Evans [9] argues that a good systems integrator must ask good questions, have a good framework to work with, follows a well-defined process, and have an access and is able to work effectively with a good group of professionals. Indeed, effective systems integration necessitates that all functions, aspects, and components of the system be accounted for along with an assessment of all the associated risks thereof. To be responsive to Evans' vision and to accomplish this risk assessment task, hierarchical holographic modeling (HHM) [11] is adopted in this paper. The effort here builds on systems engineering thinking (see, for example, recent work by Mar [20] and Sage [24], [25]) and on a holistic framework for the assessment and management of risk associated with software development introduced by Chittister and Haimes [5]. In his comprehensive vision, Sage's introspective to the emergence of information technology and software engineering as dominant forces in our technological society, and his vision for the need to develop sound theoretical and methodological formulations upon which to develop procedures to manage technology are in congruence with the HHM philosophy [25]. Fundamentally, HHM is grounded on the premise that large-scale and complex systems, such as software development, should be studied and modeled from more than a single perspective, decomposition, vision, or schema. And, because such complexities cannot be adequately modeled or represented through a planar or a single vision, overlapping among these visions is not only unavoidable, but can be helpful in a holistic appreciation of the interconnectedness among the various components, aspects, objectives, and decisionmakers associated with such systems. This comprehensive and holistic approach is particularly essential in systems integration, where the couplings and interconnectedness among all parts and functions of the system as well as the influence of its environment on it must be accounted for [13]. Indeed, if all sources of risk are not identified, successful systems integration cannot be expected. The HHM framework presented here ensures that most, if not all, categories of risk from the multiple visions and perspectives are taken into account. In other words, when all sources of risk are identified and assessed systemically, and thus managed, then the process of systems integration is controllable; otherwise, the process is not controllable and the success of systems integration is left to chance.

The HHM framework for the identification of all sources of risks associated with systems integration consists of seven visions, perspectives, points of view, or decompositions (see

Fig. 3). In their totality, these seemingly seven disparate visions of sources of risk constitute the building stones of the risk identification mosaic. Briefly, the seven visions are:

- *Software Development*: Seven functions constitute the *software development* vision: requirements, specifications, architecture, process, testing, product, and support systems for integration. In other words, in this vision we attempt to identify and investigate most sources of risk associated with each function, stage, or phase of the software developmental life cycle. Needless to say that each of these functions is related (in the risk assessment process) not only to all other functions within the developmental life cycle but is also related to all other visions of the HHM. The software *development* vision addresses the issues within the development; it addresses the methods or the process perspectives—what and how to do it.
- *Temporal*: Six categories of sources of risk are identified for the *temporal* vision: requirements phase, architectural design, initial operating capability (IOC) phase, maintenance phase, final operating capability (FOC) phase, and upgrade and growth phase. At each stage of the system's life cycle—from cradle to grave—one should attempt to assess, to the extent possible, what can go wrong along with the associated frequency, severity, and consequences. These six categories provide a comprehensive scheme for the identification and assessment of the sources of risk associated with each category at different points in time, and with all the other sub-decompositions (within the six other visions of the HHM). Consider, for example, the sources of risk associated the *Requirements* phase. Certainly, decisions on finalizing the system's requirements affect and are affected by most subcategories of the *quality* vision (technical risk, cost overrun, schedule delay). Furthermore, there exists interdependence between the Requirements category and the other six visions within the HHM. The temporal vision addresses the propagation of the development effort within the life cycle of a product. For example, the requirements are likely to change over time; this change in requirements is very prevalent from IOC to FOC.
- *Leadership*: To the four categories of sources of risk are identified by Covey for leadership [7]: personal trustworthiness, interpersonal trust, managerial empowerment, and institutional alignment, we add two more—communication ability and technical competence—to make up the *leadership* vision. The *leadership vision* constitutes, probably, the best demonstration of the efficacious framework of the HHM. This vision, according to Covey [7] "recognizes that people are the highest value because people are the programmers—they produce everything else at the personal, interpersonal, managerial, and organizational levels." One can easily see how each of the six categories of the *leadership* vision relate to all the sub-decompositions of the other six visions within the HHM.
- *Environment*: The environment perspective addresses five sources of possible risk of failures: hardware failure, software failure, organizational failure, human failure, and failure due to sources external to the system. The *environment vision* is the most general and generic of all other visions. Its mission is to provide coverage of sources of risk where other visions may miss.
- *Acquisition*: Four categories of sources of risk are associated with the *acquisition* process that transcends the customer-contractor relationships: proposal phase, changes after the contract is issued, the review process, and the acceptance phase. This vision is self evident; most sources of risk associated, for example, with the *quality vision* have their genesis in the acquisition process, namely, in the selection of the contractors and in the interface between the contractor and the customer.
- *Quality*: The *quality vision* addresses both technical and nontechnical risks: technical performance of the product, cost overrun, and time delay in schedule. To assess the risk associated with the *quality vision*, one must address the overlapping with all other visions and their 29 sub-decompositions.
- *Technology*: This perspective addresses the type and extent to which new *technology* is used for process and/or product. This is largely based on know-how and experience. The type of technology and the extent of its use have a major impact on the risk associated with the *quality vision* and the *acquisition vision*, among others. Often the acquisition community (buyer and contractor) may underestimate the technical difficulty of a project and overestimate the availability of new technology.

