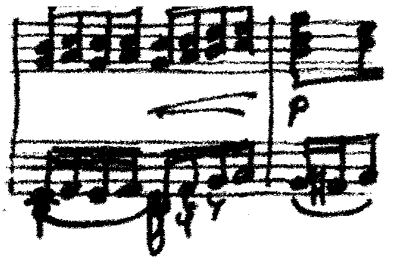


Anchoring the Software Process

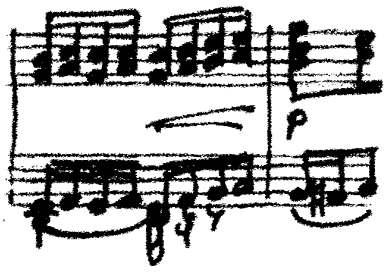
BARRY BOEHM, *University of Southern California*

Software organizations need common milestones to serve as a basis for their software development processes. The author proposes three such milestones, gives an example of their use, and discusses why they are success-critical for software projects.

For a few golden moments in the mid-'70s, it appeared that the software field had found a set of common anchor points: A sequence of milestones around which people could plan, organize, monitor, and control their projects. These were the milestones in the waterfall model. They typically included the completion of system requirements, software requirements, preliminary design, detailed design, code, unit test, software acceptance test, and system acceptance test.[1] These milestones let companies, government organizations, and standards groups establish a set of interlocking regulations, specifications, and standards that covered a full set of software project needs.

Unfortunately, just as the waterfall model was becoming fully elaborated, people were finding that its milestones did not fit an increasing number of project situations. For example, the ideal of a complete, consistent software requirements specification ran into the following problems:

0740-7459/96/$05.00 © 1996 IEEE



♦ *A prototype is worth 100,000 words.* Written requirements specifications trying to describe the look and feel of a user interface were nowhere near as effective as a user-interface prototype.

♦ *Gold plating.* Fixed requirements specifications in advance of design tended to encourage elaborate additions. Asked about their requirements, users would often reason, "I don't know if I'll need this feature or not, but I might as well specify it just in case."

♦ *Inflexible point solutions.* Fixed requirements specifications also tended to produce point solutions optimized around the original problem statement. These solutions were frequently difficult to modify or to scale up to meet increased workload levels.

The primary initial response to the waterfall model's problems was *evolutionary development.*[2] The central milestones here are the releases of increments of system capability; new content is determined from experience with earlier system releases. The critical milestone is thus the initial release: a package of software with sufficient capability to serve as a basis for user exercise, evaluation, and evolutionary improvement. However, this "initial release" milestone frequently suffered from three major problems.

♦ *Inflexible point-solutions.* Frequently, the initial release is optimized for initial demonstration- and exploratory-mode success. For example, it may store everything in main memory to provide rapid response time. Then, when users want to transition to large-scale use, the initial point-solution architecture will not scale up.

♦ *High-risk downstream capabilities.* The initial release often defers considerations such as security, fault tolerance, and distributed processing in favor of providing early functionality and user interface capabilities. The users may like the results and expect the deferred considerations to be delivered equally rapidly. This often puts the project in big trouble because the initial release's architecture cannot be easily extended to support these other key considerations.

♦ *Off-target initial release.* Evolutionary developers often begin by saying, "Let's find out what the user needs by building an initial release and seeing what the users want improved." The lack of initial user-activity analysis frequently leads to a first release that is so far from user needs that they never bother to learn or use it.

The difficulties with the waterfall and evolutionary-development models have led to the development of several alternative process models, such as risk-driven, reuse-driven, legacy-driven, demonstration-driven, design-to-cost or -schedule, incremental, as well as hybrids of any of these with the waterfall or evolutionary-development models. This proliferation has made it difficult for software organizations to establish a common

RECENT INITIATIVES AND THE KEY MILESTONES

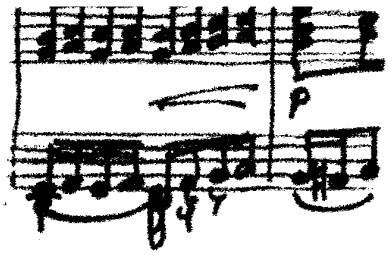
Recent software process initiatives have provided guidelines that make it easier to depart from lock-step software processes. These initiatives—such as Mil-Std-498[1] (now evolving into EIA/IEEE J-Std-016)[2] and ISO/IEC Standard 12207[3]—are compatible with the life-cycle objectives, life-cycle architecture, and initial operational capability milestones.

J-STD-016. This standard supersedes DoD Std-2167A and Std-7935A, which largely focused DoD projects on waterfall-model software processes. J-Std-016 avoids lock-in to the waterfall model by focusing on required software activities rather than phases, stating that activities "may overlap, may be applied iteratively, may be applied differently to different parts of the software, and need not be performed in the order listed below." It provides examples of its application to waterfall, incremental, and evolutionary processes, with a guidebook that offers more detailed usage examples and tailoring guidelines.

