The art of systems architecting

The design of complex systems must blend the art of architecture with the science of engineering

Few engineers would be surprised, today, to see the word "architecture" in their professional journals; nor would they have to think twice about its meaning. Architecture is understood to be the underlying structure of things—whether of buildings, communication networks, neural networks, spacecraft, computers, software or organization charts—systems all.

Less recognized, perhaps, is that where there are architectures, there must be architects. Moreover, there must be a process, "architecting," by which the architectures are created, designed, and built.

Architecting, itself, has ancient roots. It first appeared in Egyptian writings some 4000 years ago. Many of its basic principles were codified by the Roman, Vitruvius, in the 1st century B.C. But only in the last 50 years or so has the concept of "systems" been comparably formalized.

The merging of architecting and systems into systems architecting is still more recent. That process is accelerating, driven by three factors: the increasing architectural complexity and scope of global-sized projects and markets, the ubiquity of computers in virtually all modern systems, and the power of computer aids to system design.

Retrospectively, it is apparent that the success or failure of many defense, space, and civil systems of the last half century has depended in large part on how they were structured. The most successful ones were conceived, built, tested, certified, and operated in a way that ensured their integrity and performance. They were based on a consistent set of principles and techniques that were maintained throughout all phases of the project. And their designs were resilient enough to bend to the inevitable changes brought about by time and circumstance.

As civil architects would say, they had good bones. They also had fine architects who supervised their programs from beginning to end. The systems that have failed—whether technologically, politically, or economically—lacked these essentials.

Eberhardt Rechtin University of Southern California

Such conclusions are not new. Similar ones have been drawn over the centuries about pyramids, cathedrals, cities, ships, and fortifications. What is new is that those conclusions are now being seen as relevant to the engineering of electrotechnical systems.

Systems engineering and architecting are brought together by certain common principles. Each is concerned with complexity and reliability—systems engineering because of its nature and architecting because of its scope. The form of a system is strongly driven by the functions it is called upon to perform. Architecting defines that form by matching, fitting, balancing, and compromising proposed functions and forms until a practical result can be achieved.

SYSTEMS PRINCIPLES. Our first focus is on the principles of systems—an area more familiar to most engineers. Different people can mean different things by the word "system." For the purposes of architecting, a useful and sufficiently precise definition of a system is: "A collection of different things so related as to produce a result greater than what its parts, separately, could produce." Indeed, the purpose of building a system is to achieve that greater result.

Example: The system function of an assembled automobile is transportation, unavailable from the parts separately.

This definition of system has at least two important consequences. *First, all systems have subsystems and all systems are parts of larger systems.* Hence the systems world is inherently unbounded: no matter where a boundary might be drawn, things important to the system will exist outside it.

The same situation occurs in classical architecture.

Example: Designing a water faucet (a very small system, but a system nonetheless) means considering not only its own function, but also the demands of the systems within which it operates, such as the inclusion of urban water-usage restrictors to minimize the effects of drought on scenic lakes several hundred kilometers away.

Thus, complex systems cannot be architected, built, or operated in isolation. Their "outer" and "inner" worlds will always intrude. The best that can be done is to draw boundaries so that intrusions, when they occur, are secondary and not primary—that is, that they can be accommodated without breaking the system.

Example: The design of a manned spaceflight program, a strategic defense system, a national wideband communication network, an airline, or a light rail system is strongly influenced by extra-technical imperatives—economic, human, social, political, and international. These imperatives must be included in the design or the system will fail or abort, quite possibly well before completion. Placing them "outside" the system for design purposes could destroy any chance of success.

The architect's task is made particularly difficult by the fact that rarely, if ever, is there a single optimal solution for all parties and all circumstances. The objective, instead, is a kind of general satisfaction based on practicality, fit, balance, and compromise. As experienced architects will affirm, that takes both science and art.

The second architecturally relevant consequence of our definition of system is *the value added by a system must come from the relationships between the parts, not from the parts per se.* Each part already contributes its own inherent value to the system. But it is the total of all the parts working together that yields the whole system—a phenomenon sometimes known as synergism.

