

Systems Engineering of Computer-Based Systems

State of Practice Working Group
IEEE Computer Society Task Force on ECBS

Chair: Stephanie White, Grumman Corporate Research Center
Members: Mack Alford, Consultant; Julian Holtzman, University of Kansas;
Stephen Kuehl, Logicon; Brian McCay, Mitre; David Oliver, General Electric;
David Owens, Paramax Systems; Colin Tully, Colin Tully Associates; Allan Willey, Motorola

This working group report defines the need for a discipline devoted to engineering of computer-based systems, identifies current practice and needed research, and suggests improvements that are achievable today.

During the past decade, advances in microprocessor and network technology have led to the proliferation of complex systems with distributed processing and databases, internal communication systems, and heterogeneous components. The processing components can by themselves comprise a system, or they can be embedded in a physical system such as an automobile, aircraft, or medical diagnostic system. Both the encompassing system and the processing system are known as computer-based systems (CBSs).

Developing large computer-based systems with complex dynamics and component interdependencies requires analysis of critical end-to-end processing flows to determine feasibility and proper allocation. 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.¹

Industry has recognized the size and scope of problems with systems engineering processes, and several organizations have been formed to advance the systems engineering discipline. These include the American Institute of Aeronautics and Astronautics' Systems Engineering Working Group, the National Council on Systems Engineering, and Europe's Atmosphere Project.² However, recognition of the need for a special discipline addressing the system engineering of computer-based systems (ECBS) is just emerging, as evidenced by the recently formed IEEE Computer Society Task Force on ECBS and recently published textbooks.³

The IEEE Computer Society created its task force to promote the ECBS discipline, encourage research in the field, and establish a framework for education and training in 1991.^{4,5} This working group report, in presenting the case for an ECBS discipline, discusses current practices and identifies market and social imperatives for their improvement by

Table 1. Examples of systems and their relative complexity.

Complexity	Defense	Commercial	Public	Telecommuni- cations	Automotive	Financial
Very High	Intelligence fusion	Payroll information management	Weather forecasting		Intelligent highways	
	Space station	Airline reservations	Air traffic control	Central office switch		Econometric model
	NORAD	Wire services	IRS	Video text		Trader's desk
Medium	B1 bomber or cruiser	B-777	Bay Area Rapid Transit	Network control	Dealer networks	Portfolio management
	Logistics depot	Manufacturing automation	Nuclear reactor	Campus backbone	Vehicle management	New York Stock Exchange
	F-111	Process control	Observation satellite	Protocol servers	Auto braking	Claim form processing
	Patriot missile	Federal Express	US Postal Service		Engine management	Electronic funds transfer
Low	Smart bomb	Cell control		LAN control	Cruise control	Automated teller

- giving examples of CBSs and the number of engineers practicing ECBS in the United States,

- defining a CBS and ECBS responsibilities,

- identifying standard and advanced practices and areas for further research, and

- setting achievable targets for the engineering of computer-based systems designed to improve benefit-to-cost ratio, corporate profitability, and customer satisfaction.

System engineering

System types. Computer-based systems have become larger, more encompassing, and more complex. Table 1 gives examples of systems and relative complexity. Some are embedded systems, while others are stand-alone or cooperative computer systems. Virtually all systems above the low-complexity level (and even some of those) are computer based, event driven, and implemented using distributed hardware and software. Developing high-complexity systems can cost billions of dollars (low-complexity systems may cost less than a million), and system size and complexity is growing. Consider, for example, the complex

and stringent requirements that must be met by these systems:

- Space Station Freedom has approximately 1.5 million requirements.

- The Air Traffic Control System requires downtime of no more than six seconds per year for critical functions.

- Next-generation fighter aircraft are often dynamically unstable and, therefore, require extensive computer control to aid pilot control. Wing and tail control surfaces must be regulated at least 40 times per second to sustain acceptable flying characteristics.

- The modern automobile has more computing power than the Lunar Lander had when it landed on the moon. By the year 2000, the automobile computer is expected to have more interactive sensor-control loops than the largest chemical refinery in Baytown, Texas.

- Fully computerized systems, such as the Therac-25 radiation therapy machine,⁶ can cause injury or death, leading to increased requirements to prove the safety of system design.

System designer role. During early project phases, a system designer identifies requirements, translates requirements into designs, verifies that designed system behavior meets requirements,

allocates functions and behavior to components, and builds system descriptions. During the development and test phases, a system designer interprets requirements, guides designers of related systems, and directs trade-off studies. He or she also verifies that detailed designs will yield the required system behavior and evaluates change proposals. A system designer integrates multiple viewpoints, operating on line, to identify critical issues and the impact of external constraints. He or she resolves these conflicts as early in the project as possible. Although the system designer's tasks represent only 5 to 10 percent of the total project effort, their adequacy, accuracy, and timeliness are critical to the success of all complementary activities.

Design methods must be more systematic, and a few design teams have

Please join us

Readers interested in joining the Computer Society Task Force on Engineering of Computer-Based Systems should send their name, affiliation, address, phone and fax numbers, and e-mail address to TF ECBS, PO Box 400, Burlington, MA 01803.

accomplished this goal using manual technologies. Most teams, however, have not followed design "due process" because of the ponderous data-manipulation and cross-referencing tasks required, as well as the inability to "see" the systemwide behavioral implications of their designs. The pressure of project schedules and the time for coordinating multidisciplinary design teams through paper documents have been additional roadblocks.

System designer training. A system designer may have been trained in engineering, mathematics, physics, computer science, or management information. A system designer, who may have a variety of titles such as system engineer, computer system analyst, value engineering analyst, or industrial engineer, is recognized primarily by the type of work he or she performs. Many started in the Sputnik era and learned, on the job, from aerospace regulations written to guide the design of manned space-flight systems, intercontinental ballistic missile systems, and ballistic missile defense systems. The majority of these individuals do not understand distributed processing risks and issues because the earlier systems were not heavily distributed. Younger system designers coming from software backgrounds are ill-equipped to address system engineering and hardware issues such as distributed processor reliability, logistics, and human factors.

