Anchoring the Software Process

Barry Boehm, USC

November 1995

(An abridged version appears in IEEE Software, July 1996)

Abstract

The current proliferation of software process models provides flexibility for organizations to deal with the unavoidably wide variety of software project situations, cultures, and environments. But it weakens their defenses against some common sources of project failure, and leaves them with no common anchor points around which to plan and control. This article identifies three milestones -- Life Cycle Objectives, Life Cycle Architecture, and Initial Operational Capability -- which can serve as these common anchor points. It also discusses why these particular three milestones or their equivalents are success-critical, particularly for large software projects, but for other software projects as well.

1. Introduction

For a few golden moments in the mid-1970's, it appeared that the software field had found a sequence of common anchor points: a set of milestones around which people could plan, organize, monitor, and control their projects. These were the milestones in the waterfall model, typically including the completion of system requirements, software requirements, preliminary design, detailed design, code, unit test, software acceptance test, and system acceptance test [Royce, 1970].

These milestones enabled companies, government organizations, and standards groups to establish a set of interlocking regulations, specifications, and standards. These covered a full set of software project needs, such as for cost and schedule estimation, project plans, reviews and audits, configuration management, and quality assurance. Extensive completion criteria were established for each milestone, such as completeness, consistency, traceability, testability, and feasibility.

What Happened to the Waterfall Model?

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. Some particularly dysfunctional mismatches involved projects developing user-interactive systems, projects involving extensive software reuse, or projects involving high-risk elements. For example, the ideal of a complete, consistent software requirements specification ran into the following problems:

- 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 software gold-plating. Users asked about their requirements would frequently 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 tended to produce point solutions optimized around the original problem statement. These solutions were frequently difficult to modify or to scale up to increased workload levels.

The ideal of a full-project Critical Design Review (CDR) of a complete, consistent detailed design also ran into problems. For large projects with 5,000 pages of detailed design specifications, the reviews were inevitably incomplete. They tended to impede progress, as developers were supposed to wait for several weeks for the review to complete before proceeding to code.

CDRs were effective for hardware developments, where the CDR was the final checkpoint for finding and fixing design defects before staffing up for production. For software, the project was staffed up well before CDR, making significant problems detected at CDR expensive to fix.

The ideal of a software acceptance test keyed to demonstrating compliance to a set of documented requirements specifications encountered problems as well. Foremost among these was that static tests of input-output transformations

provided little insight on whether the software would acceptably support common user task sequences or mission scenarios. The acceptance test also failed to cover many off-nominal problem situations subsequently encountered in beta-testing, field-testing, hardware-software integration, or in actual mission operations.

Evolutionary Development Milestones and Problems

The primary initial response to the waterfall model's problems was evolutionary development[McCracken-Jackson, 1982]. The only primary class of milestone in evolutionary development is the release of an increment of system capability. Its new content is determined from experience with the earlier releases of the system. Thus, the critical milestone in evolutionary development is 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 has the following problems:

- Inflexible point-solutions. Frequently, the initial release is optimized for initial demonstration-mode and exploratory-mode success. For example, it may store everything in main memory in order 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. Frequently also, the initial release will defer such considerations as security, fault-tolerance, and distributed processing in the interest of providing early functionality and user interface capabilities. The users may like the results, and expect the deferred considerations to be delivered equally rapidly. Usually, this puts the project in big trouble because the initial release's architecture cannot be easily extended to support the desired security, fault tolerance, distributed processing, or 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 analysisfrequently leads to an initial release which is so far from user needs that the users never bother learning and using it.

Process Proliferation

These difficulties with the waterfall and evolutionary development models have led to the development and use of a number of alternative process models: risk-driven (spiral), reuse-driven, legacy-driven, demonstration-driven, design to cost or schedule, incremental, and hybrid combinations of these with the waterfall or evolutionary development models.

This proliferation of software processes has made it very difficult for software organizations to establish a common frame of reference and commonly-defined milestones to serve as a basis for software life-cycle planning, measuring, controlling, and communicating with external organizations. In many cases, organizations have remained with admittedly flawed models such as the waterfall, because they have believed that the value of any common framework was worth the price of its imperfections.