J-Std-016's guidance on system requirements analysis and system architectural design are quite consistent with the LCO milestone. For example, system requirements analysis involves user-input analysis, operational-concept definition, and iterative application of requirements analysis and design. Its guidance on software requirements analysis and software design are compatible with the LCA milestone, but the standard misses several opportunities to emphasize the coupling of software architecture to anticipated requirements evolution and to establish the feasibility rationale as a first-class citizen. It requires the recording of global design decisions, but relegates their rationale to the "notes" sections in the system and software design data-item descriptions.

J-Std-016 is also an advance from previous DoD standards in that it covers activities involved in proceeding from software configuration item-acceptance tests to the equivalent of the IOC milestone. The standard's guidelines for application to incremental and evolutionary development also show how these apply to the IOC milestone.

ISO/IEC 12207: IT life-cycle processes. The ISO/IEC 12207 standard is similar to J-Std-016 in that it focuses on activities and core processes—acquisition, supply, development, operation, maintenance, life-cycle process support, and organizational life-cycle processes—that can be performed sequentially, repeated, and combined according to the project's choice of life-cycle



frame of reference and common milestones for software life-cycle planning, measuring, controlling, and communicating with external organizations. In many cases, organizations have remained loyal to admittedly flawed models — such as the waterfall — because they believe that the value of any common framework is worth the price of its imperfections.

ANCHORING THE PROCESS: THREE MILESTONES

Since 1988, when I first published an article on the spiral model,[3] I have reviewed many results of its implementation. Several of these implementations were effective; several were flawed. A not-too-extreme example of the latter? "We decided that using the spiral model meant that we didn't have to write anything down, so now everybody is off doing different things and we don't know how to pull them all together."

In analyzing the various results, I discovered that one of the most consistent correlates of success versus failure was the degree to which the projects employed the equivalents of three critical milestones:

- life-cycle objectives,
- life-cycle architecture, and
- initial operational capability.

The management team I established for the US Defense Department's Software Technology for Adaptable, Reliable Systems program successfully used the equivalents of these milestones. STARS created a set of software environment life-cycle process and software asset library capabilities supporting software reuse and product-line management. As the experience with STARS shows, and the box on page 76 explains, LCO, LCA, and IOC can anchor not just individual projects, but the management of software product lines with domain architectures and reusable components. The three milestones are also compatible with recent process standards initiatives, as the box on page 74 explains.

Life-cycle objectives. As Table 1 shows, the key element of the LCO milestone is stakeholder concurrence on the system's objectives.

Top-level system objectives. To establish the LCO's top-level system objectives, the system's key stakeholders must operate as a team to determine the system boundary by making key decisions on what will and will not be included in the system. The part that will not be included will therefore be in the system's environment: key parameters and assumptions on the nature of users, data volume and consistency,

models. Example models cited in this regard are waterfall, evolutionary builds, preplanned product improvement, and spiral.

ISO/IEC 12207's provisions for system requirements analysis and architectural design are consistent with the LCO milestone. This standard goes beyond J-Std-016 in emphasizing the need to co-define the system requirements and architecture and to document the results of feasibility evaluations. It includes, significantly, the "feasibility of system architectural design" as a requirements-evaluation criterion and "traceability to" and "consistency with" system requirements as architectural design-evaluation criteria. The treatment of software requirements analysis and architectural design activities is similar and consistent with the LCA milestone.

ISO/IEC 12207 is also consistent with the IOC milestone in its accommodation of builds or increments: its culminating development-process activities are "software installation" and "software acceptance support." Overall, ISO/IEC 12207 goes farther than J-Std-016 in countering the problem areas associated with the waterfall and evolutionary-development milestones. However, it also misses some opportunities to integrate the architectural rationale into the architecture, to include risk resolution as an architecture-evaluation criterion, and to emphasize the most likely directions of requirements change as an integral part of the requirements.

REFERENCES

1. *Military Standard 498: Software Development and Documentation*, US Dept. of Defense, Dec. 1994; available via Defense Printing Service, Philadelphia, PA 19111-5094.
2. *Trial Use Standard J-Std-016-1995: Software Life-cycle Processes*, EIA/IEEE, Dec. 1995; available via Global Engineering, 1-800-854-7179.
3. *International Standard ISO/IEC 12207: Information Technology — Software Life-Cycle Processes*, Int'l Standards Org., Aug. 1995; available via Global Engineering, 1-800-854-7179.

SOFTWARE PRODUCT-LINE MANAGEMENT

If your organization applies the LCO, LCA, and IOC milestones separately to each individual software project, it will get a suboptimal outcome: a series of separate "stovepipe" systems with many redundantly developed and incompatible components. To achieve the cost, schedule, and quality benefits of software reuse, you need to develop a software product-line management approach. This involves extending the definitions of the LCO and LCA milestones.