In many ways, Fig. 3 does not do justice to the task of communicating the HHM concept to the reader. The most critical shortcoming of Fig. 3 is that the HHM philosophy builds on a multidimensional representation of a system—in the discussion here, the multidimensional representation of the sources of risk in systems integration. Yet, the two-dimensional depiction of the HHM in Fig. 3 conceals the couplings, interconnectedness, and the interactions among the various subsystems that constitute the sources of risk. Each of the seven visions (and as our knowledge of the role of software in systems integration in software-intensive systems improves over time, the list is likely to extend further and incorporate new aspects or visions that are not foreseen today) can be viewed as *the primary vision* from the engineer's, manager's, or analyst's perspectives. For example, a project manager may want to focus on the *quality vision* (see Fig. 4). In this case, *technical risk, cost overrun risk, and schedule delay risk* are envisioned as the manager's primary concerns. The HHM framework then enables this manager to trace, assess, and analyze all other factors affecting and affected by these primary sources of risk.

Fig. 4 depicts only one level of the HHM—namely, the *environment vision*. Fig. 5, on the other hand, depicts—again in a planar, limited way—an inversion of Fig. 4, technical risk, cost overrun risk, and schedule delay risk are viewed from the perspectives of the Environment vision. The ultimate efficacy of the proposed HHM framework lies in at least two dimensions: a) its capability to account for and display as complete a set of sources of risk associated with systems integration from their multidimensional perspectives as the analysts and