Anchoring the Software Process

The remainder of this article will summarize the nature of three common milestones for anchoring the software process:

- Life Cycle Objectives (LCO)
- Life Cycle Architecture (LCA)
- Initial Operational Capability (IOC)

For each milestone, it provides a rationale for why the milestone is critical and common to virtually all software developments. It relates the milestones to recent process initiatives such as MIL-STD-498, EIA/IEEE J-STD-016, ISO standard 12207, the Win Win extension to the Spiral Model, and software product line management initiatives. It provides an example of the milestones' use: the ARPA-Services-Industry STARS program. It ends with a set of conclusions on the milestones and their ability to anchor the software process.

2. Milestones for the Future

Since the publication of the article on the Spiral Model [Boehm, 1988], the author has been able to review the results of a number of effective or flawed implementations of the model. A not-too-extreme example: "We decided that using the spiral model meant that we didn't have to write anything down, so now everybody's off doing different things, and we don't know how to pull them all together."

One of the most consistent correlates of success vs. failure on these projects was the degree to which they employed the equivalents of the three critical milestones below, particularly the first two front-end milestones. The key elements of these two milestones: stakeholder concurrence on the system's Life Cycle Objectives, and determination and validation of the system's Life Cycle Architecture, are summarized together in Table 1. Each is discussed further along with its rationale below.

A. Life Cycle Objectives (LCO)

The "Top-level system objectives and scope" part of the LCO milestone involves establishing the system boundary: the set of key decisions on what will and will not be included in the system to be developed. 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, workload levels, interoperating external systems, etc. These should be characterized not just at their initial operating levels, but in terms of their likely evolution, in order to avoid the point-solution difficulties discussed in the Introduction.

The "Operational Concept" involves working out scenarios [Carroll, 1995] of how the system will be used in operation. 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, scenarios for these should be developed as well. Scenarios for software and system maintenance need to be worked out, including determination of which organizations will be responsible for funding and performing the various functions. These organizations are some of the key

stakeholders whose concurrence is needed for realistic and supportable system definitions.

Table 1. Elements of Critical Front End Milestones

MilestoneElement	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 TBDs (to-be-determined items) • 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,	• Elaboration of WWWWWHH* for Initial Operational Capability (IOC) -Partial elaboration,

	maintainers, interoperators, general public, others • Identification of life-cycle process model -Top-level stages, increments • Top-level WWWWWHH* by stage	identification of key TBDs for later increments
Feasibility Rationale	• Assurance of consistency among elements above -Via analysis, measurement, prototyping, simulation, etc. -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

The "System Requirements" in the next part of the LCO definition in Table 1 are not absolute cast-in-concrete specifications as in the waterfall or related contract-oriented models. Instead, they record the collective stakeholders' concurrence on essential features of the system, whose detail can be modified easily and collaboratively as new opportunities (reuse opportunities, strategic partners), problems (budget cuts, technical difficulties), or developments (reorganizations, divestitures) arise.

The definition of "System and Software Architecture" should be at a sufficient level of detail to support analysis of the architecture's feasibility in supporting the system's objectives and requirements. Having more than one feasible choice of architecture is acceptable at the LCO stage; an example would be the existence of two feasible central commercial-off-the-shelf (COTS) products with different architectural implications. However, if no architectural option can be shown to be feasible, the project should be canceled; or its requirements, scope and objectives reworked. A record of infeasible options which were considered and dropped should be kept as insurance that these options will not be adopted in ignorance later.

A critical component of the initial "Life-Cycle Plan" is the identification of the major stakeholders in the system to be developed and evolved. These frequently

involve system user, customer, developer, and maintainer organizations. If the system is closely coupled with another system, the interoperator organization is a key stakeholder. If system safety, privacy, or other general-public issues are important, a representative of the general public should be a stakeholder. These are stakeholders whose concurrence on the system requirements is needed; otherwise, the system may not reflect their needs and will not be a success. Another critical component of the life cycle plan is the identification of the process model(s) to be used (waterfall, evolutionary, spiral, incremental, design-to-cost/schedule, or hybrid combination of these and others).

For the main part of the Life-Cycle Plan, an organizing principle is needed which scales down to provide simple plans for simple projects. A good approach is the WWWWWHH principle, which organizes the plan into 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?) ; and Resources (How much of each resource is necessary ?). Using this approach, the essential decision content of a life cycle plan for a small, straightforward project can be packed into one page or two briefing charts.