Example: The Douglas Aircraft DC-3, the first commercial airliner to make a profit for its owners, consisted of airfoils from the National Advisory Committee on Aeronautics (the predecessor to the U.S. National Aeronautics and Space Administration), monocoque designs already tried by the Boeing Aircraft Co., engines on the shelf, and control systems then in development. But, with modified elements in novel combination, the DC-3 made air travel efficient, reliable, and pleasurable.

It follows that *systems architects who work with systems must be specialists in relationships—not generalists who know a little bit about all the parts.* Their specialty is

Defining terms

Architecting: the process by which a system is created, designed, and built.
Heuristics: empirical rules of thumb derived from experience and judgment, useful for attacking problems too complex to be solved by analytical techniques alone.
System: a collection of different things related in such a way as to produce a result greater than what its parts, separately, could produce.
Systems architecture: the underlying structure of a system, such as a communication network, a neural network, a spacecraft, a computer, major software, or an organization.

0018-9235/92/$3.00©1992 IEEE

a concentration on the system as a whole: that is, on those elements, interfaces, and factors that have the most effect on overall system performance, cost, and schedule. Systems architects must necessarily know, or learn, a great deal about some details—those that impinge on the overall system—but need not, and probably should not, pay much attention to the rest, which are best handled by the subsystem experts with whom the systems specialists work.

Example: A launch vehicle systems engineer need not know the detailed design of a solid rocket motor. But that systems engineer would be expected to understand in some detail how the rocket's segments were connected, how each rocket was attached to others and to the payload, what its performance tolerances must be compared to other rockets on the same vehicle, what control authority is provided by its thrust control mechanisms, and so forth.

Without question, it requires expertise to know which details, interfaces, tradeoffs, and compromises count the most and which, the least. Poor choices can be disastrous; too many choices can be overwhelming.

ART AND SCIENCE. Much more so than engineering, architecting is an art as well as a science. There is an art to creating any structure, whether a building, a ship, a spacecraft, a network, or any other system. It is not just a figure of speech to praise a system as elegant or as having style.

The artistic element of architecting is most apparent when architects are asked how they create what they do, how they come up with alternatives out of the blue that withstand the scrutiny of analyses, and how they know that when all the parts come together, a system never built before will work to a client's satisfaction.

The usual, and not very helpful, answer is, "I just use common sense." Further inquiry leads to the discovery that what is really meant is contextual sense: doing what is sensible in the context of the problem. Commercial aircraft architects do it in creating a safe and profitable aircraft, spacecraft architects in producing a reliable and efficient spacecraft, and software architects in developing powerful and user-friendly software. And that means the use of empirical insights, tricks of the trade, and lessons learned from past successes and failures—that is, heuristics.

The art in architecting is a special process, essential in treating situations too complex for analysis. It evolved centuries ago as a way to handle ill-structured problems with all their uncertainties, unknowns, conflicting requirements, and sociopolitical imperatives—problems typical of complex systems.

At the risk of oversimplification, discipline-oriented engineering is deductive, analytical, and rational, while systems-oriented architecting is inductive, intuitive, synthetical, and pragmatic. At one extreme are the powerful applied science tools of engineering; at the other are the often personal arts of architecting. Straddling both is the practice of systems engineering.

Architects of buildings who have studied the process of architecting—how architects work—have identified four methods in common use that depend on the nature and phase of the project, the particular problem to be solved, and the style of the architect: ***Normative methodology*** relies on standard, quantitative solutions based on subjective value judgments ("good" vs. "bad" practice). Building codes, communication protocols, and design handbooks are examples. ***Rational methodology*** is based on quantitative analysis and algorithms that tell how to find a solution, but not what it is. The scientific method of data gathering, hypothesis, and test is an example. Calculus is another. Rational methodology—the mainstay of systems engineering—is intended to be as objective as possible. Optimization is one of its goals.