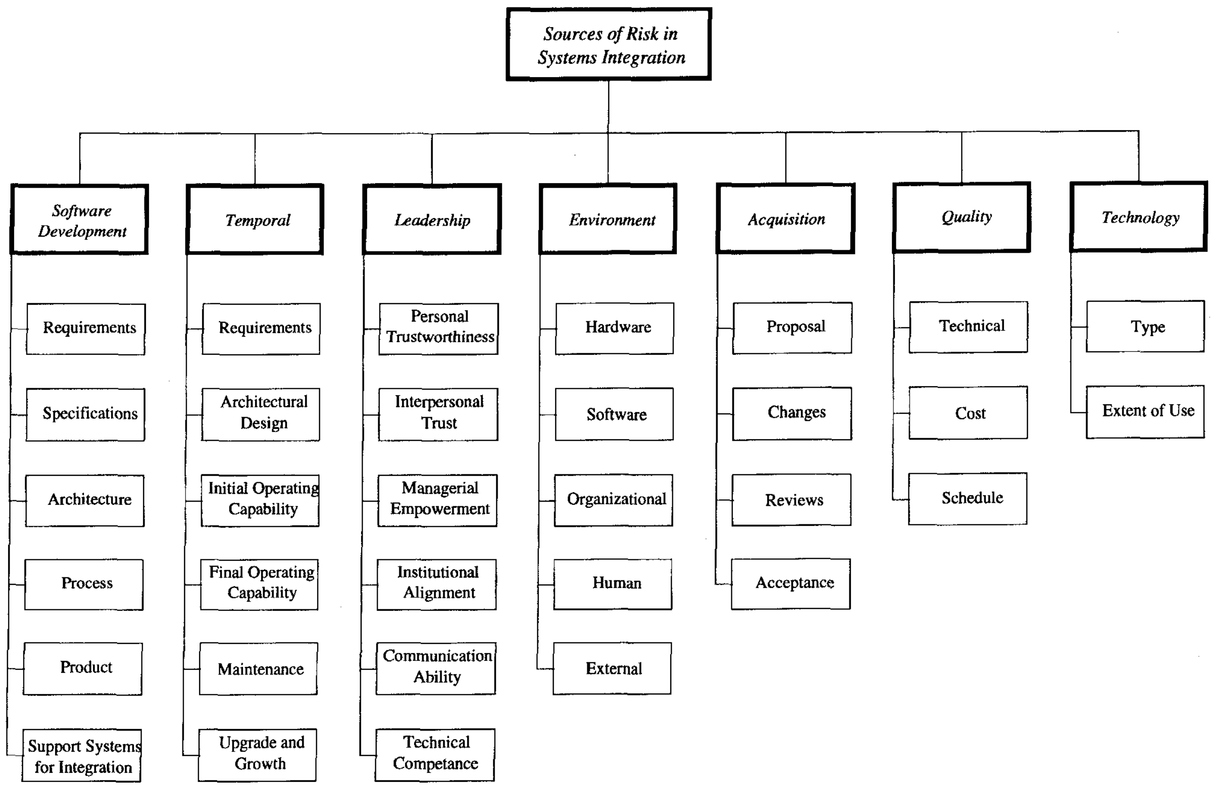

Fig. 3. Hierarchical holographic modeling framework for the identification of sources of risk in systems integration.

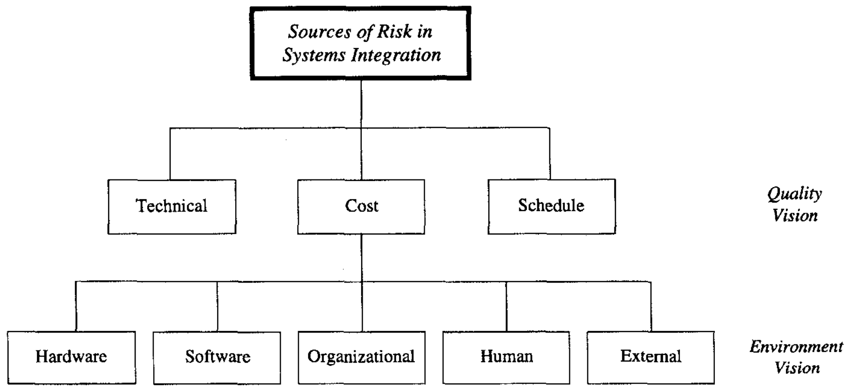

Fig. 4. Risk assessment: quality-based HHM structure.

experts can envision, and b) its facility to provide in-depth varied interpretations of the multifarious dimensions of risks arising whether from the functional, temporal, managerial, developmental, quality, or from other perspectives.

A cardinal principle that transcends all risk management activities is the essential need for prior identification and assessment of risk. These seven visions/decompositions of sources of risk and their corresponding 32 sub-decompositions should provide a comprehensive coverage of the sources of risk associated with systems integration; otherwise, additional visions must be introduced. (The generation of mitigation strategies to control these risks is the subject of another paper.) It is important to keep in mind, however, that since the HHM approach recognizes that no single vision or perspective of a system is adequate to represent a decomposed system by its components, only when all seven visions are considered in their overlapping nature, would a comprehensive identification of all sources of risk be achieved.