Number of computer-based systems designers. The population of system designers must be identified by what they do, rather than by industry, type of system being produced, or even job ti-

Computer-based systems require a different systems-engineering knowledge base than non-CBSs.

ties. Based on data provided during 1988 hearings before the United States Congress Joint Economic Committee, Ascent Logic Corporation estimated that, at that time, 300,000 people were doing system design in the US. Based on the percent of systems containing distributed computer systems and the relative complexity of these systems, Ascent Logic estimates that approximately one-third to one-half of US system designers are focusing on ECBS.

Engineering computer-based systems

Definition of CBS. A computer-based system consists of all components necessary to capture, process, transfer, store, display, and manage information. A CBS reference model based on the Posix distributed system model (Figure 1) is one of several models^{7,8} the task force is considering.

In the figure, processing entities include analog and digital hardware, firmware, and software. Communications entities provide network services that allow multiple processing entities to exchange information, transparent to application software. Information ser-

VICES provide information exchange between processing entities and storage devices, for example, disks or tapes. Human/computer interaction services, including windows, graphics, and command services, support interaction between processing entities and people. CBSs interact with the physical environment through sensors and actuators and also interact with external CBSs.

ECBS. By their very nature, CBSs require a different systems-engineering knowledge base than non-CBSs. All CBSs involve application software and associated services that are conceptual in nature and inherently difficult to grasp. Requirements satisfied by software are frequently ambiguous and subject to change. This leads to design changes that may sacrifice system architecture flexibility to ensure compliance with specified performance requirements. Furthermore, software changes in complex CBSs can result in unpredictable behavior, both internal and external to the CBS. Distributed CBSs are unique in that resources are frequently geographically dispersed and controlled by different organizations, so data exchange among systems requires interfaces to describe content and protocols to describe format.

We advocate a dedicated discipline to address these complex, unique CBS attributes. ECBS, as a discipline, is analogous to systems engineering in the traditional sense. What differs is the focus and hence the necessary skills. ECBS is concerned with the following responsibilities in addition to those of traditional systems engineering:

- identification of CBS requirements,

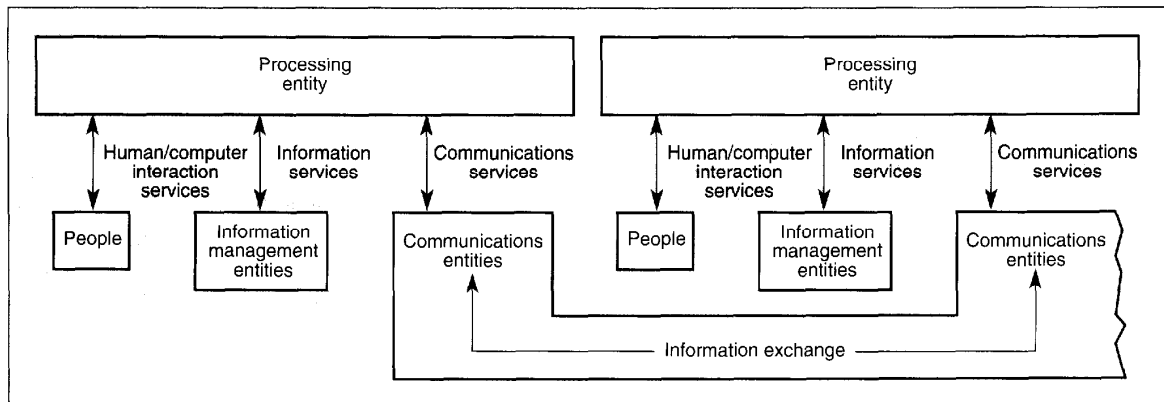


Figure 1. Distributed CBS reference model.

- design decisions concerning the CBS's distributed nature (its architecture).
- allocation of resources to component developers and management of the coordinated process,
- allocation of functions and data to CBS resources (processors, software, datastores, displays, interfaces),
- CBS strategies with respect to safety, security, and fault tolerance,
- global system management strategy,
- definition of services (see Figure 1),
- performance allocations (timing, sizing, availability),
- testing (component, integration, interoperability with the external environment),
- logistics support (maintenance, training), and
- implementation of the CBS within the existing environment.

Performing these tasks requires engineering trade-offs, prompted by operational requirements, limited resources (for example, finances and personnel), CBS component design (bandwidth, memory size, I/O subsystem, database system), system environment constraints (operational environment, security measures), and performance thresholds (timeliness, throughput, availability).

ECBS state of practice

Our conclusions concerning ECBS practice, presented below, are based on the extensive experience of working group members in defense and commercial industry, academia, engineering, research, and tool development. Forty ECBS Task Force members confirmed the working group's conclusions at the March 1992 internationally attended ECBS workshop.

The following description of ECBS practice addresses eight important issues—the ECBS process, requirements definition, design, interfaces, management, automation, documentation, and communication—but is not meant to cover all issues. The ECBS State of Practice Working Group would like to hear from other ECBS practitioners concerning additional state-of-practice issues.

ECBS process. When the system in question is a pure CBS system, the CBS

Table 2. State of ECBS process.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Undefined ECBS process 	<ul style="list-style-type: none"> • Systems-engineering process modeling (may include ECBS tasks) 	<ul style="list-style-type: none"> • Methods and tools for process modeling • ECBS roles and tasks

engineer is the principal system engineer, and the ECBS process is fully integrated with the systems engineering process. When the CBS is embedded in a larger system, the two processes must be tightly integrated, and information must flow freely in both directions. Engineers should integrate ECBS feasibility studies with those of related technologies, for example, sensor performance, human factors, and vehicle handling. Designers must fuse ECBS strategies and those of other disciplines to achieve reliability, safety, security, and fault tolerance. ECBS engineers should be instrumental in system model definition as the CBS model is a subset of the system model, defined to a greater level of detail. This integrated task requires unified system modeling and CBS modeling tools.