Argumentative methodology is based on broad participation of interested parties. Brainstorming is one of its techniques; quality circles is another. This methodology aims for imaginative consensus. Its strength is group commitment to a common goal.

Heuristic methodology is based on common sense or rules of thumb derived from experience and judgment. The law of supply and demand is an example from economics. Murphy's Law is an example from system design. The aim here is reasonable, satisfactory solutions and an avoidance of system-level disasters. More than the other three methodologies, the heuristic methodology is an art.

The normative and rational methodologies are widely taught in engineering schools and used extensively in practice. These methods have powerful tools available—statistics, probability theory, modeling, optimization, tradeoff charts, simulation, statistics, operations research, and performance analysis. They help break down problems into solvable subproblems whose solutions are then integrated into the whole. System certification is quantitative and not easily disputed. These two methodologies comprise the "science" part of architecting and the technical foundation of systems engineering, detailed design, and integration.

In argumentative methodology, free-ranging discussion reigns. Its weakness is design by committee. On the surface, it seems to contradict a widely held view that the best architectures are the product of a single mind or small team. Success, therefore, requires good team dynamics—a well-designed human system guided by the empirical knowledge of human behavior. In this respect, the argumentative methodology might be considered a managerial variation of the more general heuristic method.

HEURISTICS UNBOUND. The heuristic methodology is particularly characteristic of architecting. In contrast with the other methodologies, it is synthetical, inductive, and experiential. Its tools are heuristics—rules of thumb for discarding out of hand unreasonable options, for maintaining the integrity of system goals, for taking precautions against pitfalls ahead, and for recalling lessons learned.

But most important, heuristics is the only methodology that

Some heuristics for building a system

The conceptual phase:

- The choice between architectures may well depend upon which set of drawbacks the client can handle best.
- Extreme requirements should remain under challenge throughout system design, implementation, and operation.
- Don't assume that the original statement of the problem is necessarily the best or even the right one.
- No complex system can be optimum to all parties concerned, nor all functions optimized.
- A model is not reality.
- Complex systems will develop and evolve within an overall architecture much more rapidly if there are stable intermediate forms than if there are not.
- Build in and maintain options as long as possible.
- Don't make an architecture too smart for its own good.

The build and test phases:

- The product and process must match.
- An element good enough in a small system is unlikely to be good enough in a more complex one.
- Within the same class of products and processes, the failure rate of a product is linearly proportional to its cost.
- High-quality, reliable systems are produced by high-quality architecting, engineering, design, and manufacture, not by inspection, test, and rework.
- Regardless of what has gone before, the acceptance criteria determine what is actually built.
- To be tested, a system must be designed to be tested.
- Qualification and acceptance tests must be both definitive and passable.
- The cost to find and fix a failed part (or software bug) increases by an order of magnitude as that part is successively incorporated into higher levels in the system.

The operations phase:

- Before the flight, it's opinion. After the flight, it's obvious.
- The first quick-look failure analyses are often wrong.
- For every competitive system, there is a countersystem.
- Success is defined by the beholder, not the architect.
- There's nothing like being the first success.

Systems Architecting: Creating and Building Complex Systems, by Eberhardt Rechtin, published by Prentice Hall, Englewood Cliffs, N.J., 1991.

directly attacks problems too complex to be solved by analytical techniques alone; that is, those characteristic of unbounded systems.

A good example of a descriptive heuristic, mentioned earlier, is Murphy's Law: "If something can go wrong, it will." An associated prescriptive heuristic, an antidote to Murphy's Law, is: "Simplify, simplify, simplify." Neither of these mentions statistics or statistical quality control. Instead, they suggest a strategy: if a system is built in such a way that something could possibly go wrong, no matter how improbable, fix it.

An older heuristic, often used for simpler products, is: "If it ain't broke, don't fix it"—a strategy demonstrably not competitive for large-scale, complex products. It is not competitive primarily because an element "good enough" in a small system is unlikely to be good enough in a more complex one. (As more and more parts are added, the system reliability will go down unless the individual part reliabilities go up.)