The most important thing to achieve for the Life Cycle Objectives milestone is the conceptual integrity and compatibility of its components above. The element which assures this is the "Feasibility rationale." It uses 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 would support the system's operational concept. A further key element of the rationale is the business case analysis, which establishes that the system would generate enough business value to be worth the investment. A counterpart in the defense sector is the Cost and Operational Effectiveness Analysis (COEA).

B. Life Cycle Architecture (LCA)

As indicated in Table 1, most of the elements of the LCA milestone are elaborations of the elements of the LCO milestone. The critical element of the LCA milestone is the definition of the system and software architecture itself. This

consists of definitions of the system and software components (either a hardware component, a computer program, a data ensemble, or a combination of such items), connectors (elements which mediate interactions among components), configurations (combinations of components and connectors), and constraints (e.g., resource limitations and shared assumptions about the operating environment). [Shaw-Garlan, 1996] provides an excellent treatment of software architectures.

Other key features of the LCA milestone are the specifics of COTS and reused software choices, which frequently drive both the architecture and the requirements; the specifics of quality attribute levels such as response time, reliability, and security, which are also significant architecture drivers; and the identification of likely directions of architectural evolution, to reduce the chances of the architecture itself becoming obsolete.

As with the LCO milestone, the most important things to achieve with the LCA milestone are:

- The feasibility rationale, which establishes the consistency and conceptual integrity of the other elements.
- The stakeholders' concurrence that the LCA elements are compatible with their objectives for the system.

A feature distinguishing the LCA milestone from the LCO milestone is the need to have all of the system's major risks resolved, or at least covered by an element of the system's risk management plan. For large systems, passing the LCA milestone is the point at which the project will significantly escalate its staff level and resource commitments. Proceeding into this stage with major risks unaddressed has led to disasters for many large projects. Some good guidelines for software risk assessment can be found in [Boehm, 1989; Charette, 1989; and Carr et al., 1993].

Distinguishing Features of the LCO and LCA Milestones

Here are the major features of the LCO and LCA milestones which distinguish them from most current software milestones, which provide a rationale for their success-critically on projects, and which enable them to function successfully 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 which anticipate and accommodate system evolution. This is the reason for calling them the "Life Cycle" Objectives and Architecture milestones.
- Elements can be either specifications or executing programs with data (e.g., prototypes, COTS products).
- Specifications are driven by risk considerations rather than completeness considerations. Critical interface specifications should be complete because it is risky otherwise. Written specifications of user interfaces should generally not try for completeness because their definition via prototypes is less risky.
- The LCO and LCA milestones are not peculiar to a single process model. One can go 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 essential. This establishes mutual stakeholder buy-in to the plans and specifications, and enables a collaborative team approach to unanticipated setbacks rather than an adversarial approach as in most contract models.

C. Initial Operational Capability (IOC)

Another distinguishing feature of the LCO and LCA milestones is that they are the milestones with the most serious consequences if one gets any parts of them wrong. At the other end of the development cycle, the milestone with the most serious consequences of getting things wrong is the Initial Operational Capability

(IOC). Greeting users with a new system having ill-matched software, poor site preparation, or poor user preparation has been a frequent source of user alienation and killed projects.

The key elements of the IOC milestone are:

- Software preparation, including both 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.
- User, operator and maintainer preparation, including selection, teambuilding, training and other qualification for familiarization 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. Thus, for example, these objectives drive the tradeoff between IOC date and quality of the product (e.g. between the safety-critical Space Shuttle Software and a market window- critical commercial software product.) The difference between these two cases is narrowing as commercial vendors and users increasingly appreciate the market risks involved in buggy products [Cusumano-Selby, 1995].

As with the LCO and LCA milestones, the IOC milestone is compatible with multiple classes of preceding and following 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, pre-planned product improvements, annual or other planning and development cycles. And the process of getting from LCA to IOC can be any appropriate mix of waterfall, evolutionary, incremental, spiral, design to cost/schedule, or other models. Again, this enables organizations to use the LCO, LCA, and IOC milestones as anchor points without overconstraining their intermediate processes. Tailored versions of the three

milestones can also be used to anchor major system upgrades or reengineering efforts.

