

Engineering Computer-Based Systems: Meeting the Challenge



As ECBS assumes a greater role in computing-product development, engineers face increasing pressure to resolve the challenges of disparate architectures, long interdisciplinary system life cycles, method and tool incompatibility, and spiraling complexity.

*Stephanie
Farbman
White*
Long Island
University

*Bonnie E.
Melhart*
Texas Christian
University

*Harold W.
Lawson*
Lawson Konsult AB

Embedded computers in system products and services have become so pervasive that we cannot easily envision our society functioning without them. Although this trend has provided many new opportunities for exploiting the technology, our dependency on computer-based systems has made society vulnerable in many ways. It is thus essential to establish improved practices in the engineering of computer-based systems. The Technical Committee on the Engineering of Computer-Based Systems¹ was formed to meet this challenge, and it actively pursues the goal of improved practices, including establishing ECBS as a profession and academic discipline. The “ECBS Technical Committee” sidebar describes these efforts.

ECBS addresses a wide variety of life-cycle issues involved in the engineering and management of the computer content for system products and services, emphasizing a holistic approach across multiple application domains. Such systems include telephony and communications, airline reservations, electronic banking and commerce, process control and computer integrated manufacturing, avionics and aerospace, transportation, medical instruments, microprocessor-controlled domestic equipment, and military applications.

ENGINEERING CONTENT

ECBS's broad scope² relates closely to systems engineering, raising the question: What is the engineering content of this discipline? Engineers design and develop systems or components composed of structures that have specific behaviors. They learn to evaluate alter-

natives and make trade-offs between various combinations of structures, functions distributed to the structures, and the resulting behaviors. The trade-offs aim to achieve particular properties with respect to performance, robustness, safety, security, cost, quality, environmental consequences, time schedules, and so on.

These structural and behavioral aspects—while quite apparent for fundamental disciplines such as electrical, mechanical, and civil engineering—also permeate the systems level. To work effectively at this level, engineers must be able to define, evaluate, and make trade-offs related to function distribution between the alternative multidomain structural composites, that is, architectures, used in eliciting desired—or inhibiting undesired—system behaviors. Although it has some domain-specific aspects, CBS engineering knowledge and skills are for the most part generic and applicable across multiple domains.

KEY ASPECTS

CBS engineers must understand the importance of achieving an appropriate relationship between system architecture and the life-cycle processes applied in conceptualizing, developing, producing, operating, and maintaining computer-related system content.

The importance of well-structured and consistent system architectures cannot be overemphasized. Many of the problems that arise when dealing with complex computer-based systems relate to the poor architectural underpinnings that complicate virtually all life-cycle processes as well as supporting methods and tools.

A strong architecture significantly reduces system complexity by properly distributing function between

hardware, software, and humans. Further, a strong architecture is typically based upon a few well-defined, consistent concepts and provides a minimal but sufficient set of interface standards and mechanisms.

A strong architecture tips the balance so that dependencies upon processes and related methods and tools become lighter. Integration, testing, verification and validation, operation, and maintenance are simplified. On the other hand, a weak architecture often tips the balance by making processes, methods, and tools and their assessments heavier and more complex to compensate for the architecture's complexities. Obviously, then, CBS engineers must strive to create strong architectures.³

Even with strong architectures, treating complexities and meeting the stringent demands for safety and security require well-defined life-cycle processes allocated appropriately to the actors involved in achieving system-related objectives. Key life-cycle process activities that CBS engineers perform thus include:

- Identify, analyze and bound the domain-specific problem the CBS addresses.

- Establish criteria that match the CBS solution to market needs.
- Based on trade-offs, develop alternative solutions to the problem and evaluate them with respect to CBS criteria.
- Develop CBS conceptual models.
- Elicit and formulate system requirements; analyze, reconcile, allocate, and manage them; and prepare the system specification.
- Design, assess, and select feasible computer-based systems and their architectures.
- Effectively apply modeling, prototyping, and simulation techniques.
- Derive the requirements for the CBS hardware, software, communication, and human-machine interfaces.
- Analyze CBS behavior and performance and prepare specifications.
- Take an interdisciplinary approach to designing the hardware, software, communication, and human-machine interface subsystems.
- Establish test, verification, integration, and validation strategies.



ECBS Technical Committee

The IEEE Computer Society Technical Committee on the Engineering of Computer-Based Systems envisions that its role is to provide worldwide focus for ways to meet the challenges associated with the incorporation of computers in systems. The committee provides that international forum, that focus.