The systems engineer, with the support of the CBS engineer, is responsible for allocating system requirements, functionality, and resources to the CBS. The CBS engineer, with the support of software, hardware, and communications engineers, is responsible for distributed CBS design. He or she allocates CBS requirements, functionality, and resources to subsystems, processors, communication elements, software, data stores, displays, and the human/computer interface.

Table 2 summarizes the state of the ECBS process (additional information on the itemized points appears below).

Process definition. Many major companies are focusing on the Software Engineering Institute (SEI) Capability Maturity Model for planning their system engineering processes, but in general, corporations have not as yet defined their ECBS processes. Some companies use static modeling methods and tools such as the ICAM (integrated computer-aided manufacturing) definition language to capture and document their processes, while establishing quantifiable process metrics. A few compa-

nies use executable modeling tools (for example, RDD-100 and Statemate) for process modeling. These tools are based on net and state machine theory. Most process modeling tools in use today were designed for requirements modeling. Industry needs better methods and tools that support process assessment and improvement.

Process modeling needs. Process models are important because they guide projects in performing their major tasks. Industry needs a well-defined, executable ECBS process model that incorporates relationships to the overall systems engineering process, and metrics to measure process improvement and product quality. The defined ECBS process must be flexible enough to foster process improvement in a timely manner. Software process models such as SEI's⁹ provide a logical starting point for modeling the ECBS process.

An ECBS discipline is needed. Industry needs an ECBS discipline to research methods, develop technology, and educate professionals in the complex interdisciplinary issues that affect CBS development and performance. Trained CBS engineers must be on the job from program start-up, identifying critical processing flows, supporting feasibility analysis and requirements allocation, and ensuring that design decisions do not cause unnecessary CBS risks. The US Defense Systems Management College (DSMC) Systems Engineering Management Guide¹⁰ indicates that by the end of Concept Exploration, before the Demonstration Validation phase has started, roughly 70 percent of a system's life-cycle costs are locked in by design decisions. Some allocation decisions are made even earlier in the system development life cycle, before proposal submittal, when subcontracting responsibilities are decided. Many of these design decisions have major consequences for the CBS. Most systems engineers do not

Table 3. State of requirements definition.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Natural language for systems engineering • Inconsistent languages 	<ul style="list-style-type: none"> • Applying software modeling techniques to systems engineering 	<ul style="list-style-type: none"> • Consistent language • System modeling • Model-generated scenarios

Table 4. State of ECBS design process.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Static functional models • Hard-coded dynamic models 	<ul style="list-style-type: none"> • Planning for change • Starting decision capture 	<ul style="list-style-type: none"> • Systems/software dialogue • Trade-off support • Decision analysis • Reuse

have the skills to understand the consequences, perform the necessary analysis, or establish the required risk-avoidance measures.

Requirements definition. Table 3 summarizes the state of practice in requirements definition.

Language. Natural language, such as English, is the standard practice for system specifications. Most software engineers use software requirements models, usually based on structured analysis, frequently modeling data, dataflow, and control flow. Hardware engineers develop models using a hardware description language. The use of inconsistent languages by different disciplines leads to communication and traceability problems.

Information passed from systems engineering to other specialties must be complete and represented in that specialist's methodology and notation. Information must be transferred without manual reentry. Changes must be propagated in both directions. We need to pay attention to both semantic issues and tool interface issues during the transition to a consistent language and integrated toolset.

Methods and models. The typical systems-engineering requirements process iterates between requirements elic-

itation and system representation. A multiview system model provides the vehicle for communication between developer and customer, presenting system facts and system demonstration in an organized form. The same model provides views of function and capability to the implementer.

Static textual representations of a system depicting tiered sets of functions are insufficient. More advanced practitioners are applying software engineering methods such as structured analysis to systems engineering. These methods are superior to textual representations but are insufficient if they are not executable. Executable methods found in tools such as RDD-100 and Statemate do not yet support all ECBS functions, but they do provide dramatic technical advances over static modeling methods.

Practitioners need effective methods for specifying system performance requirements that support system design and derivation of software performance requirements. They also need useful paradigms that promote reuse of existing requirements specifications and use of previously developed components. Checklists are needed, for example, of the various trade-offs that could arise and ways of resolving them. Analysis for completeness, consistency, and correctness is primitive. Tools should apply logic, temporal analysis, and

domain understanding to the analysis problem.

Operational scenarios. Practitioners need models for generating a wide range of operational scenarios, including many with low probability. They need these scenarios to make an early determination as to whether requirements are consistent and adequate. Without these scenarios, engineers must wait until the system is operational to determine whether the system works properly within its environment.

Design process. Table 4 summarizes the state of the ECBS design process.

System/software engineering dialogue. In too many cases, systems engineers do not understand what information should be provided to CBS implementers. Systems engineers provide information in the wrong sequence and with insufficient detail.

We could solve these problems by defining a dialogue structure. For example, we should define the correct level of information needed to determine feasibility versus the correct level needed for design.

Static and dynamic models. Engineers, using standard software practices, produce static functional models that provide various representations for human review and analysis (for example, entity relationship, dataflow, and control flow diagrams). These engineers seldom use dynamic techniques for modeling system behavior such as executable state machines or nets. They frequently hard code dynamic models of fixed-point designs, and they use few parameters to support requirements changes and trade-off analysis. They have a limited ability to trade capability (dataflow and logic control) against resource utilization and performance. Researchers have done little investigation of nonlinearities caused by scale-up of capability or data, and engineers seldom analyze or model scale effects.

Planning for change. CBSs can change significantly during their life cycle, and research to improve processes must address this fact. When requirements change, engineers evaluate design decisions to determine if they are still valid. To support this evaluation, decisions must be well-defined, and documenta-

tion must include the decision rationale. ECBS needs effective design-decision capture. New processes must handle multiple system variants (system families) efficiently. Developers must trace the implication of design changes to/from higher level specifications and across families.

System developers are beginning to use open system architectures, as engineers plan for change.