For the LCO milestone, you need to determine the breadth of the product-line domain across which reusable components will be shared (an example set of breadth choices is transaction processing, message processing, military message processing, or military medical message processing). For the LCA milestone, you need to develop a domain architecture for the product line, rather than just a life-cycle architecture for an individual system.

workload levels, interoperating external systems, and so on. These should be characterized not just at their initial operating levels, but in terms of their likely evolution, to avoid the point-solution difficulties.

Operational concept. To formulate the operational concept, stakeholders work through scenarios[4] of how the system will be used. These scenarios may involve prototypes, screen layouts, dataflow diagrams, state transition diagrams, or other relevant representations. If the ability to perform in off-nominal situations (component failures, crisis situations) is important, you should develop scenarios for these as well. You should also work out scenarios for software and system maintenance and determine which organizations will be responsible for funding and performing the various functions. These organizations are some of the key stakeholders; their concurrence is needed for realistic and supportable system definitions.

System requirements. Unlike the waterfall or related contract-oriented models, the system requirements here are not cast-in-concrete specifications. Instead, you use them to record the collective stakeholders' concurrence on essential system features, the details of which can be modified easily and collaboratively as new opportunities (reuse opportunities, strategic partners), problems (budget cuts, technical difficulties), or developments (reorganizations, divestitures) arise.

System and software architecture. The architecture definitions should be sufficiently detailed to support analysis of the architecture's feasibility in supporting system objectives and requirements. Having more than one feasible choice of architecture is acceptable at the LCO stage; you could have, for example, two workable central commercial-off-the-shelf products with different architectural implications. However, if you can't show any architectural option to be feasible, you should cancel the project or rework its scope and objectives. Also, keep a record of infeasible options that were considered and dropped to ensure that others don't adopt them in ignorance later.

Life-cycle plan. In your initial life-cycle plan, identify the major stakeholders in the system: they are often the system user, customer, developer, and maintainer organizations. If the system is closely coupled with another system, the interoperator organization is also a key stakeholder. If system safety, privacy, or other general-public issues are important, you should include a representative of the general public as a stakeholder. Without the concurrence of these stakeholders on system requirements, the system may not reflect their needs and will not be a success.

Another critical point of the life-cycle plan is to identify the process model or models to be used (such as waterfall, evolutionary, spiral, incremental, design-to-cost or -schedule, or a hybrid). For the main part of the life-cycle plan, you need an organizing principle that scales down to provide simple plans for simple projects. A good approach is the WWWWWHH principle, which organizes the plan as follows:

- Objectives: *Why* is the system being developed?
- Milestones and schedules: *What* will be done by *when*?
- Responsibilities: *Who* is responsible for a function? *Where* are they organizationally located?
- Approach: *How* will the job be done, technically and managerially?
- Resources: *How much* of each resource is needed?

By using this approach, you can pack the essential decision content of a life-cycle plan for a small, straightforward project into one page or two briefing charts.

Feasibility rationale. The most important thing that you need to achieve for the LCO milestone is the conceptual integrity and compatibility of all the milestone's components. The element that assures you can do this is the feasibility rationale. With it, you use an appropriate combination of analysis, measurement, prototyping, simulation, benchmarking, or other techniques to establish that a system built to the life-cycle architecture and plans can support the system's requirements and operational concept. Another key element is the business case analysis, which establishes whether or not the system can generate enough business value to be worth the investment. (A defense sector counterpart is the cost and operational effectiveness analysis, or COEA.)

Life-cycle architecture. As Table 1 shows, most of the LCA elements are elaborations of the LCO elements. The critical element of the LCA milestone is the definition of the system and software architecture itself. This consists of defining the system and software components (either a hardware component, a computer program, a data ensemble, or a combination of such items), connectors (elements that mediate interactions among components), configurations (combinations of components and connectors), and constraints (such as resource limitations and shared assumptions about the operating environment). Mary Shaw and David Garlan provide an excellent treatment of software architectures.[5]