Origins

In the late 1950s, engineers and educators who recognized the need for developing ECBS as an engineering discipline founded ECBS centers in several universities. These ECBS pioneers included Dan Eckman, who founded the Systems Research Center at Case Institute of Technology in 1958; John Wescot at Imperial College; and Ted Williams at Purdue University. These early efforts focused on the process control industry.

The need for formalizing ECBS became obvious to some in the late 1980s. These advocates organized an IEEE Computer Society workshop in Neveh Ilan, Israel, in 1990. The workshop, led by Jonah Lavi and attended by 28 engineers from nine countries, addressed several issues:

- defining the ECBS discipline, its contents, and problematics;
- determining why ECBS is not widely recognized as an active field of teaching and practice; and
- promoting ECBS in industry and academia.

These efforts led to the establishment of an ECBS Task Force within the IEEE Computer Society in 1991. In November 1995, the Computer Society elevated the Task Force to a permanent Technical Committee.

Current activities

Formal ECBS conference proceedings published by the Computer Society since 1994 collectively represent an overview of the state of the art in ECBS. The TC allocates its work to several focus groups and has provided a reference definition of CBS engineering.¹ *Computer's* Integrated Engineering column often presents articles that address major CBS problems. Finally, curriculum proposals have been made that provide useful guidance in formulating ECBS programs at the master's and bache-

lor's degree levels.^{2,3}

The next conference and workshop will take place 8-11 April 2002 in Lund, Sweden. Further information can be obtained from <http://www.digital.com/conferences/ecbs02/> (current Oct. 2001).

References

1. S. White, "Computer-Based Systems Engineering," *McGraw-Hill Yearbook of Science and Technology*, McGraw-Hill, New York, 1997, pp. 108-110.
2. J. Lavi, B. Melhart, and I. Pyle, "Engineering of Computer-Based Systems—A Proposed Curriculum for a Degree Program at the Master Level," *Proc. Conf. and Workshop on Engineering of Computer-Based Systems (ECBS 97)*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 54-63.
3. J. Lavi et al., "Engineering of Computer-Based Systems—A Proposed Curriculum for a Degree Program at the Bachelor Level," *Proc. Conf. and Workshop on Engineering of Computer-Based Systems (ECBS 01)*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 344-356.

- Develop validation and test plans.
- Work with hardware, software, and communication engineers to develop integration environments.
- Integrate and test the CBS.
- Define the process and technologies required for CBS development and implementation.
- Develop CBS project plans and manage the projects, including assessing and evaluating their risks.
- Address the special problems associated with emergent behavior.
- Collaborate with engineers from other disciplines to develop the system in which the CBS is embedded.
- Maintain and evolve installed computer-based systems, including their legacy components.

PROBLEM AREAS

Several fundamental problem areas have arisen related to the state of the practice in engineering CBSs, most of which were described in a 1993 *Computer* article.⁴

Language. The use of inconsistent languages by different disciplines, especially in establishing requirements definition and design, leads to communication and traceability problems.

System-software engineering dialog. In too many cases, systems engineers do not understand what information they should provide to CBS implementers. They provide information in the wrong sequence or provide it in insufficient detail. Defining a dialog structure could solve these problems. For example, the dialog structure should define the correct level of information for determining system feasibility. This information is different from the level needed to design the system.

Static and dynamic models. Software engineers have a limited ability to exchange functional capability for resource utilization and performance. Researchers must focus on the nonlinearities that scaling up capability or data cause, and engineers must analyze or model scale effects.

Ramifications of systems engineering decisions. Systems engineers make high-level, architectural, or system-wide design decisions. These policy decisions should inform and constrain subsequent design and management decisions relating to various subsystems. How to present and propagate these key decisions—for example, monitoring subsystem design decisions to ensure that they do not conflict with system-level design decisions—remains unclear. Further, many high-level or system-level decisions result in major consequences to the CBS. Engineers do not understand these consequences when the decisions are made, which creates an acute need to determine what types of decisions have major consequences, the ramifications these decisions have, and how such decisions *should* be made.

Domain architectures. To keep complexity to a minimum, designers must base domain architectures on principles, concepts, and strategies that fit the problem class naturally. Standard domain architectures are appearing as building blocks for precedented systems. More are needed.

Design synthesis. Design environments do not adequately support design synthesis. Models must explicitly represent dependencies and constraints among various views. A modeling and simulation infrastructure should accommodate different levels of granularity so that the developer can work efficiently to answer questions about the model. Maintainability, availability, and scalability metrics must be available so that designers can properly evaluate these capabilities and perform trade-offs among these and other design properties. Environments must support detailed modeling, simulation, and analysis of the operational system working within its environment. Especially in high-assurance applications, engineers must carefully match software components to the context of their use. Documented disasters such as the Ariane 5 Flight 501 (<http://www.inria.fr/actualities-fra.html>) and the Therac-25 x-ray machine (N.G. Leveson and C.S. Turner, “An Investigation of the Therac-25 Accidents,” *Computer*, July 1993, pp. 18-41), driven by applications that incorporated reused software, demonstrate this need.