VII. Acquisition as a Precursor to Successful Systems Integration

This section will focus on the *acquisition vision* to demonstrate the efficacious attributes of the hierarchical holographic modeling framework for risk identification. The fact that most acquisitions of software-intensive systems face chronic cost overrun and delay in their delivery has become a common affair. For example, the C-17 transport aircraft, which has been called "the most computerized, software-intensive, transport aircraft ever built," is reported to be two years behind schedule

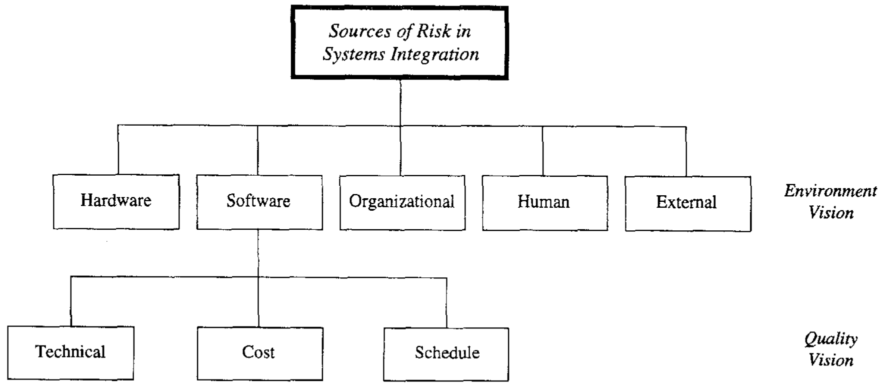

Fig. 5. Risk assessment: environment-based HHM structure.

and $1.5 billion over its initial cost estimate of $4.1 billion [10]. Many attribute these problems to the complex, outdated bureaucratic, and litigious governmental acquisition process. Of course, many nongovernment software acquisitions have also had severe overruns and cancellations, or the final product was never used. A recent report to the Deputy Undersecretary of Defense (DUSD) on the Acquisition Reform [8] highlights this concern:

> Software acquisition and development within the Department of Defense continues to be a significant management problem. With the ever-increasing importance of software as *the critical component* of today's Defense systems, and a growing annual budget roughly estimated at between $24 billion and $32 billion, a better understanding of the acquisition of software products and their development is becoming a critical issue.

Clearly, one way is to focus on the acquisition process (i.e., consider the problem from the *acquisition vision* of the HHM as depicted in Fig. 3), and delineate all other sources of risk and adverse consequences from that vision (while tracing the other 25 decompositions that are associated with the six other visions). In this case, selecting the contractor(s) and addressing the legal, organizational, technical, and financial issues associated with the selection process become the anchoring point. Furthermore, from a systems integration venue, an ad hoc acquisition process may prove to be the Achilles' heel for the entire project life cycle. One may trace, for example, the changes category of risk within the *acquisition vision* into the *temporal vision* and analyze its impact on IOC or FOC. Such a systemic process of risk assessment markedly adds to the assurance that a successful systems integration can be achieved.

The need to revamp the software acquisition process to more effectively manage the risks of cost overrun and time delay associated with software development has been widely discussed (see, for example, Haimes and Chittister [1993]). The earlier cited DoD report [8] addresses these problems from the educational perspectives and states "the technologies and commercial practices underlying many software acquisition competencies are rapidly changing, imposing a critical need for continuing education." Major recommendations of this report are to provide adequate "resources beginning in fiscal year 1995 to ensure the upgrade of existing mandatory courses for the Acquisition Management (Program Management and Communication-Computer Systems), Contracting, Quality Assurance, Acquisition Logistics, Systems Planning, etc. in order to meet minimum software competency requirements." These recommendations are harmonious with the ensuing discussion in subsequent sections on the revolutionary process that is taking place in the software area.

In summary, the purpose of this discussion on the acquisition process is to demonstrate the efficacy of the HHM in risk identification and to highlight the dependency of successful systems integration on a comprehensive risk assessment framework.

VIII. The Need for Metrics

Risk, which was defined earlier as a measure of the probability and severity of adverse effects [19], is a quantitative measure. To manage risk, one must be able to quantify it to the extent possible. This premise, which has been upheld in improving quality and effectiveness in the manufacturing of hardware—both for computer and noncomputer products has merits that are applicable to software development. The almost revolutionary improvements in quality and productivity that have been realized in manufacturing during the last decade or two are attributed by many to new ethics that embody a commitment to quality management and to the improvement of human resources at all levels of the organizational structure. Anyone who reviews the principles for a better management advanced by such quality improvement activists as Deming [1982], Groocock [1986], and Crosby [1984] cannot escape several common denominators, the most important of which are attention to the centrality of human resources and the importance of quantification in achieving greater reliability and predictability and in the management of quality. Indeed various metrics and other statistical measurement tools have been developed and successfully applied as a holistic instrument of management in manufacturing.