Another benefit of these common milestones is the ability to define endpoints for cost and schedule estimates. Such estimates become rather meaningless if you can't reference them to well-defined endpoints. Much of the definition of the LCO, LCA, and IOC milestones has been done via the USC-UCI Affiliates' program for the definition and development of the COCOMO 2.0 cost model [Boehm et al., 1995].

3. Relation to Recent Initiatives

Recent software process initiatives have provided guidelines which make it easier to depart from lock-step software processes such as the waterfall model, These initiatives are also compatible with the three common anchor points above. This section discusses recent software process standards such as MIL-STD-498 [DoD, 1994], currently evolving into EIA/IEEE J-STD-016 [EIA/IEEE, 1995]; and ISO/IEC Standard 12207 [ISO, 1995]. It also describes recent elaborations of the spiral model, such as the Win Win Spiral Model [Boehm-Bose, 1994] and their relation to the three anchor points. It concludes with a discussion of the applicability of the anchor points not just to individual projects, but to the management of software product lines with domain architectures and reusable components.

3.1 MIL-STD-498: Software Development and Documentation

This standard supersedes the U.S. Department of Defense's DoD-STD-2167A and DoD-STD-7935A, which largely focused DoD projects on waterfall-model software processes. MIL-STD-498 goes away from this approach by focusing on required software activities rather than phases. It states that the 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" (Section 4.1). It provides examples of its application to waterfall, incremental, and evolutionary processes (Appendix G), and a guidebook providing more detailed usage examples and tailoring guidelines.

Its guidance on System Requirements Analysis (Section 5.3) and System Architectural Design (Section 5.4.2) are quite consistent with the LCO milestone

above. 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 (Section 5.5) and Software Design (Section 5.6) are compatible with the LCA milestone above, but the standard misses several opportunities to emphasize the coupling of software architecture to anticipated directions of requirements evolution and the establishment of the Feasibility Rationale as a first-class citizen. (It requires the recording of global design decisions, but relegates their rationale to a portion of the Notes sections in the system and software design Data Item Descriptions).

MIL-STD-498 also advances from previous DoD standards to cover activities involved in proceeding from software configuration item acceptance tests to the equivalent of the IOC milestone (Sections 5.10 through 5.17). Its guidelines for application to incremental and evolutionary development in Appendix G also show how these apply to the IOC milestone. EIA/IEEE J-STD-016 extends MIL-STD-498 to cover commercial as well as DoD software processes. Terms such as "CSCI," "HWCI,", and "DID" are now "software items," "hardware items," and "software product descriptions." Clauses related to contracting and deliverable-vs.-non-deliverable software are eliminated, and some requirements on reuse, tailoring, traceability, and programming language are reworked.

3.2 ISO/IEC 12207: Information Technology Life Cycle Processes

The ISO/IEC 12207 standard [ISO, 1995] is similar to MIL-STD-498 in that it focuses on activities and core processes (acquisition, supply, development, operation, maintenance, supporting life cycle processes, organizational life cycle processes) which can be performed sequentially, repeated, and combined according to the project's choice of life cycle model(s). Example models cited in this regard are waterfall, evolutionary builds, pre-planned product improvement, and spiral (Annex B.4).

ISO/IEC 12207's "System requirements analysis" (Section 5.3.2) and "System architectural design" (Section 5.3.3) provisions are consistent with the LCO milestone. It goes beyond MIL-STD-498 in emphasizing the need to co-define the system requirements and architecture, and to document the results of feasibility evaluations. Significantly, it includes "feasibility of system architectural design" as

a requirements evaluation criterion, and “traceability to...” and “consistency with the system requirements” as architectural design evaluation criteria. The treatment of the counterpart “Software requirements analysis” and “Software architectural design” activities is similar (Sections 5.3.4 and 5.3.5) and consistent with the LCA milestone.

ISO/IEC 12207 is also consistent with the IOC milestone in its accommodation of builds or increments, and in that its culminating development process activities are “Software installation” and “Software acceptance support” (Sections 5.3.12 and 5.3.13). Overall, ISO/IEC 12207 goes farther than MIL-STD-498 in countering the problem areas cited in Section 1 of this paper. However, it also misses some opportunities to make the architectural rationale an integral part of 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.