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

Costing. Existing cost models and literature on experiments can help determine software costs. Systems engineering and ECBS require similar resources. Although software cost modeling is widely practiced, the different cost models disagree and suffer from poorly documented underlying assumptions, hypotheses, and estimation techniques. Different models result in significantly different cost estimates, using 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-grained metrics for each phase, and they need to better represent the ECBS life-cycle process model. Work

In too many cases, systems engineers do not understand what information they should provide to CBS implementers.

Mapping multiple levels of software structure from the application via programming languages, middleware, operating systems, and hardware levels causes rampant complexity.

must be tracked to calibrate cost models. Industry usually can estimate the cost of predated jobs, but it struggles to accurately cost out unprecedented ones.

Platform stability. The instability of the hardware and the system software upon which many computer-based systems operate, and upon which CBS engineering methods and tools depend for support, presents a major challenge. Today's commercial platform systems are unnecessarily complex, leading to reliability, safety, security, availability, and maintainability problems. The complexities arise due to two factors. First, "feature-itis" results in the implementation of too many functions. Second, mapping multiple levels of software structure from the application via programming languages,

middleware, operating systems, and hardware levels causes rampant complexity. Unfortunately, CBS engineers must expend significant effort to deal with these platform-related complexities, which reduces the time available for adding valuable content to the system application. Thus, the availability of stable, verifiable hardware and system-software platforms is a vital ingredient in achieving successful CBSs.⁵

MEETING THE CHALLENGE

ECBS practitioners are making progress in addressing these problem areas and in meeting the general challenge associated with improving the state of CBS engineering practice. Janos Sztipanovits⁶ described the need for research in this area. The articles selected for this issue address several ECBS challenges.

Model-based engineering

That CBSs derive from different design perspectives—software, hardware, performance, safety, and reliability—constitutes a well-known problem. But these design perspectives are actually interdependent. By using integrated models, we can analyze the consequences of different design decisions within the system as a whole, gradually refining the system description into a hardware and software implementation. This approach, known as model continuity, allows continuous and incremental evaluation of the system. Important issues include determining the amount of detail required within a model to obtain reasonable results and supporting hardware and software modeling at different detail levels.

Although managing and tracking all interdependencies is beyond current capabilities, researchers have developed technologies that allow rigorous description and modeling of system, hardware, and software design perspectives and their interrelations.⁷ Indeed, developers have already successfully applied such technologies to large-scale projects.⁸

In "Composing Design-Specific Domain Environments," Ákos Lédeczi and colleagues describe an approach to model-based engineering using model-integrated computing. This topic is particularly relevant for specialized CBS domains—perhaps even single projects—and has led to significant reductions in development and maintenance costs on their projects.

Integrated architecture

CBSs are inherently heterogeneous. They combine application-specific and off-the-shelf software, hardware, and interface elements, each of which can have varying degrees of structural, functional, and behavioral complexity. Traditional engineering practice, which has evolved through experience, mandates that we develop the system, hardware, and software architectures as separate entities. This approach has led to cost and schedule overruns and to systems that fail to perform as intended. Today, developing all of a project's architectures synergistically through codesign, with an understanding that every system element will influence other elements, is increasingly important.⁹

Two articles address integrated architectures. In "Flexibility as a Design-Driver for Computer-Based Systems," Tommi Mikkonen and Peeter Pruuden describe their approach to late system architecture decision making, providing examples of CBSs whose development relied on flexible system design. "Exploring Embedded-Systems Architectures with Artemis" by Andy D. Pimentel and colleagues describes a modeling and simulation methodology for designing heterogeneous embedded systems architectures.

Systems composition strategies

Software typically serves as the glue in CBSs, coupling the physical processes into a single integrated system. Ensuring consistency across subsystems and facilitating the dynamic composition and recomposition of systems require highly automated system-software synthesis and integration technologies. These technologies include simulation-based design, methods for systematic handling of the many goals and constraints impinging on the design process, performance evaluation techniques, trade-off measurements, evaluation of multilevel multicomponent hierarchically specified models, and experimental frameworks.