Other key features of the LCA milestone are

- specifics of commercial-off-the-shelf and reused software choices,

TABLE 1. ELEMENTS OF CRITICAL FRONT-END MILESTONES

Milestone element	Life-cycle objectives (LCO)	Life-cycle architecture (LCA)
Definition of operational concept	♦ Top-level system objectives and scope -System boundary -Environment parameters and assumptions -Evolution parameters ♦ Operational concept -Operations and maintenance scenarios and parameters -Organizational life-cycle responsibilities (stakeholders)	♦ Elaboration of system objectives and scope by increment ♦ Elaboration of operational concept by increment
Definition of system requirements	♦ Top-level functions, interfaces, quality attribute levels, including: -Growth vectors -Priorities ♦ Stakeholders' concurrence on essentials	♦ Elaboration of functions, interfaces, quality attributes by increment -Identification of to-be-determined items (TBDs) ♦ Stakeholders' concurrence on their priority concerns
Definition of system and software architecture	♦ Top-level definition of at least one feasible architecture -Physical and logical elements and relationships -Choices of COTS and reusable software elements ♦ Identification of infeasible architecture options	♦ Choice of architecture and elaboration by increment -Physical and logical components, connectors, configurations, constraints -COTS, reuse choices -Domain-architecture and architectural style choices ♦ Architecture evolution parameters
Definition of life-cycle plan	♦ Identification of life-cycle stakeholders -Users, customers, developers, maintainers, interoperators, general public, others ♦ Identification of life-cycle process model -Top-level stages, increments ♦ Top-level WWWWWHH* by stage	♦ Elaboration of WWWWWHH* for IOC -Partial elaboration, identification of key TBDs for later increments
Feasibility rationale	♦ Assurance of consistency among elements above -Via analysis, measurement, prototyping, simulation, and so on -Business case analysis for requirements, feasible architectures	♦ Assurance of consistency among elements above ♦ All major risks resolved or covered by risk-management plan

* WWWWWHH: Why, What, When, Who, Where, How, How Much

which often drive both the architecture and the requirements;

♦ specifics of quality attribute levels such as response time, reliability, and security, which are also significant architecture drivers; and

♦ identification of likely directions of architectural evolution, which reduces the chance that the architecture will become obsolete.

As with the LCO milestone, the most important things stakeholders should achieve with the LCA milestone are

♦ a feasibility rationale, which establishes the consistency and conceptual integrity of the other elements, and

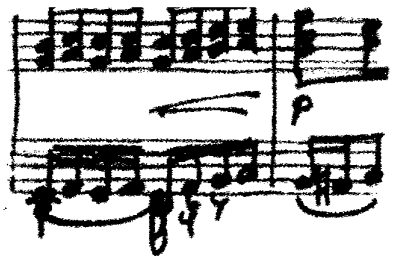
♦ stakeholders' concurrence that the LCA elements are compatible with their objectives for the system.

The LCA milestone differs from the LCO milestone in that you must have all the system's major risks resolved or at least covered by an element of the system's risk management plan. For large systems, when you pass the LCA milestone you significantly escalate both staff level and resource commitments. Proceeding to this stage with major risks unaddressed has led to disasters for many large projects. Several good guidelines are available for software risk assesment.[6-8]

I can't overemphasize how critical the LCA milestone is to your project and your career. If you haven't satisfied the LCA milestone criteria, *do not* proceed into full-scale development. Reconvene the stakeholders and work out a new project plan that will successfully achieve the LCA criteria.

LCO/LCA: Distinguishing features. The LCO and LCA milestones are distinguished from most current software milestones in that they provide a rationale for project success that lets them serve as anchor points across many types of software development.

♦ Their focus is not on requirements snapshots or architecture point solutions, but on requirements and architectural specifications that anticipate and accommodate system evolution. This is



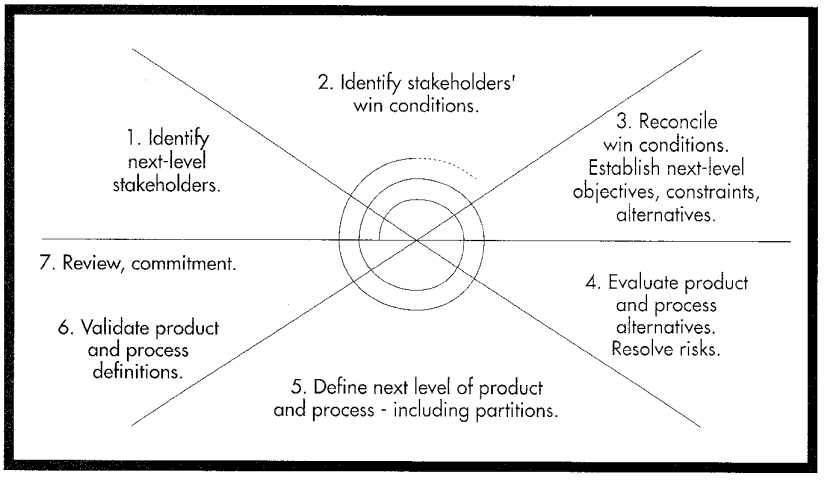


Figure 1. *The Win-Win spiral model.*

the reason for calling them the "life cycle" milestones.

- Elements can be either specifications or executing programs with data (such as prototypes, or COTS products).
- Specifications are driven by risk considerations rather than completeness considerations. Critical interface specifications should be complete or you will face integration risks. However, you should not try for complete written specifications of user interfaces because they are generally less risky to define via prototypes.
- The milestones are not peculiar to a single process model. You can move successfully from an LCO to an LCA via a waterfall, spiral, evolutionary, or COTS-driven process.
- The feasibility rationale is an essential element rather than an optional add-on.
- Stakeholder concurrence on the milestone elements is crucial because it establishes mutual stakeholder buy-in to the plans and specifications, and enables a collaborative team approach to unanticipated setbacks rather than the adversarial approach in most contract models.