The practical value of heuristic insights is seen in their extensions by W. Edwards Deming, Joseph M. Juran, Genichi Taguchi, and others. These are known as total quality management (TQM), continuous measurable improvement (CMI), just-in-time (JIT) inventories, and other techniques.

A quite different heuristic, useful in aerospace and computer design, is: "In partitioning a system into subsystems, choose a configuration with minimal communications between the subsystems." With a few word changes, this could be applied to the design of communication networks, organizations, and system subcontracts.

Example: A common question in the design of complex, smart spacecraft is whether to use a centralized or distributed computer system to run the major subsystems, such as propulsion, guidance, control, command, telemetering, science instrumentation, and system test. The centralized configuration is generally lighter, smaller, computationally more efficient, and less power consuming. The distributed configuration requires less information transfer, and solves local problems locally. But most important, it permits each subsystem to be self-testing without recourse to a central control unit. As such, it enables subsystem subcontractors to deliver checked-out units to a prime contractor. To do likewise, a centralized configuration requires central computer copies at each subcontractor site and/or delivery of subsystems that can be checked out only at the system level—a contracting nightmare and a prescription for overruns and delays. Fixing such problems usually requires adding weight, space, power, and computer capacity late in the development. The decentralized configuration is now preferred, for management and not technical reasons.

Heuristics cannot promise that a heuristically designed system will be the best performing of all possible systems. But, from experience, that type of system will be much

Teaching systems architecting: science and art

Instructing others in systems architecting involves both its science and its art. Teaching the underlying science is straightforward; teaching the art is still in the experimental stage. Existing guidelines are few, even from classical architecture.

What does seem to be true, though, is that until the science of systems architecting is understood and the complexities of systems experienced, the role of its art is little appreciated. Consequently, at the University of Southern California (USC) in Los Angeles, systems architecting is taught only to engineers with three or more years of experience (the average is seven years)—those who already recognize that the science of engineering, though powerful indeed, is somehow not sufficient to meet the demands they face in building complex systems.

The challenge is to teach the art without crushing its creative core under a burden of memorized dictates and caveats.

CODIFYING COMMON SENSE. Fortunately and perhaps surprisingly, codifying the common sense of successful systems architects is not so difficult. The first step is to show students that what is meant by common sense is contextual sense: what seems sensible depends on the system's particular context (whether the system is a building, an aircraft, a spacecraft, a missile, a computer, software, a network, or an organization).

That means that a heuristic (an empirical rule of thumb) that applies in one context may not apply in another. True, a heuristic that is sensible in one context may be sensible in another, but showing applicability has to be by example to be credible. It cannot be deduced mathematically. Instead, a proposed heuristic, on presentation to people versed in a field, has to seem "reasonable" to them and then must survive their almost automatic mental search for supporting or contradicting examples.

A diligent search for useful heuristics reveals a wealth of wisdom and lessons learned in many fields. Once given the concept of heuristics as architectural aids, an alert student can spot statements of common sense in most articles describing successful or unsuccessful systems.

At USC, well over 100 heuristics have been found or newly articulated for the acquisition of electrical and aerospace systems. Comparable numbers no doubt can be found scattered through the literature of computers, software, manned space flight, shipbuilding, law, economics, and management.

Though it might be wished that only a handful of heuristics existed that might be used as a checklist for all occasions, that does not seem to be the case. Different heuristics apply to the different phases of system acquisition—conceptualization, engineering, design, production, test, certification, and operations.

Basically, architecting is both multidimensional and relational; it has many parts, some of them related to several others. Its problems come in sets, not singly or in sequence. Similarly, its practitioners are multifaceted, broad-ranging, "renaissance" people. Like systems, they seem to have no bounds to their thinking and inquiry. If heuristics are to be taught, they have to be taught not as a bounded set of rules, but as a technique, an abbreviated form of experience and the starting point for creativity.