The software engineering development process has not as yet reached that level of integrated management, in contrast, for example, with manufacturing. One reason for this deficiency is the relative lack of appropriately available metrics for software engineering development (see [21], [22]). The number of lines of code remains a dominant metric used to represent a variety of other attributes of the software complexity and architecture. Furthermore, the centrality of software as an overall systems integrator within larger scopes and domains has not been fully recognized. The shift away from hardware and toward increasing reliance on software in the systems integration scheme has not been matched by a related development of appropriate software metrics—metrics with the capability to measure, predict, and control the software engineering development process. Such metrics can lead to a more cost-effective type of software engineering that promises quality, reliability, and adherence to prespecified time requirements and initially agreed-upon costs. Some argue for the emergence of another powershift in software engineering—one from emphasis on process to emphasis on product. The development of appropriate software engineering metrics would actually accelerate this shift, not necessarily from process to product, but to a better systems integration and a holistic approach driven by systemic risk assessment and management. In this case, the interface between process and product would be better understood and cease to be viewed in dichotomous terms.

Furthermore, it is not sufficient to identify the universe of risk factors along the life-cycle path of software development. To understand and appreciate the potential consequences of these risks and ultimately to take measures to mitigate their adverse effects or to prevent them, they must be measured and quantified to the extent possible. The introduction of appropriate metrics for software development thus becomes very important.

Observing the evolution of the field of Multiple Criteria Decision Making (MCDM) over the last three decades, one cannot escape the lesson learned from the challenges that faced (and continue to face) modelers and systems analysts during that period. In their quest to model, understand, and optimize systems with multiple, often conflicting, and noncommensurate objectives, whether for hardware-engineering-based systems or for large-scale socio-economic systems, experts in the MCDM field recognized the need for new measures (metrics) with which to quantify these objectives in non monetary units. Once the tools for optimizing multiple objectives were developed, a strong impetus emerged in the advancement in the art and science of systems modeling, and in the measurement and quantification of these multiple objectives. One can safely project that as the field of software engineering continues to grow, and as the importance of software risk management continues to dominate management decisions, then more effort will be devoted to the creation of metrics for software development.

The following examples of current metrics in software development demonstrate their "primitive" nature: complexity is often measured through the number of branches, number of decision blocks, number of lines of code, and/or the number of functions to be performed. Size of software development is often measured by the number of lines of code, memory needed, and/or the time required for coding and design. Probability is commonly measured through the ease of moving from one application to another end point. And, productivity is often measured through the number of lines or function points per unit time.

Clearly, significant concerted effort is needed for the development of more quantitative metrics so that an overall systems integration may be performed through the use of available quantitative risk-based decisionmaking methods and tools. The premise "to manage risk one must identify, assess and measure it," has multiple implications. First and foremost of those implications is the desire to evaluate the extent of the risk's probability and severity. Other implications include the desire to evaluate the effectiveness of various policy options for risk mitigation, or to assess the cost effectiveness of such actions. Furthermore, the ability to measure risk with credible metrics provides an environment that is conducive to the acceptance by management of the cost-benefit-risk tradeoffs associated with mitigating actions. These and other reasons that justify the need for and importance of metrics in software engineering may not be new. The emergence of software systems engineers as systems integrators, however, accentuates the important role of metrics in systems integration, particularly for software-intensive systems.

IX. Epilogue

The emergence of software as a powerful instrument in today's society and technology and the subsequent powershift from hardware to software call for a new assessment of the role of software as a systems integrator and the corresponding new role of software engineers. This shift also represents an important change in functionality, where software assumes more of the systems integration and implementation role.