IOC. At the start of the development cycle, if you skip or err on any part of the LCO or LCA milestones there are serious consequences. At the end of the development cycle, the IOC is the milestone with the most serious consequences of neglect. It can help you avoid the possibility of offering users a new system that has ill-matched software, poor site preparation, or poor user preparation—all of which are frequent sources of user alienation and killed projects.

The IOC's key elements are

- software preparation, including operational and support software with appropriate commentary and documentation, data preparation or conversion, the necessary licenses and rights for COTS and reused software, and appropriate operational readiness testing;
- site preparation, including facilities, equipment, supplies, and COTS vendor-support arrangements; and
- user, operator, and maintainer preparation, including selection, team-building, and training for usage, operations, or maintenance.

The nature of the IOC milestone is also risk-driven with respect to the system objectives determined in the LCO and LCA milestones. For example, these objectives drive the trade-off between IOC date and product quality (such as that between the safety-critical space shuttle software and a market-window-critical commercial software product). However, the difference between these two cases is narrowing as commercial vendors and users increasingly appreciate the market risks involved in buggy products.[9]

As with LCO and LCA, the IOC milestone is compatible with multiple classes of processes. It can be preceded by combinations of hardware-software integration, alpha testing, beta testing, operational test and evaluation, or shadow-mode operation. It can be followed by any mix of incremental or evolutionary developments, preplanned product improvements, and annual or other planning and development cycles.

Transitions. To move from LCA to IOC you can use any appropriate mix of waterfall, evolutionary, incremental, spiral, design to cost or schedule, or other models. Again, this lets your organization use the LCO, LCA, and IOC milestones as anchor points without overconstraining your intermediate processes. You can also use tailored versions of the three milestones to anchor major system upgrades or reengineering efforts.

Finally, these milestones let you define endpoints for cost and schedule estimates. Such estimates become rather meaningless if you can't reference them to well-defined endpoints. In fact, the primary validation of the LCO, LCA, and IOC milestones' relevance to industry and government came from an effort to define common milestones for Cocomo 2.0 cost model[10] estimates by a working group of the USC-UCI Cocomo 2.0 affiliates (see Acknowledgments).

THE WIN-WIN SPIRAL MODEL

The spiral model of software development begins each cycle of the spiral by performing the next level of elaboration of the prospective system's objec-

tives, constraints, and alternatives. As my review of implementations showed, a primary difficulty in applying the spiral model has been the lack of explicit process guidance in determining these objectives, constraints, and alternatives. Prasanta Bose and I recently developed the Win-Win spiral model[11] which uses the Theory W (win-win) approach[12] to converge on a system's next-level objectives, constraints, and alternatives. The Theory W approach involves identifying the system's stakeholders and their win conditions, and using negotiation processes to determine a mutually satisfactory set of objectives, constraints, and alternatives for the stakeholders.

Figure 1 illustrates the Win-Win spiral model. The original spiral model had four sectors, beginning with "establish next-level objectives, constraints, alternatives." The two additional sectors in each spiral cycle, "identify next-level stakeholders" and "identify stakeholders' win conditions," and the "reconcile win conditions" portion of the third sector, provide the collaborative foundation for the model. They also fill a missing portion of the original spiral model, namely, the means to answer the questions "Where do the next-level objectives and constraints come from?" and "How do you know they're the right ones?" The refined spiral model also explicitly addresses the need for concurrent analysis, risk resolution, definition, and elaboration of both the software product and the software process. In particular, the nine-step Theory W process translates into the following spiral model extensions:

- *Determine objectives.* Identify the system life-cycle stakeholders and their win conditions. Establish initial system boundaries and external interfaces.
- *Determine constraints.* Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some stakeholders.
- *Identify and evaluate alternatives.* Solicit suggestions from stakeholders. Evaluate them with respect to stakeholders' win conditions. Synthesize and negotiate candidate win-win alternatives. Analyze, assess, and resolve win-lose or lose-lose risks.
- *Record commitments,* and areas to be left flexible, in the project's design record and life-cycle plans.
- *Cycle through the spiral.* Elaborate win conditions, screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans.