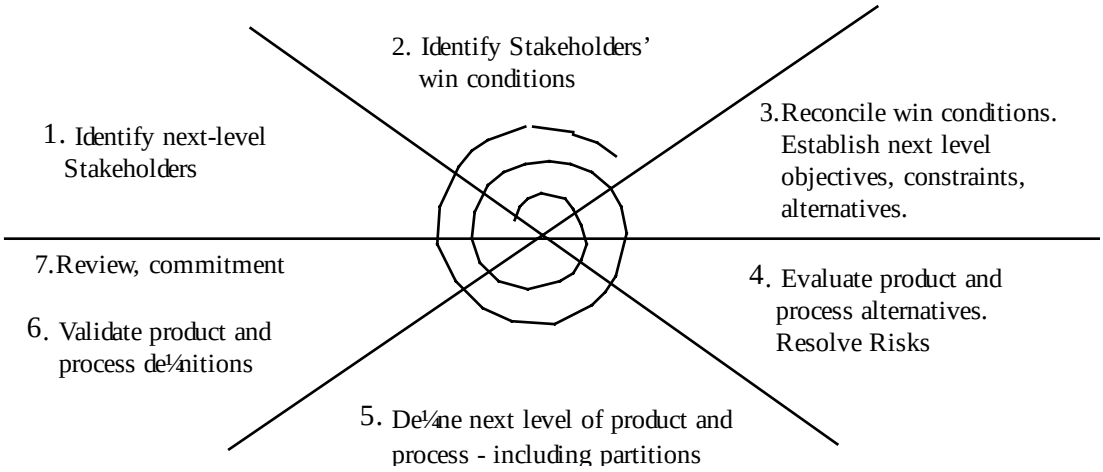
3.3 The Win Win Spiral Model

The Spiral Model of software development [Boehm, 1988] begins each cycle of the spiral by performing the next level of elaboration of the prospective system's objectives, constraints, and alternatives. A primary difficulty in applying the spiral model has been the lack of explicit process guidance in determining these objectives, constraints, and alternatives. The recently - developed Win-Win Spiral Model [Boehm - Bose, 1994] uses the Theory W (win-win) approach [Boehm-Ross, 1989] to converge on a system's next-level objectives, constraints, and alternatives. This 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: the means to answer, “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:

FIGURE 1. The Win-Win Spiral Model



- 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.

Relation of Stakeholder Concerns to 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 Life Cycle Architecture milestone in terms of the customer, user, architect, system engineer, developer, and maintainer stakeholders [Gacek et al, 1995].

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 interfaces with interoperator's system
Maintainer	• Guidance on software modification • Guidance on architecture evolution • Definition of interoperability with existing systems

Table 2: Stakeholder Concerns as Architecture Evaluation Criteria

For example, the *customer* is likely to be concerned with getting first-order estimates of the cost, reliability, and maintainability of the software 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 also have longer-range concerns that the architecture be compatible with corporate software product line investments. *Users* need software architectures in order to be able to clarify and negotiate their requirements for the software being developed, especially with respect to future extensions to the product. The user will be interested at the architecting stage in the impact of the software structure on performance, usability, and compliance with other system attribute requirements. As with architectures of buildings, users also need to relate the architecture to their usage scenarios.

Architects and Systems Engineers are concerned with translating requirements into high-level design. Therefore, their major concern is for consistency between the requirements and the architecture during the process of clarifying and negotiating the requirements of the system. *Developers* are concerned with getting an architectural specification that is sufficient in detail to satisfy the

customer's requirements but not so constraining as to preclude equivalent but different approaches or technologies in the implementation. Developers then use the architecture as a reference for developing and assembling system components, and also use it 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 in order to keep it interoperable with existing systems. The *maintainer* will be concerned with how easy it will be to diagnose, extend or modify the software, given its high-level structure.

Relation of Spiral Cycles to LCO and LCA Milestones

Table 3 shows a representative set of spiral cycles with their relationship to the LCO and LCA milestones. In Table 3, three spiral cycles are shown, with LCO occurring after cycle 1 and LCA occurring after cycle 3. However, other cycle configurations are acceptable as well. For example, for a large system, one could have an earlier 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 selected and grown in detail as risks are identified and resolved, and interactions among the artifacts are explored. Once such a Life Cycle Architecture 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.