This paper examines this evolutionary process which has been taking place in software engineering and in its role in software-intensive systems. Along with the need to manage this change, the paper also addresses the emergence of a new cadre of software engineers who at one time or another act as managers of risk, as systems integrators, or as risk experts who manage engineering systems. In all cases, however, the role of software engineering in software-intensive systems has become more critically important than ever before as systems are becoming more complex. In the quest for achieving a successful systems integration, this paper also considers the developmental life cycle of software and highlighted six major functions or milestones—*requirements, specifications, architecture, process, testing, and end product.* To manage risk one must identify all sources of risk, assess them and ultimately develop mitigating strategies for their management. The thesis advanced in this paper maintains that risk assessment and management, as a process, serves as the overall cross-functional systems integration agent that connects and unites these six functions/milestones along the life-cycle path. This thesis is indeed grounded on the premise that the process of risk assessment and management is the *sine qua non* requirement for ensuring against unwarranted time delay in a project's completion schedule, cost overrun, and failure to meeting performance criteria. Here, two sets of questions are advocated in the risk assessment and management process. First, in risk assessment, one should ascertain what can go wrong, what is the likelihood that things can go wrong, and finally what might be the consequences. Then, in risk management, one should ascertain what risk mitigation options are available, what are their associated tradeoffs in terms of risk, costs and benefits, and what is the impact of current policy decisions on future options. The paper relies on a hierarchical holographic modeling (HHM) framework for the identification of most sources of risk associated with systems integration. This HHM framework builds on seven visions or decompositions of sources of risk and on 29 sub-decompositions. These seven primary visions are termed: *Software Development, Temporal, Leadership, Environment, Customer-Contractor Interface, Quality, and Technology*. Although each vision presents only a one-dimensional perspective, all, considered together, offer a multidimensional holographic representation of most, if not all, sources of risk. The paper devotes one section to the importance of developing metrics in software engineering in order to do a better job in measuring these sources of risk.

References

[1] B. I. Blum, *Software Engineering: A Holistic View*. New York: Oxford Univ. Press, 1992.

[2] B. W. Boehm, "A spiral model for software development and enhancement," *Computer*, pp. 61–72, May 1988.

[3] ——, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice–Hall, 1981.

[4] M. J. Carr, S. L. Konda, I. Monarch, F. C. Ulrich, and C. F. Walker, "Taxonomy-based risk identification," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rpt. CMU/SEI-93-TR-6, 1993.

[5] C. Chittister and Y. Y. Haimes, "Risk associated with software development: A holistic framework for assessment and management," *IEEE Trans. Syst., Man, Cybern.* vol. 23, no. 3, pp. 710–723, May/June 1993.

[6] ——, "Assessment and management of software technical risk," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 2, pp. 187–202, Feb. 1994.

[7] S. R. Covey, *Principle-Centered Leadership*. New York: Fireside, 1992.

[8] Deputy Under Secretary of Defense, "Report of the software acquisition management education review team," A Report for the Deputy Undersecretary of Defense (Acquisition Reform), Mar. 7, 1994.

[9] R. Evans, "Lecture notes on systems integration," Center for Software Systems Engineering, George Mason University, Fairfax, VA, May 1994.

[10] General Accounting Office, "Embedded computer systems: Significant software problems on C-17 must be addressed," Washington, DC, GAO-IMTEC-92-48, Government Printing Office, May 7, 1992.

[11] Y. Y. Haimes, "Hierarchical holographic modeling," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 9, pp. 606–618, Sept. 1981.

[12] Y. Y. Haimes and C. Chittister, "An acquisition process for the management of nontechnical risks associated with software development," *Acquisition Rev. Q.*, vol. II, no. 2, pp. 121–154, 1995.

[13] A. D. Hall, III, *Metasystems Methodology*. New York: Pergamon, 1989.

[14] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*. Reading, MA: Addison-Wesley, 1989.

[15] S. Kaplan and J. Garrick, "On the quantitative definition of risk," *Risk Analysis* vol. 1, no. 1, 1981.

[16] J. Z. Lavi *et al.*, "Computer-based systems engineering workshop," in *Proc. SEI 1991 Conf. Software Engineering Education*, Lecture Notes in Computer Science, Springer Verlag, 1991(a), no. 536.

[17] ——, "Computer-based systems engineering workshop," in *Position Papers for the Int. Workshop in Practical Computer-Based Systems Engineering—Its Scope and Methods*, B. Thome, Ed., Atmosphere, Esprit Project 2365, Nov. 29, 1991(b), pp. 35–46.

[18] N. G. Levenson, *Safeware: System Safety and Computers*. Reading, MA: Addison-Wesley, 1995.