Ramifications of systems engineering decisions. Systems engineers make "high-level," "architectural," or "systemwide" design decisions. These are policy decisions that should inform and constrain subsequent design and management decisions relating to various subsystems. It is unclear how to present and propagate these key decisions. 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. In addition, many high-level or system-level decisions result in major consequences to the CBS.

Engineers do not understand these consequences when making these decisions. Research is needed to determine what types of decisions have major consequences, the ramifications of these decisions, and how such decisions should be made.

Specification for reuse. Engineers using standard practice do not plan for reuse, usually building system parts that are too tightly coupled to a specific application to be reused. Engineers using advanced practice are performing domain analysis and designing parts that are reusable within their domain.

Industry needs effective models of system classes and of common subsystems/components that are used across classes. (An analogy from biology: mammals, fish, birds, insects are classes of animals; all have eyes, bones, hearts, mouths as subsystems and components.) Models of classes and components are important for supporting domain analysis and effectively storing high-level system segment or subsystem descriptions in a library. To effectively use a library, it is important to be able to assert: "What I want is exactly like that except for..."

Design architecture. Table 5 summarizes the state of ECBS architecture.

Table 5. State of ECBS architecture.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Performance at expense of architecture • Component hierarchies, few guidelines 	<ul style="list-style-type: none"> • Standard architectures, open systems 	<ul style="list-style-type: none"> • Domain architectures • Empirical data relating methods to quality designs

Table 6. State of ECBS interface definition.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Documented by data element description and protocol 	<ul style="list-style-type: none"> • Environment and subsystem documentation and simulation 	<ul style="list-style-type: none"> • Interface modeling • Capabilities for modeling host systems • Human/computer interface partitioning

Performance at expense of architecture. In standard practice, design emphasis is on system performance issues, such as end-to-end timing, at the expense of other architecture issues. These performance-optimized solutions are inflexible and hard to adapt to changing requirements or technology advancements.

Component hierarchies, standard architectures. Designers usually decompose unprecedented systems into component hierarchies using few guidelines. Standard domain architectures are appearing as building blocks for predated systems. To keep complexity to a minimum, it is important that designers base domain architectures on principles, concepts, and strategies that are natural for the class of problem.¹¹

Empirical data unavailable. Designers do not sufficiently use partitioning rules associated with maintainable systems. Therefore, there is not enough analytical data to validate these rules.

Interfaces. Table 6 summarizes the state of interface definition.

Level of abstraction needed. Engineers normally document interfaces by data element description and protocol, which is inadequate. We need research to define the level of abstraction that sup-

ports modeling the boundary properties of a set of subsystems. From these boundary properties, we must be able to verify that combined subsystem behavior matches the system requirements. If this cannot be done, a system must be built and tested to determine whether it meets the requirements.

Modeling the host system. The CBS must interact with an encompassing system such as a piloted vehicle or medical diagnostic system, which is frequently modeled with mock-ups and simulators. Engineers need additional capabilities for modeling host systems to predict the effects of the designed CBS on the host. Host systems are frequently human activities systems within which designed systems are to be used.

Human/computer partitioning. When engineers design systems, they implicitly specify the tasks of system users. Designers need better knowledge of how to partition functions and responsibilities between people and designed systems. Designers should not assign functionality to a system just because it is technically feasible, or even cost effective, but rather because the function is needed and the system is superior at performing it. Practitioners need better methods for modeling the human-computer interface. They usually do not model this interface adequately, since

Table 7. State of ECBS management.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Good cost data for precedented systems 	<ul style="list-style-type: none"> • Domain-specific work-breakdown structures • Risk management • Data collection 	<ul style="list-style-type: none"> • Determining ECBS costs • Process and product characterization for data collection

there is no defined process for integrating the knowledge of all stakeholders.

Project management. Table 7 summarizes the state of ECBS management.

Costing. There is existing literature on models and experiments for determining software costs. Similar documen-

tation is needed for systems engineering and ECBS.

Although software cost modeling is widely practiced, software cost models are not in agreement, and the underlying assumptions, hypotheses, and estimation techniques are not well documented. Different models result in significantly different cost estimates, us-

ing gross overall metrics based primarily on lines of code. These cost models apportion total software cost-and-effort estimates to each life-cycle phase.

Engineers need fine-grain metrics for each phase, and they need to better represent the ECBS life-cycle process model. Work must be tracked to calibrate cost models. Industry can usually estimate the cost of precedented jobs, but it still has problems costing unprecedented jobs.

Work-breakdown structure. Large projects normally track cost and schedule against a work-breakdown structure (WBS). If the WBS and CBS architecture are inconsistent, the WBS cannot support CBS status tracking. This inconsistency frequently exists, since en-

Seven working group members answer the question . . .

If there were an engineering of computer-based systems specialty, how would it improve the systems engineering discipline?

Answer #1. ECBS solves problems with the conventional top-down process by addressing end-to-end threads of system behavior from environmental stimulus to system response.

Expanded answer. Benjamin S. Blanchard, in *System Engineering Management* [John Wiley & Sons, 1991], defines systems engineering as the "effective application of scientific and engineering efforts to transform an operational need into a defined system configuration through the top-down iterative process of requirements definition, functional analysis, synthesis, optimization, design, test, and evaluation."

There are problems with the conventional top-down process for the following reasons:

(1) Engineers using a top-down process must understand present requirements, and be able to predict future requirements because corrections and additions can invalidate allocation decisions and require widespread architectural change.

(2) Interfaces must be fully defined and controllable. If subsystem interdependencies are not recognized at design time, they will not be recognized until system integration, when they will be very expensive to repair.

(3) Higher level decisions presuppose knowledge about the system. These decisions may be invalidated as lower level design is performed.

(4) Top-down definition ignores the process of understanding emergent properties. These dynamic higher level properties cannot be predicted from examining the set of components.

Computer-based systems engineering addresses these issues through improved communication and with interdisciplinary prototyping, modeling, analysis, change management, and risk assessment. The ECBS engineer uses executable models that characterize the operation of software, hardware, and communication components to manage dynamic system behavior such as end-to-end timing and network loading.