One technique of instruction might be the one commonly used for teaching law, economics, and conventional architecture: case studies. But the context of the system poses a practical difficulty. Doing a detailed case study of an aircraft system might not be of much help to a telecommunications architect and vice versa. Unless the heuristic fits the student's system and context, it will not be remembered when it counts. And in a typical USC class of graduate students—systems engineers from local aerospace and electronic firms—there may be as many systems of interest as there are students.

A CASE IN POINT. In the classes in systems architecting at USC, each student is asked to choose a system of personal interest, using it as a frame of reference to which the ideas in the course can be attached. Each student thus develops a personal case study as the course progresses. Although the technique works well, it does require that the student know at least one system in some detail. For the time being, at least, that confines it to the graduate level and for practicing engineers.

The hundred-plus heuristics studied so far range from the energy management system of the General Motors Corp.'s upcoming electric car to the U.S. government's Strategic Defense Initiative. And the insights the students have brought to the class—both their own and those of the architects they interviewed—have contributed substantially to the field of systems architecting.

Better yet, the students have been able to apply the lessons learned back on their jobs by assessing and modifying the systems studied. One fact about heuristics has become evident: whereas the science of architecting can be used by people of many skill levels, using its art effectively takes experience and judgment in context.

That result might have been expected; it is characteristic of any art. No one becomes a musician by studying mathematics or drawing, much as those three arts might be intertwined philosophically.

USC has now established a formal Master of Science degree in systems architecture and engineering. The utility of the program has been validated by the number of graduate students enrolling in it and by executive-level endorsement from major companies in southern California.

The program consists of systems architecting theory and practice complemented by analytic courses in each student's specialty, such as aerospace and mechanical systems, automation and control systems, communication and signal-processing systems, computer and information systems, and construction and manufacturing systems.

USC's experience in this field has demonstrated that the tools for the art of architecting systems can be taught, that they can be applied effectively, and that they will appreciably shorten the time needed for students to become professional systems architects. —*E.R.*

less likely to encounter unpleasant surprises down the road. More resilient to changes in its technical, budgetary, political, and competitive environment, it is more likely to be "good enough" (satisfactory) and less likely be "the best possible" (optimum)—if indeed a "best" or optimum exists at all.

Why might an optimum not exist? Two relevant heuristics tell why. First, "No complex system can be optimum to all parties concerned, nor all functions optimized." Secondly, "A system tightly optimized for a particular competitive situation is unlikely to be presented with that situation (the competition knows better than to do so!)."

CRADLE TO GRAVE. Architecting does not end with a first rough sketch, but continues through production into operation. In many systems that have been the architectural successes of this century, such as the Apollo moon project, the original architects were active for the entire project.

Lacking such architecting continuity, systems integrity would have been threatened, early consideration of likely events and problems down the road might not have occurred, controversies over system test and acceptance would have been exacerbated, and accountability for operations would have been diffused.

Direct participation by the architect is most critical during two phases of system building: conceptualization and certification. Two heuristics succinctly say why. For conceptualization: "All the serious mistakes are made in the first day." And for certification: "Regardless of what has gone before, the acceptance criteria determine what is actually built." The architect's role is in establishing acceptance criteria at the outset and certifying compliance with them at the end.

During the other phases—system engineering, detailed design, development, production, diagnosis, and operation—the architect is a monitor, advising the client on how well the system is conforming to plan at each stage, suggesting actions that would benefit the system as a whole, and helping the system accommodate to change.

COMPLEMENTING STRATEGIES. Architecting is an ongoing process—one that continues until the system is acquired. It is not just a front-end design-and-proposal effort tossed over the transom to the next group.

In this respect, architecting historically predates and affirms the principles of concurrent engineering, total quality management, continuous measurable improvement, and just-in-time inventory. Like architecting, those strategies also call for bringing together design, engineering, and production in the interest of better products and systems.