[19] W. W. Lowrance, *Of Acceptable Risk*. Los Altos, CA: Kaufmann, 1976.

[20] B. W. Mar, "Systems engineering basics," *Systems Engineering*, vol. 1, no. 1, pp. 7–16, 1994.

[21] R. Martin, S. Carey, M. Coticchia, P. Fowler, and J. Maher, Jr., *Proc. Workshop on Executive Software Issues*, Software Engineering Institute, Carnegie Mellon University, Aug. 2–3 and Nov. 18, 1988, Tech. Rpt. CMU-SEI-89-TR-6 and ESD-89-TR-6, Jan. 1989.

[22] L. H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, Within Budget*. Englewood, NJ: Prentice-Hall, 1992.

[23] A. P. Sage and J. D. Palmer, *Software Systems Engineering*. New York: Wiley, 1990.

[24] A. P. Sage, "The many faces of systems engineering," *Systems Engineering*, vol. 1, no. 1, pp. 43–60, 1994.

[25] ——, *Systems Management*. New York: Wiley, 1995.

[26] The Institute of Electrical and Electronics Engineers, Inc., IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12, 1990; (revision and redesignation of IEEE Std. 729-1983), 1990.

[27] B. Thome, Ed., *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering*. New York: Wiley, 1993.

[28] S. White, M. Alford, J. Holtzman, S. Kuehl, D. Oliver, D. Owens, C. Tully, and A. Willey, "Systems engineering of computer-based systems," *Computer*, vol. 22, no. 11, pp. 54–65, Nov. 1993.

Clyde G. Chittister (M'82) received the B.S. degree in computer science from the Pennsylvania State University, University Park.

He started his career in the field of real time process control systems, and developed and managed the design, implementation, and maintenance of automated control systems for the transportation and building automation industries. His management experience includes managing software development, program management, corporate research and development, and division level management. Mr. Chittister has had the opportunity to develop and implement both technical and business strategies for both the US and International markets. He joined the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, in 1985, and has held several senior management positions. He initiated and managed the Ada based Software Engineering Program and Systems Program at the SEI. He also managed and led the technical development for the Institute's Risk Management Program, and has published numerous articles on the identification and management of risks in software intensive systems. He was named Director of the Industry Sector in January 1995. Other professional experience includes 11 years at Union Switch & Signal, a Division of American Standard, five years with Brown Boveri Control Systems.

Mr. Chittister is a member of the IEEE Computer Society and the Society for Risk Analysis.

Yacov Y. Haimes (S'68–M'70–SM'76–F'80) received the B.S. from the Hebrew University, and M.S. and Ph.D. degrees in systems engineering from the University of California, Los Angeles.

He is the Lawrence R. Quarles Professor of Systems Engineering and Civil Engineering and founding Director, Center for Risk Management of Engineering Systems, University of Virginia, Charlottesville. Prior to joining the University of Virginia, he served as chairman of the Department of Systems Engineering, Case Western Reserve University, Cleveland, OH. He has been an active contributor to the theory, methodology, and applications of risk analysis for close to 20 years. He lectures in numerous workshops and short courses on risk assessment and management, multiobjective decisionmaking; has organized and chaired six engineering foundation conferences on risk-based decisionmaking (in 1980, 1985, 1987, 1989, 1991, and 1993) and organized and chaired more than a dozen national and international conferences. He authored/co-authored five books; edited/co-edited 17 volumes and authored/co-authored more than 200 editorials and technical publications, more than 120 of which are in archival-refereed journals and encyclopedias.

Dr. Haimes is an associate editor of *Risk Analysis* and *Reliability Engineering and Systems Safety*. He is a Fellow of the following societies: American Water Resources Association; American Association for the Advancement of Science; American Society of Civil Engineers; the International Water Resources Association; and the Society for Risk Analysis. He is the recipient of several honors and awards, including the 1994 IEEE-SMC Outstanding Contribution Award and the 1995 Georg Cantor Award from the International Society on Multiple Criteria Decision Making. He is a former Congressional Science Fellow, and has served on the staff of the Executive Office of the President and on the House Science and Technology Committee. He has served on national boards, commissions, and advisory committees (e.g., President of Universities Council on Water Resources, vice president of the IEEE Systems, Man, and Cybernetics Society).