— Stephanie White
(Excerpted from White's paper, "Requirements Engineering in Systems Engineering Practice," *Proc. Int'l Symp. Requirements Eng.*, 1993, pp. 192-193.)

Answer #2. Systems-level computer hardware/software engineering skills are needed to model requirements and component interactions and to predict a system's attributes and effects on the environment.

Expanded answer. A more rigorous answer involves three steps: (1) defining systems engineering (SE), (2) defining engineering of computer-based systems (ECBS), and (3) answering the question.

Defining SE. Systems engineering is

- the identification of requirements R for a target system T that specify how T should interact with one or environment systems;

- the design of T in terms of a set of interacting component subsystems C so that T will display a set of (functional and nonfunctional) attributes A that predictably meet the set of requirements R ;

gineers define the WBS before they know the computer system architecture. To solve this problem, industry needs WBSs that support each CBS application domain.

Risk management. Successful projects perform risk management but use primitive methods. Engineers do not analyze detailed data from previous programs to understand cause and effect. Some risk-assessment tools do exist, but they do not interact with requirements and design tools where engineers identify risk issues. Industry needs general tool suites that support risk management views.

Data collection. In good practice, engineers collect data to manage the cur-

- the design of P , a production system (process to produce T); and
- the design of D , a development system (process to produce models of T and P) that considers F , the set of component (or functional) engineering processes (necessary to design C), and uses M , a set of modeling formalisms (for any of the above).

Note the distinction between a production system P and a development system D . P produces the target system T , or physical simulations/prototypes of T , using physical resources and production workers (not engineers); P is a substantial system in many cases (for example, space shuttles, bridges, VLSI chips) but trivial in the case of software. D produces and operates on information models of T or P , using physical resources, modeling formalisms M , and engineers. Both P and D need to be designed.

Defining ECBS. ECBS is the system-level application of computer hardware/software engineering skills in identifying requirements R for target system T and in designing T , in the case where

a subset (some or all) of C constitutes a set of computer (information handling) subsystems C_i ,

In these circumstances, computer hardware/software engineering skills are apt to be used in designing a production system P and/or development system D .

The answer. ECBS improves the SE discipline because it is indispensable to the increasing proportion of systems where C_i is non-null. System-level computer hardware/software engineering skills are needed, for instance, to model interactions involving C_i , to model part or all of R , to predict A and the effects of T on E , and so on.

If C_i is null, ECBS is by definition not a part of the SE process for T . Nevertheless, some ECBS methods may still be used in P and/or D . This is a second way in which ECBS improves the SE discipline.

— Colin Tully

Table 8. State of ECBS process automation.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Task oriented, some tool interfaces, software environment 	<ul style="list-style-type: none"> • Frameworks, approach to repetitive tasks 	<ul style="list-style-type: none"> • Integrated systems/ ECBS environment • Efficient change management

rent project. In best practice, they use data to improve engineering processes and make predictions for future projects. Research has provided some capabilities for collecting software sizing data and using the data to size new software components.¹²

ECBS needs better techniques for process and product characterization, and

industry must collect and have access to the data it knows how to characterize.

Process automation. Table 8 summarizes the state of ECBS process automation.

Levels of tool support. There are several layers of ECBS tool support. At the

Answer #3. ECBS adds accuracy and completeness to systems specifications.

Expanded answer. Adding the ECBS discipline to systems engineering will provide the competence to accurately model and specify the complex behavior and architecture of the operator, computer, and software portions of modern systems. The methods and tools to produce specifications that have been checked by computer for consistency and correctness are becoming available and can be used to generate executable specifications for other engineering disciplines, such as mechanical engineering. The ECBS discipline is necessary to supply the trained human element for this progress in engineering.

— David Oliver

Answer #4. An ECBS discipline would lead to improved educational programs in systems engineering.

Expanded answer. Few universities cover systems engineering as it's described in this report. To a degree, systems engineering is simply the practice of good design. Beginning with requirements analyses (defining the problem), progressing through trade-offs (examining potential solutions), and culminating in verification and validation that the solution matches the problem, we believe that teaching SE to engineering students should be a basic requirement for accreditation.

Design is poorly taught in most universities partly because faculty have little design experience, but largely because of the concentration on artifacts instead of processes. (Weren't we graded on the answer, not how we got it?) Thus, young engineers graduate with little appreciation or understanding of the process of bringing a product or service into being.

Can SE be taught alongside the traditional curricula? In most cases the answer is "no," since traditional departments are bounded by walls that are difficult to cross. Can we instead expect to see a multitude of disciplines: SE for aerospace, SE for electrical engineering, SE for chemical engi-

Table 9. State of ECBS documentation.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Natural language specifications 	<ul style="list-style-type: none"> • Databases and documentation generators 	<ul style="list-style-type: none"> • Integral role with process • Designed approach

lowest functional level, automated tools — for example, an automated requirements capture tool — accomplish a specific task in a particular phase of the system life cycle. At the next level, tools assist the systems designer in accomplishing many tasks across many life-cycle phases. A requirements tracing tool provides this level of support. Fur-

neering? Again, the answer is probably "no," since traditional programs are already overloaded and would resist canceling current technical courses or areas to introduce one that produces a generalist. But we can expect to see SE for computer engineering or computer science, since these disciplines are growing in breadth, not merely depth. As detailed in the body of this report, we believe that this ECBS discipline would take the lead in improving design and SE for all engineering disciplines.

— Julian Holtzman

Answer #5: Given the increased criticality and uniqueness of software, an ECBS discipline would bridge the knowledge gap between systems and software engineering.

Expanded answer. ECBS would formalize today's ad hoc processes where systems engineers are treating software like other system components and software engineers are practicing systems engineering without a license. An ECBS discipline would develop engineers, trained in both systems and software engineering practices, who would understand the structure of the software and how it interrelates to the rest of the system and its external environment. The aim would be to reduce cost of ownership through ECBS practices that consider the intricacies of software.