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

Table 3: Relation of Win-Win Spiral Model to LCO and LCA Milestones

3.4 Software Product Line Management

If an organization applies the LCO, LCA, and IOC milestones separately to each individual software project, it will get a sub-optimal outcome. It will get a series of separate "stovepipe" systems with many redundantly-developed and incompatible components. In order to achieve the cost, schedule, and quality benefits of software reuse, it is important to develop a software product line management approach.

This involves extending particularly the LCO and LCA milestone definitions. For the LCO milestone, it is important to determine the breadth of the product line

domain across which reusable components will be shared (e.g., transaction processing, message processing, military message processing, military medical message processing). For the LCA milestone, it involves developing a domain architecture for the product line, rather than just a life cycle architecture for an individual system.

A recent major initiative, the STARS program, has developed a set of software environment life cycle process, and software asset library capabilities supporting software reuse and product line management. As STARS itself has been a large, complex software system which successfully used the win-win spiral model and the life cycle milestones described above, we will discuss it in the next section as an application example.

4.0 Application Example: The STARS Program

The U.S. DoD (Department of Defense) STARS (Software Technology for Adaptable, Reliable Systems) Program began in 1982 as an integrated program to address the overall ensemble of DoD software problems. By 1989, it was focused on developing a set of prototype software engineering environments (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 such areas as tool support, tool integration, tailorability, robustness, compatibility with CASE tools, portability, and maintenance costs, which were expected to be borne by DoD.

The author assumed responsibility for the STARS program on his arrival as a Defense Advanced Research Projects Agency (DARPA) office manager in late 1989. He and the program's prospective new program manager, Dr. Jack Kramer, prepared to apply the spiral model to address the program's risks. They found that a serious set of risks involved incompatibilities among the expectations of the program's stakeholders. They decided to enhance the spiral model with a Theory W approach to determine whether a win-win solution for STARS was feasible (and if not, to discontinue the program).

As indicated in Section 3, 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 in Table 4 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, it became necessary for the STARS prime contractors 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 are now parts of Loral, Inc.).

However, although the commercial counterparts were very willing to develop SEEs which 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 it a much more cost-effective solution to program in C. Since 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 SEE's.

Similarly, a number of other overconstrained situations were resolved into win-win situations for the stakeholders in Table 4. Some additional resulting features of the revised STARS program were [Bamberger, 1990]:

- 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 re-hosting costs.
- 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.

The STARS equivalent of the LCO milestone involved a set of "Success Plans" developed by the STARS prime contractors and endorsed by the other major stakeholders in a STARS/Users Workshop [Bamberger, 1990]. The LCA milestone involved risk-driven life cycle architecture definitions of the STARS environments by the STARS primes. These included executing prototypes, and rationales reflecting 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 involved each prime contractor delivering its STARS environment to a DoD Service project for use on a significant-sized representative application. An Air Force space system used the IBM system; an Army signal processing system used the Unisys - HP system; and a Navy flight simulator system used the Boeing - DEC system.

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 over 3 to 0.35 errors per 1000 delivered lines of code.

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 sub-contractors and CASE tool vendors	• Profits from large tools marketplace • Reduced tool re-hosting costs • Open architecture, multi-platform, polylingual • Stable evolution, voice in evolution strategy
Other DoD software contractors	• Productivity leverage on software business • Open architecture, multi-platform, 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, 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

Although space limitations preclude detailed discussion of other projects which have successfully focused their developments on the equivalents of the LCO, LCA, and IOC milestones, two other examples are worth briefly citing. 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 [Royce, 1990]. The Microsoft software development approach described in [Cusumano-Selby, 1995] is converging toward an LCO-type milestone with its activity-based planning techniques. It does not have a strong LCA milestone, but it has 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.

5. Conclusions

In order to avoid the problems identified in the introduction to this article (stakeholder mismatches, gold plating, inflexible point solutions, high-risk downstream capabilities, uncontrolled developments), software projects need a risk-driven mix of flexibility and discipline. The three key life cycle milestones described here (Life Cycle Objectives, Life Cycle Architecture, and Initial Operational Capability) enable flexible mixes of sequential, cyclic, incremental, and evolutionary process models, while providing anchor points around which to perform disciplined planning and control.

The risk-driven content of the three key milestones enables them to be specifically tailorable to a given software situation, yet general enough to apply to most software project situations. Their use as life-cycle anchor points has proven successful on large, complex software projects. And their emphasis on stakeholder commitment to shared system objectives provides an approach for realizing software's most powerful capability: its ability to help people and organizations cope with change.