TABLE 2: STAKEHOLDER CONCERNS AS ARCHITECTURE EVALUATION CRITERIA

Stakeholder	Concerns/Evaluation criteria
Customer	♦ Schedule and budget estimation ♦ Feasibility and risk assessment ♦ Requirements traceability ♦ Progress tracking ♦ Product-line compatibility
User	♦ Consistency with requirements and usage scenarios ♦ Future requirement growth accommodation ♦ Performance, reliability, interoperability, other quality attributes
Architect and system engineer	♦ Product-line compatibility ♦ Requirements traceability ♦ Support of tradeoff analyses ♦ Completeness, consistency of architecture
Developer	♦ Sufficient detail for design and development ♦ Framework for selecting/assembling components ♦ Resolution of development risks ♦ Product-line compatibility
Interoperator	♦ Definition of interoperability with interoperator's system
Maintainer	♦ Guidance on software modification ♦ Guidance on architecture evolution ♦ Definition of interoperability with existing systems

Stakeholder concerns/milestone criteria. The stakeholder win-win approach enables us to define a much more thorough set of evaluation criteria for the LCO, LCA, and IOC milestones. For example, Table 2 identifies a set of evaluation criteria for the LCA milestone in terms of the customer, user, architect, system engineer, developer, and maintainer stakeholders.[13]

As the table shows, the *customer* is likely to be concerned with getting first-order estimates of the software's cost, reliability, and maintainability based on its high-level structure. This implies that the architecture should be strongly coupled with the requirements, indicating if it can meet them. The customer will often have a longer-range concern: that the architecture will be compatible with corporate software product-line investments.

Users need software architectures to clarify and negotiate their requirements for the software being developed, especially with respect to future product extensions. At the architectural stage, the user will be interested in the impact of the software structure on performance, usability, and compliance with other system attribute requirements. As with the architecture of buildings, users need to relate the architecture to their usage scenarios.

Architects and systems engineers are concerned with translating requirements into architectural design. Therefore, their major concern is for consistency between the requirements and the architecture during the process of clarifying and negotiating the system requirements.

TABLE 3: RELATION OF WIN-WIN SPIRAL MODEL TO LCO AND LCA MILESTONES

	LCO		LCA
Cycle 1	**Cycle 2**	**Cycle 3**	
Determination of top-level concept of operations	Determination of detailed concept of operations	Elaboration of detailed concept of operations by increment, especially IOC	
System scope/boundaries/interfaces; top-level requirements	Top-level HW, SW, human requirements	Determination of requirements, growth vector by increment, especially IOC	
Small number of feasible candidate architectures (including major COTS, reuse choices)	Provisional choice of top-level information architecture	Choice of life-cycle architecture; some components of above TBD (low-risk and/or deferrable)	
Top-level life-cycle responsibilities (stakeholders), process model, cost/schedule parameters	Make detailed process strategy, responsibilities, cost/schedule allocation	Thorough WWWWWHH plans for IOC; essentials for later increments	
Stakeholder concurrence on top-level analysis supporting win-win satisfaction	More detailed analysis supporting win-win satisfaction	Stakeholder concurrence on thorough analysis supporting win-win satisfaction	
Top-level rationale, including rejected candidate architectures	More detailed rationale underlying system choices	Elaboration of rationale, including risk-resolution results	

Developers are concerned with getting an architectural specification that is detailed enough to satisfy the customer's requirements, but not so constraining as to preclude equivalent but different approaches or technologies in the implementation. They then use the architecture as a reference for developing and assembling system components, and to provide a compatibility check for reusing pre-existing components.

Interoperators use the software architecture as a basis for understanding (and negotiating about) the product to keep it interoperable with existing systems.

The *maintainer* is concerned with how easy it will be to diagnose, extend, or modify the software, given its high-level structure.

Spiral cycles and up-front milestones. Table 3 shows a set of three spiral cycles and their relationship to the LCO and LCA milestones: LCO occurs after cycle 1 and LCA occurs after cycle 3. However, other cycle configurations are acceptable as well. For example, for a large system, you could have an exploratory cycle before cycle 1 and could expand cycle 2 into two or more cycles (not necessarily sequential).

By the LCA milestone, the spiral cycles have converged on a compatible set of objectives, constraints, and alternatives for the system's life-cycle concept of operation, requirements, architecture, and plans. During this spiral process, these artifacts are defined and grow in detail as stakeholders identify and resolve risks and explore artifact interactions. Once such an LCA and its associated artifacts are in place, the project can use a waterfall, spiral, evolutionary, or other selected process to pursue the system's post-architecture development and evolution.

STARS PROJECT

The DoD's STARS project began in 1982 to address overall DoD software problems. By 1989, STARS was focused on developing a set of prototype software-engineering environments, or SEEs, for DoD use via contracts with three prime contractors — Boeing, IBM, and Unisys — and their subcontractor teams. However, there were major mismatches between the program's planned products and the needs of its prospective government and industry users, operators, and maintainers. These shortfalls were in areas such as tool support, tool integration, tailorability, robustness, compatibility with CASE tools, portability, and maintenance costs, which the DoD was expected to bear.

In late 1989, I assumed responsibility for the STARS program as office manager at the Defense Advanced Research Projects Agency. Along with the new STARS program manager, Jack Kramer, I prepared to apply the spiral model to address the program's risks. We found that incompatibilities among stakeholder expectations constituted a serious set of risks. We thus decided to enhance the spiral model with a Theory W approach to determine whether a win-win solution for STARS was feasible. If not, we would discontinue the program.