There are, nonetheless, notable differences between them and architecting. Compared with concurrent engineering, architecting is broader in its scope, covering higher-level system concepts and focusing less on detailed design. Also, its organizational structure of a small team contrasts with the collective style of concurrent engineering, and it places heavier emphasis on the certification (buy-off) process. On the other hand, concurrent engineering, which is valuable in making designers more aware of production problems, postulates an ongoing product line and generally puts to one side questions of concept, certification, public policy, and post-sale accountability.

> The success or failure of many civil and defense systems depends mainly on their architecture

Thus, concurrent engineering and architecting complement each other. They might even be combined. For example, the architect could act as the chairman of the concurrent-engineering group, maintaining overall systems integrity and balancing the conflicting demands of design, engineering, and production.

Quality management strategies such as total quality management, continuous measurable improvement, and just-in-time inventory are based on normative rules for processes to make similar products with an established architecture. In contrast, architecting creates something new, an original effort for which data (and solutions deduced from the data) are unavailable, established procedures may be inadequate, and clients' needs are ill-formed, inconsistent, and conflicting—even into operation.

STAKES ARE HIGH. Staying up-to-date in systems architecting requires an awareness of ongoing research in the field. In today's competitive world, learning and teaching cannot remain static. Not only must individuals participate in lifelong learning, but instruction also must advance based on several promising areas of research:

- Techniques for assessing architectures, architecting, and architects prior to, during, and after system construction. Application: management.
- The behavioral and functional profiling of architects and architectural teams for creating effective architectures. Application: hiring personnel and forming compatible teams.
- The effects of the political process on system architecture and design; for example, changes in taxation, regulation, privatization, national security policy, and space exploration mission priorities. Application: government.
- The design of ultraquality systems whose failure rate is intended to be so low as to be statistically unmeasurable prior to use, as for spacecraft to the planets and for nuclear power plants. Application: product and process certification.
- The design of purposefully opposed systems, such as opposing weapons and cryptographic systems. Application: defense and security systems.
- Biologically inspired architectures for intelligent machines, such as neural networks, associative memories, and artificial intelligence languages. Application: designers of smart systems and their software.

All in all, the future of systems architecting and instruction in it is exceptionally promising. Technological developments and programs of national scope are increasing the need, which is felt by both individuals and companies.

TO PROBE FURTHER. Perhaps the first text to relate the process of architecting to that of systems and their engineering is *Systems Architecting: Creating and Building Complex Systems*, by Eberhardt Rechtin, published by Prentice Hall, Englewood Cliffs, N.J., 1991.

A classic book of heuristics—and satire—applicable to defense systems acquisition is *Augustine's Laws*, by Norman R. Augustine, published by the American Institute of Aeronautics and Astronautics, Washington, D.C., 1982.

A famous system architect-engineering text, applicable well beyond its own field, is *The Mythical Man-Month, Essays on Software Engineering*. by Frederick P. Brooks Jr., published by Addison-Wesley, Reading, Mass., 1982.

ACKNOWLEDGMENTS. The work reported here has been supported by the School of Engineering of the University of Southern California in Los Angeles, with encouragement and student tuition support from southern California aerospace companies, especially Aerospace Corp., Hughes Aircraft, the National Aeronautics and Space Administration/California Institute of Technology's Jet Propulsion Laboratory, Northrop, Rockwell International, and TRW. Key insights have been provided by Harry Hillaker, Gerald Nadler, Robert Spinrad, Albert D. Wheelon, and dozens of students.

ABOUT THE AUTHOR. Eberhardt Rechtin (F) is president-emeritus of Aerospace Corp., El Segundo, Calif., and professor of engineering at the University of Southern California in Los Angeles with appointments in electric systems, aerospace, and industrial and systems engineering.

Rechtin has been an assistant director of the California Institute of Technology's Jet Propulsion Laboratory in Pasadena, a director of the Defense Advance Research Projects Agency, an assistant secretary of defense (telecommunications), and the chief engineer of Hewlett-Packard Co. He was the chief architect and director for the National Aeronautics and Space Administration of the Jet Propulsion Laboratory's Deep Space Network for its first 10 years of construction and use in tracking spacecraft. ●