6. Acknowledgments

This research is sponsored by the Advanced Research Projects Agency (ARPA) through Rome Laboratory under contract F30602-94-C-0195 and by the

Affiliates of the USC Center for Software Engineering. The current Affiliates are Aerospace Corporation, Air Force Cost Analysis Agency, AT&T, Bellcore, DISA, Electronic Data Systems Corporation, E-Systems, Hughes Aircraft Company, Interactive Development Environments, Institute for Defense Analysis, Jet Propulsion Laboratory, Litton Data Systems, Lockheed Martin Corporation, Loral Federal Systems, Motorola Inc., Northrop Grumman Corporation, Rational Software Corporation, Rockwell International, Science Applications International Corporation, Software Engineering Institute (CMU), Software Productivity Consortium, Sun Microsystems, Inc., Texas Instruments, TRW, U.S. Air Force Rome Laboratory, US. Army Research Laboratory, and Xerox Corporation. I would also like to 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.

7.References

[Bamberger, 1990]. J. Bamberger, ed., "STARS/Users Workshop: Final Report - Issues for Discussion Groups," CMU/SEI-90-TR-32, Software Engineering Institute, Pittsburgh, PA, December 1990.

[Boehm, 1988]. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61-72.

[Boehm, 1989]. B.W. Boehm. "Software Risk Management", IEEE Computer Society Press, 1989.

[Boehm - Bose, 1994], B.W. Boehm and P. Bose, "A Collaborative Spiral Software Process Model Based on Theory W," Proceedings, ICSP 3, IEEE, Reston, Va. October 1994.

[Boehm et al., 1995]. B.W. Boehm, B. K. Clark, E. Horowitz, R. Madachy, R.W. Selby, and C. Westland, "Cost Models for Future Software Processes: COCOMO 2.0," Annals of Software Engineering, 1995.

[Boehm - Ross, 1989]. B.W. Boehm and R. Ross, "Theory W Software Project Management: Principles and Examples," IEEE Trans. Software Engr., July 1989.

[Carr et al., 1993]. M.J. Carr, S.L. Konda, I. Monarch, F.C. Ulrich, and C.F. Walker, " Taxonomy-Based Risk Identification," CMU/SEI-93-TR-06, Software Engineering Institute, 1993.

[Carroll, 1995]. J. M. Carroll, Scenario-Based Design, Wiley, 1995.

[Charette, 1989]. R.N. Charette, Software Engineering Risk Analysis and Management, McGraw Hill, 1989.

[Cusumano-Selby, 1995]. M.A. Cusumano and R.W. Selby, Microsoft Secrets, Free Press, 1995.

[DoD, 1994]. U.S. Department of Defense, "Military Standard 498: Software Development and Documentation," 5 December 1994. Defense Printing Service, Philadelphia, PA 19111-5094.

[EIA/IEEE, 1995]. EIA/IEEE, "Trial Use Standard J-STD-016, Software Life Cycle Processes," December 1995. Available via Global Engineering, 1-800-854-7179.

[Gacek et al., 1995], C. Gacek, A. Abd-Allah, B.K. Clark, and B.W. Boehm, "Focused Workshop on Software Architectures: Issue Paper," Proceedings of the ICSE 17 Workshop on Software Architecture, April 1995.

[ISO, 1995]. International Standards Organization, "International Standard ISO/IEC 12207: Information Technology -- Software Life-Cycle Processes," 1 August 1995. Available via Global Engineering, 1-800-854-7179.

[McCracken-Jackson, 1982]. D.D. McCracken and M.A. Jackson, "Life-Cycle Concept Considered Harmful," ACM SW Engr. Notes, April 1982, pp. 29-32.

[Royce, 1990]. W.E. Royce, "TRW's Ada Process Model for Incremental Development of Large Software Systems," Proceedings, ICSE 12, IEEE/ACM, March 1990, pp. 2-11.

[Royce, 1970]. W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," Proc. Wescon, August 1970. Also available in Proceedings, ICSE 9, IEEE/ACM, 1987.

[Shaw-Garlan, 1996]. M. Shaw and D. Garlan, Software Architecture; Perspectives on an Emerging Discipline, Prentice-Hall, 1996.