Commercializing STARS. The first two steps in the Win-Win spiral model are to identify the system's stakeholders and their associated win conditions. Table 4 summarizes the results of these steps for STARS. As often happens, the union of the stakeholders' win conditions produced an overconstrained situation. The STARS prime contractors were government contracting companies or divisions, and were not prepared to commercially sell and service the STARS SEEs. But without commercially supported SEEs, DoD could not afford to operate and maintain them. Thus, for the program to remain viable, the STARS prime contractors had to find commercial counterparts willing to sell and service the STARS SEEs. Eventually, each was able to do so: Boeing with DEC, IBM Federal Systems with IBM Canada, and Unisys Defense Systems with Hewlett-Packard. (IBM Federal Systems and Unisys Defense Systems became part of Loral, which is

now part of Lockheed Martin.)

However, although the commercial counterparts were very willing to develop SEEs that would support software development in the DoD-mandated Ada programming language, they were not willing to develop all their new SEE software in Ada, as then required by STARS. Their rationale was that their existing investments in C software and their need to support C for commercial SEE customers made programming in C much more cost-effective. Because such a cost-benefit rationale fit DoD's Ada waiver criteria, DARPA was able to create a win-win solution by waiving the Ada programming requirement for the STARS SEEs.

Winning compromises. A number of other overconstrained situations were also resolved into win-win situations for stakeholders. The revised STARS program also included[14]

- reorientation around much stronger software process and reuse support to achieve software quality and productivity win conditions;
- inclusion of a set of three demonstration projects, jointly sponsored by DARPA and a DoD Service (Army, Navy, Air Force), to reduce the risks of subsequent STARS SEE adoption by major Service programs.
- negotiation of a set of common open STARS SEE interface specifications, to enable CASE vendors to reach a larger marketplace and reduce tool rehosting costs; and
- addition of several STARS affiliates' programs to provide CASE vendors, DoD Service organizations, and other DoD software contractors with access to intermediate STARS products and a voice in the STARS evolution strategy.

STARS milestones. The equivalents of the common milestones in the STARS program required different things of the prime contractors.

- The LCO equivalent required the prime contractors to develop a set of "success plans" and get them endorsed by the other major stakeholders in a STARS/Users Workshop.[14]
- The LCA equivalent required the prime contractors to define risk-driven life-cycle architectures for the STARS environments. These included executing prototypes and rationales that reflected their responsiveness to the life-cycle objectives, such as the common open-interface specifications. (Responsiveness was not total; for example, commercial considerations outside DARPA's control caused Boeing-DEC to adopt the Atherton tool-integration framework rather than the SoftBench framework adopted by IBM and Unisys-HP.)
- The STARS IOC milestone required that each prime contractor deliver its STARS environment to a DoD Service project for use on a representative application of significant size: The IBM system was used on an Air Force space system, the Unisys-HP system was used on an Army signal-processing system, and the Boeing-DEC system was used on a Navy flight simulator system.

TABLE 4. STARS STAKEHOLDER WIN CONDITIONS

Stakeholder class	Win conditions
STARS prime contractors and their commercial counterparts	♦ Software Engineering Environment (SEE) sales ♦ DoD acceptance of commercial SEE product line ♦ Productivity leverage on primes' software business ♦ Satisfied customers and users
STARS subcontractors and CASE tool vendors	♦ Profits from large tools marketplace ♦ Reduced tool rehosting costs ♦ Open architecture, multiplatform, polylingual ♦ Stable evolution, voice in evolution strategy
Other DoD software contractors	♦ Productivity leverage on software business ♦ Open architecture, multiplatform, ease of extension ♦ Rapid availability, ease of use, reasonable cost ♦ Stable evolution, voice in evolution strategy
DoD software-support organizations	♦ Support of software maintenance functions ♦ Similar concerns to DoD software contractors ♦ Support of software reengineering, Ada transition
DoD services and agencies	♦ Accelerator for, and compatibility with, Service/Agency software initiatives ♦ Significant improvement in software productivity and quality ♦ Low risks of SEE adoption on critical projects
ARPA, Congress, taxpayers	♦ All of win conditions above ♦ SEE life-cycle affordability

Results. Under the management of John Foreman and Linda Brown, the successor DARPA STARS program managers, the STARS applications are generally reporting significant benefits from using the environment, process, and product line/reuse capabilities. For example, early results from the Air Force

Space Command's STARS application reported a cost improvement from $140 to $57 per delivered line of code and a quality improvement from more than 3 to 0.35 errors per thousand delivered lines of code. The Navy STARS program has reported a factor of 3 to 10 in quality improvement.[15]

Several other projects have successfully focused on the equivalents of these milestones. For example, the TRW-Air Force Command Center Processing and Display System-Replacement (CCPDS-R) project developed over 500,000 lines of complex distributed software within budget and schedule using an LCO-LCA-IOC approach with five increments. The initial increment, including the executing distributed kernel software, was part of the LCA milestone, which included demonstration of its ability to meet requirements growth projections.[16] The Microsoft software-development approach is converging toward an LCO-type milestone with its activity-based planning techniques.[9] It does not have a strong LCA milestone, but it does have a strong IOC milestone preceded by extensive beta testing, reflecting Microsoft's increasing appreciation of the risks involved in shipping software with high defect rates.