In "Rosetta: Semantic Support for Model-Centered Systems-Level Design," Perry Alexander and Cindy Kong describe a language for modeling composition of heterogeneous components. The process begins by using smaller models—facets—to create system-level descriptions of individual pieces. Next, facet composition combines these elements into components, subsystems, and—ultimately—entire systems. This process provides abstraction that facilitates using a holistic development approach from a project's earliest stages.

Tool composition technology

The simple tools used to design systems a decade ago are no longer effective for designing current systems. Instead, we need integrated development environments. Unfortunately, the various design and verification tools use different model representations and operate at different abstraction levels. This tool incompatibility encourages engineers to develop hardware and software in isolation rather than as an integrated product. Current technology primarily addresses tool integration from the viewpoint of data transfer between tools. We need to replace this approach with semantic integration. Several articles in this issue describe methods that rely on tool composition, including the ISIS project that Ledeczki and colleagues discuss and the Artemis project that Pimentel describes.

Managing complexity

In “Virtual Design of Multiengineering Electronics Systems,” Mikko Kerttula and Timo Tokkonen suggest that supporting virtual reality CBSs will require sophisticated tools that are currently “on the drawing board.” They describe virtual reality prototyping and product design approaches that use a modeling strategy to deal with increasing product complexity in multitechnology products.

A major development in managing complexity, ISO/IEC 15288 System Engineering—System Life Cycle Processes,¹⁰ a new international standard, has reached final committee draft status and is expected to become an international standard in 2002. This standard will, for the first time, clearly identify the structure of systems and provide a holistic multidisciplinary approach to dealing with the processes related to system life cycles. The standard, in a generic manner, addresses several of the key CBS aspects, problems, and challenges we’ve identified. It is expected that this standard will be a primary technical complement to the ISO 9000:2000 quality standard. *

Acknowledgments

We thank all those who submitted interesting papers for this special issue—all these contributions relate directly to meeting the ECBS challenge. The active participation of the referees who assisted in evaluating these papers has been vital to selecting the few presented here.

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, Berlin, 1991.

2. B. Thome, ed., *Systems Engineering, Principles, and Practice of Computer Based Systems Engineering*, John Wiley & Sons, New York, 1993.
3. H.W. Lawson, “Philosophies for Engineering Computer-Based Systems,” *Computer*, Dec. 1990, pp. 52-63.
4. S. White et al., “Systems Engineering of Computer-Based Systems,” *Computer*, Nov. 1993, pp. 54-65.
5. H.W. Lawson, “From Busyware to Stableware,” *Computer*, Oct. 1998, pp. 117-119.
6. J. Sztipanovits, “Integrated Engineering of Computer-Based Systems,” *Computer*, Dec. 1998, pp. 68-69.
7. D.W. Oliver, T.P. Kelliher, and J.G. Keegan Jr., *Engineering Complex Systems with Objects and Models*, McGraw-Hill, New York, 1997.
8. E. Long, A. Misra, and J. Sztipanovits, “Increasing Productivity at Saturn,” *Computer*, Aug. 1998, pp. 35-43.
9. J.W. Rozenblit and S. Kumar, “Toward Synergistic Engineering of Computer Systems,” *Computer*, Feb. 1997, pp. 126-127.
10. ISO/IEC 15288 Systems Engineering—System Life Cycle Processes, Final Committee Draft, 2001-07-22, <http://www.15288.com> (current Oct. 2001).

Stephanie Farbman White is a full professor in the College of Information and Computer Science on the C.W. Post Campus of Long Island University. She is also president of System World, which develops systems-engineering technologies. Her research interests include systems engineering for software-intensive systems, requirements engineering, process modeling, and model-based analysis and design. White received a PhD in computer science from Polytechnic University in Brooklyn. She is a member of the ACM and the IEEE. Contact her at Stephanie.White@liu.edu.

Bonnie E. Melhart is associate dean in the College of Science and Engineering at Texas Christian University. Her research interests include parallel systems, dependability, and software quality. She received a PhD in information and computer science from the University of California, Irvine. She is a member of the ACM, the IEEE, and the IEEE CS Technical Committee on ECBS. Contact her at B.Melhart@tcu.edu.

Harold W. (Bud) Lawson is managing director of Lawson Konsult AB, Stockholm. He has contributed to several pioneering hardware-, software-, and application-related endeavors. Lawson received a PhD from the Royal Technical University, Stockholm. He received a Computer Society Computer Pioneer Award in 2000 for his invention of the pointer variable concept (1964-1965). Lawson is a Fellow of the ACM and the IEEE. Contact him at bud@lawson.se.

Tool incompatibility encourages engineers to develop hardware and software in isolation rather than as an integrated product.