A real problem we face today is that system and software maintenance are done by different engineers than those who design the system. ECBS would provide the expertise to sustain CBSs through an established ECBS process that could be applied throughout a system's life cycle.

— Brian McCay

Answer #6: Traditional systems-engineering specialties have arisen from the requirement to ensure that a special need is covered during system definition. The ECBS specialty helps the team understand and quantify the real and imagined issues associated with computers.

Expanded answer. The diversification of today's engineered systems requires the systems engineer to assure the communication and documentation of analyses produced by specialty engineers. Most of these specialists focus their attention on traditional system trouble spots. For example, in developing electronic systems, electromagnetic compatibility/electromagnetic interference (EMC/EMI) become very important as the spectral density and field strength increase. Increased EM fields interfere with normal operations, and designers must stop this interference with

design actions based on the EMI/EMC analysts' predictions. In many design teams, the EMI/EMC specialty is still considered a "black art" due to the lack of understanding of the underlying technology.

Today, computers and software are major items in many systems. Unfortunately, many systems engineers still consider computers and software "black arts" (like EMC/EMI). The lack of an ECBS discipline lets computer hardware and software engineers practice their trades with little involvement or understanding by the overall system stakeholders. The lack of well-defined and understood requirements, and the related specialty analysis, allows system-level disasters, major failures, large cost overruns, and even deaths.

The analysis needed for proper definition of systems containing computers is improved by integrating an ECBS specialty onto the systems engineering team. The ECBS-trained systems engineer communicates the requirement to the implementer and abstracts specific implementation constraints to the system level. In addition, he or she interprets the constraints imposed by other specialties and solves the real hardware/software issues within the specific system context. Working as a systems engineering team, the various specialties can define the complex systems that depend more and more on the leverage that computers can supply.

— David Owens

Answer #7. Without the CBS engineer, the system engineer will be working with incomplete information and make improper trade-offs.

Expanded answer. The heart of the systems engineering process is to (1) specify system characteristics viewed as a black box, (2) postulate an architecture, (3) allocate functions and performance to components, (4) subject the design to the analysis of component developers (that is, for feasibility, cost, risk, and performance) and specialty engineers, (5) perform trade-offs among multiple designs to select and document one, and then (6) let component developers develop the specified component.

Without the ECBS role, estimates of CBS feasibility, cost, and schedule will not be properly developed and fed back into system trade-offs. Hence, the systems engineering process will fail due to incomplete or poor information (garbage in, garbage out). Only when an engineer performs the ECBS role will the correct CBS cost, schedule, and risk information be supplied to the systems engineer. Only then can the CBS component be designed and developed to satisfy system-level requirements.

— Mack Alford

ther, there are tools and techniques designed to support management and integral processes across all system life-cycle phases, for example, a problem tracking and reporting tool. At the highest level, there is a class of tool that provides a "framework" into which individual tools are deployed appropriately over the life cycle and that maintains all records and documentation for system development and operation. Analogous concepts are found in computer-aided design with the CAD framework initiative and in software with integrated project support environments.

Higher level tools needed. Unfortunately, there are few examples of sophisticated, high-level process automation tools in widespread use. The need is recognized, but tool suites have not kept pace.

One area needing better tools is change management. Efficient change entry is a major problem for large complex systems. Major changes may require updating information about a single entity in several places in databases of a number of tools. This multiple manual update process is both costly and error prone. In standard practice, engineers frequently do not make all the changes, due to the effort involved. A system description repository, which is a standard model/schema underlying all tools in use, is a desirable solution.

Documentation. Table 9 summarizes the state of ECBS documentation.

Integral role with process. Industry tends to focus on documents that are delivered to the customer, but each and every artifact of the process is an element of system documentation. Engineering drawings, test reports, software source code, requirements tracings, defect tracking reports, and many other such "documents" are a part of what we do. To date, industry has paid little attention to the integral role of capturing all these artifacts as part of the ECBS process. Significant change will occur only when industry integrates methods and tools to support improved processes throughout the ECBS life cycle and when documentation is an integral part of that process.

Designed and automated approach. In part, ECBS is to blame for the weak role that process automation plays. With-

Table 10. State of interpersonal communication.

Standard Practice	Advanced Practice	Research Needed
<ul style="list-style-type: none"> • Diverse team: Same syntax, different semantics 	<ul style="list-style-type: none"> • Concurrent engineering • Training 	<ul style="list-style-type: none"> • Defined roles • ECBS as integral part of systems engineering

out necessary databases and automated tools, control and maintenance of documentation is tedious, repetitive, and prone to human error. The introduction of general-purpose tools such as word processors and electronic spreadsheets has contributed to improvements in this area, but more sophisticated tools are required. We need a "designed" approach to process and documentation that is supported by automated tools.

Interpersonal communication. Table 10 summarizes the state of interpersonal communication.

Diverse team. Successful application of the systems engineering process hinges on the ability of a diverse team of specialists to communicate with a common viewpoint. This is especially true for CBSs, where diversity of backgrounds adds complexity to the communication process.

Historically, the team building an electronic system, such as a radar system, consisted primarily of electrical and mechanical engineers and technicians. The common background and clearly understood roles of team members tended to minimize communication problems. Computer-based systems development has brought individuals with different backgrounds to the team, for example, computer scientists and mathematicians. Also, as system complexity has grown, we've seen increased involvement by people from such areas as business, quality, safety, human factors, and security. Diverse backgrounds prevent the development of a common understanding of key concepts. Team members believe they're using the same semantics when, in fact, they aren't.

Undefined process. Poor process definition is also a problem. Software and computer hardware engineers, as detailed design engineers, perceive their interface with systems engineers as poorly defined. The problem lies in the lack of a precise definition of the allocation

process and a failure to trace detailed specifications to the system's top-level functionality.

Component view. Another cause of difficulty is the view that software, computer hardware, and communications assets are component pieces of the systems engineering discipline rather than a whole. Systems engineers find it difficult to break out of this partitioning paradigm, and software engineers are not apprised of the trade-offs and design decisions. For example, if CBS engineers are informed that decisions concerning intersystem data transfer are under consideration, they can inform systems engineers that additional application data is needed for error checking. If the added data is significantly large, communication link size can be an issue.