To avoid the problems of the previous model milestones —stakeholder mismatches, gold plating, inflexible point solutions, high-risk downstream capabilities, and uncontrolled developments—software projects need a mix of flexibility and discipline. The risk-driven content of the LCO, LCA, and IOC milestones let you tailor them to specific software situations and yet they remain general enough to apply to most software projects. And, because they emphasize stakeholder commitment to shared system objectives, they can provide your organization a collaborative framework for successfully realizing software's most powerful capability: its ability to help people and organizations cope with change. ◆

ACKNOWLEDGMENTS

This research is sponsored by DARPA through Rome Laboratory under contract F30602-94-C-0195 and by the Affiliates of the USC Center for Software Engineering: Aerospace Corp., Air Force Cost Analysis Agency, AT&T, Bellcore, DISA, Electronic Data Systems, E-Systems, Hughes Aircraft, Interactive Development Environments, Institute for Defense Analysis, Jet Propulsion Laboratory, Litton Data Systems, Lockheed Martin, Loral Federal Systems, Motorola, Northrop Grumman, Rational Software, Rockwell International, Science Applications International, Software Engineering Institute, Software Productivity Consortium, Sun Microsystems, Texas Instruments, TRW, US Air Force Rome Laboratory, US Army Research Laboratory, and Xerox.

I also thank Jack Kramer, John Foreman, Linda Brown, and the many STARS participants; Raghu Singh for his standards insights; and the *IEEE Software* reviewers for many improvements in this paper.

REFERENCES

1. W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," originally published in *Proc. Wescon*, Aug. 1970; currently available in *Proc. ICSE 9*, IEEE/ACM, New York, 1987.
2. D.D. McCracken and M.A. Jackson, "Life-Cycle Concept Considered Harmful," *ACM SW Eng. Notes*, Apr. 1982, pp. 29-32.
3. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
4. J. M. Carroll, *Scenario-Based Design*, John Wiley & Sons, New York, 1995.
5. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, Englewood Cliffs, N.J., 1996.
6. B.W. Boehm, *Software Risk Management*, IEEE CS Press, Los Alamitos, Calif., 1989.
7. R.N. Charette, *Software Engineering Risk Analysis and Management*, McGraw Hill, New York, 1989.
8. M.J. Carr et al., "Taxonomy-Based Risk Identification," CMU/SEI-93-TR-06, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, Penn., 1993.
9. M.A. Cusumano and R.W. Selby, *Microsoft Secrets*, The Free Press, New York, 1995.
10. B.W. Boehm et al., "Cost Models for Future Software Processes: Cocomo 2.0," *Annals of Software Engineering*, 1995.
11. B.W. Boehm and P. Bose, "A Collaborative Spiral Software Process Model Based on Theory W," *Proc. ICSP 3*, IEEE Press, New York, 1994.
12. B.W. Boehm and R. Ross, "Theory W Software Project Management: Principles and Examples," *IEEE Trans. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., July 1989.
13. C. Gacek et al., "Focused Workshop on Software Architectures: Issue Paper," *Proc. ICSE 17 Workshop on Software Architecture*, IEEE/ACM, New York, Apr. 1995.
14. J. Bamberger, ed., "STARS/Users Workshop: Final Report — Issues for Discussion Groups," CMU/SEI-90-TR-32, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, Penn., Dec. 1990.
15. R.R. Macala, L.D. Stuckey, Jr., and D.C. Gross, "Managing Domain-Specific, Product Line Development," *IEEE Software*, May 1996, pp. 57-67.
16. W.E. Royce, "TRW's Ada Process Model for Incremental Development of Large Software Systems," *Proc. ICSE 12*, IEEE/ACM, New York, Mar. 1990, pp. 2-11.

Barry Boehm is the TRW Professor of Software Engineering and Director of the Center for Software Engineering at the University of Southern California. His current research involves the WinWin groupware system for software requirements negotiation, architecture-based models of software quality attributes, and the Cocomo 2.0 cost-estimation model.

Boehm received a BA in mathematics from Harvard University and an MS and PhD in mathematics from the University of California, Los Angeles. He is a fellow of the IEEE and the AIAA, and a member of the National Academy of Engineering.

Address questions about this article to Boehm at the Center for Software Engineering, USC, Los Angeles, CA, 90089-0781; boehm@sunset.usc.edu. Additional information is available at http://sunset.usc.edu.