Cross-training needed. Academic and industry in-house training programs for systems and software engineering must take a more interdisciplinary view, share some common courses, and work toward the development of a common set of semantics.

Achievable systems engineering improvements

Developing and using complex systems requires major capital investment. Industry must make investments in large systems wisely, with thorough consideration of what the system is to do and the feasibility and rewards of doing it. The systems engineer is an important contributor to successful system development. He or she is responsible for capturing user needs and defining a feasible system design that meets all system requirements and constraints.

Industry can improve systems engineering by advancing ECBS practice. This will reduce development and life-

cycle costs and improve system quality. Corporations can implement each of the following suggestions today.

CBS modeling. Various modeling approaches augment text specification with semantically precise representations for engineering information. Since modeling and model analysis can provide a grand-scale improvement in the process of constructing systems,¹³ we must quickly and economically close the gap between current text-based practice and future model-based practice. Closing the gap means an extensive cultural change and substantial retraining.

Retraining efforts must target methods and notations that engineers can readily learn, since retraining costs usually dominate transition costs when implementing new methods. If possible, new notations, methods, and concepts should evolve gracefully from those currently in use.

A defined process. The engineering process includes a sequence of process steps and policies for process control, documentation, and staffing. Industry should define the process in layers so that the same engineering steps can be used with alternative methods of control, documentation standards, and staffing choices. Layers of description may include:

(1) *Description of process steps, including process sequences, concurrencies, iterations, recursions, inputs, and outputs.*

(2) *Description of the control process and organizational groups or boards that perform control.* During any process step (for example, requirements, design, implementation, or test), issues may be raised. Some of these issues could require changing the system baseline. This description shows how issues are raised and reviewed by resolution boards, addresses the resolution plan, identifies how resolution progress is tracked, and specifies how the resolution is documented.

(3) *Description of representations used for information captured at each step of the process.* For each engineering step, there will be several ways of representing the information.

(4) *A mapping of each step and representation onto a tool that automates that step.*

(5) *Description of staffing.* The manner in which teams of talent are as-

signed process tasks determines how the work is done. Work can be done in a sequential manner that encourages "handoff" to poorly communicating and distinct disciplines, or it can be done by interdisciplinary teams that provide downstream realities and costs in development, integration, field service, and so on. The latter approach supports concurrent engineering or house-of-quality approaches. It is the grouping of talents and timely assignment that makes the difference.

(6) *Description of the review process and information to be reviewed.* In traditional text-based practice, documents are reviewed at the end of each process step. The required content of the documents drives the process steps. Since required document content varies from contract to contract, specifying a standard process in terms of document production is difficult. This is true even when a development standard, such as Department of Defense Standard 2167A or 490, is being followed.

Industry should base the systems engineering process on a set of models that define engineering information in a way that computers can capture for interpretation, execution, consistency, and correctness. The process should support one-time entry of information for both new designs and changes. When the engineering process is based on model construction, documents can be generated from the system description repository. Models should be reviewed, as well as documents, because this practice substantially improves the process.

Domain analysis. System engineers should establish classes of real-world objects that are identified or implied by system requirements. They should perform domain analysis, defining and parameterizing object classes that are always present in that type of system. These steps would support reuse of high-level system segment or subsystem descriptions, as well as the development of domain-specific software architectures and software reuse.

Dynamic analysis and simulation. For rigorous system descriptions, industry should capture system behavior in a representation that can be executed dynamically for analysis. Developers should apply the same representation to scenarios that describe how the system interacts with its environment. In

addition, industry should use simulation to prove feasibility and develop benchmarks.

Performance requirements management. Traditionally, engineers budget performance requirements to components as they decompose the system. They budget quantities such as weight, power, heat production, heat dissipation, reliability, time response, and memory size from a parent component to subcomponents. They should use methods and tools to track these budgets, and they should compare design, simulation, and test results to budgeted specifications.

Automated document ingestion. Engineers developing large systems must handle volumes of text and graphics in requests for proposals, contracts, system documentation, documentation of engineering processes, and user manuals. When using existing documentation, as in reengineering, it is helpful to have the documents in electronic form. It is even more helpful to put the documents in an automatically linked hypertext form. The linking can be based on automatic identification of entities that users deal with in their work.

Metrics, costing, and tracking. Metrics, costing, and tracking are essential both for short-term decisions and for long-term continuous process improvement. Well-defined and broadly accepted metrics are required to quantify the work. Industry needs cost models to transform these metrics into numbers for particular applications. Organizations must track work, both to control activities and to calibrate cost models.

Scalability. Closing the gap between current practice and mature practice involves serious issues of proof of scalability. Engineers must try new methods and tools on modest-sized systems to eliminate deficiencies and incorporate unanticipated benefits.

Process improvement. The Capability Maturity Model (CMM) was developed by SEI to support software engineering process improvement. This model and method for fostering process improvement is applicable to systems engineering and ECBS if detailed tasks are changed. CMM maturity levels are useful for prioritizing the order in which new process methods and modeling techniques are developed and introduced.

Investment choices with respect to process improvement are difficult in these times of strong competition. Organizations must make decisions at a time when there is no solid baseline of methods, metrics, and costs for the systems engineering process. They can estimate anticipated cost improvement, but they cannot derive it from existing data.

In summary, software, computer hardware, and communications engineering are not well integrated, and many engineers are performing ECBS tasks without defined positions, tasks, and roles. An improved ECBS discipline is necessary for making proper CBS design trade-offs, decisions, and allocations; for process improvement; and for fostering research and training. ■

References

1. J.Z. Lavi et al., "Computer-Based Systems Engineering Workshop," *Proc. SEI 1991 Conf. Software Eng. Education, Lecture Notes in Computer Science, No. 536*. Springer Verlag, 1991.
 2. H. Obbink, "Systems Engineering Environments of Atmosphere," *Proc. European Symp. Software Development Environments and CASE Technology, Lecture Notes in Computer Science, No. 509*, Springer Verlag, 1991.
 3. *Systems Engineering, Principles, and Practice of Computer-Based Systems Engineering*, B. Thome, ed., John Wiley & Sons, 1993.
 4. A.K. Agrawala, J.Z. Lavi, and S.M. White, "Task Force on Computer-Based Systems Engineering Holds First Meeting," *Computer*, Vol. 24, No. 8, Aug. 1991, pp. 86-87.
 5. J.Z. Lavi et al., "Formal Establishment of Computer-Based Systems Engineering Field Urged," *Computer*, Vol. 24, No. 3, Mar. 1991, pp. 105-107.
 6. N.G. Leveson and C.S. Turner, "An Investigation of the Therac-25 Accidents," *Computer*, Vol. 26, No. 7, July 1993, pp. 18-41.
 7. M. Alford, "Strengthening the Systems Engineering Process," *J. Eng. Management*, Mar. 1992.
 8. S.M. White, "A Pragmatic Method for Computer System Definition," PhD dissertation, Polytechnic Univ., Univ. Microfilms, Ann Arbor, Mich., 1987; also Technical Report RE-791, Grumman Corporate Research Center.
 9. M.I. Kellner, "Supporting Software Processes Through Software Process Modeling," *Proc. Sixth Int'l Software Process Workshop: Support for the Software Process*, ACM, New York, 1990.
 10. Defense Systems Management College, *Systems Engineering Management Guide*, US Govt. Printing Office, Washington, D.C., 1990.
 11. H.W. Lawson, "Philosophies for Engineering Computer-Based Systems," *Computer*, Vol. 23, No. 12, Dec. 1990, pp. 52-63.
 12. G.J. Bozoki, "Performance Simulation of SSM (Software Sizing Model)," *Proc. 13th Ann. Conf., Int'l Society of Parametric Analysts*, 1991, pp. 14-28.
 13. D. Harel, "Biting the Silver Bullet: Toward a Brighter Future for System Development," *Computer*, Vol. 25, No. 1, Jan. 1992, pp. 8-20.
- Stephanie White**, principal engineer of software process at Grumman Aerospace's Corporate Research Center, leads an interdivisional team solving system/software engineering interface issues and serves on the Metrics Technical Advisory Group at the Software Productivity Consortium. Her primary research interests are system and software requirements modeling, analyzing system behavioral models, and traceability. She serves as vice-chair of the Computer Society's ECBS Task Force and previously chaired its State of Practice Working Group. She has an MS in mathematics from New York University and a PhD in computer science from the Polytechnic University of New York.
- Mack Alford**, now an independent consultant, continues as technical advisor to Ascent Logic Corporation. He was a founder and chief scientist of Ascent Logic, which develops interactive graphics tools to support systems and software engineering. Alford has 32 years experience in developing systems and software and has spent more than 20 years researching methods and tools to support specification, design, and test of real-time embedded distributed systems.
- Julian Holtzman**, professor and director of the Center for Excellence in Computer-Aided Systems Engineering at the University of Kansas, has more than 30 years experience in higher education and in the private sector. Currently, he is the president of Lawrence Applied Research Corporation and is involved in systems-engineering research, focusing on tool integration, software reuse, and enterprise modeling. Holtzman earned his PhD degree at Cornell University.
- Stephen Kuehl**, a computer-based systems engineer with Logicon, is supporting the V-22 avionics systems/software engineering development and implementation effort at the Naval Air Warfare Center, Aircraft Division. Before this assignment, Kuehl was systems engineering requirements analysis/definition task leader for the McDonnell Douglas Corporate Avionics Systems Integration and Acquisition (ASIA) project. He received a BS degree in telecommunications from Indiana State University and a BSEE from the University of Evansville, Indiana.
- Brian McCay** is a lead engineer at the Mitre Corporation in Bedford, Massachusetts. McCay's current interest is in managing and controlling the technical evolution of computer-based systems-of-systems through the use of systems engineering tools that capture and maintain system architectures. He is an active member of the National Council on Systems Engineering and of the IEEE Computer Society Task Force in ECBS. McCay holds a PhD in applied mathematics from Oregon State University.
- David Oliver**, who joined General Electric Corporate Research and Development in 1961, is now a systems engineer in the Information Sciences Laboratory, GE-CRD. His experience includes materials science, ultrasound, factory automation, instrumentation for quality measurement, software tools, systems engineering tools, and research transition. Oliver chairs the Process Committee of the Computer Society's ECBS Task Force. He graduated from the Virginia Polytechnic Institute and Massachusetts Institute of Technology with degrees in physics.
- David Owens** is director of software engineering on the New Shipboard Aircraft for Paramax Systems of Montreal, Canada. Owens has held various management and technical positions with NCR, Boeing, McDonnell Douglas, and the Software Productivity Consortium. His interests include the application of technology in contracted programs and of multiple processor computer systems in product applications. He received his BSEE from North Dakota State University.
- Colin Tully**, a consultant in the United Kingdom, specializes in computer-based systems engineering, the software process (modeling, assessment, and improvement), software quality audit, CASE integration and software factories, and technology transfer. He often works in Europe's ESPRIT collaborative R&D program. He is a chartered engineer, a fellow of the British Computer Society, and editor of Wiley's series on software-based systems. He has an economics degree from Cambridge University.
- Allan Willey**, a technical staff member of Motorola's Cellular Infrastructure Group, is leading an effort to enhance the process of engineering cellular products to achieve quality, productivity, and cycle-time reduction goals. Willey also chairs a Motorola committee to develop a 10-year vision for software engineering technologies and facilitates the software focus of the Council on Competitiveness. He received a BA degree in philosophy from the College of William and Mary and a BS in mathematics and MBA in operations research/management science from George Washington University.

Readers may contact White at Grumman Corporate Research Center, MS A08-35, Bethpage, Ny 11714; her e-mail address is steph@dstech.grumman.com.