A Risk-Driven Process Decision Table to Guide System Development Rigor

Barry Boehm, Jo Ann Lane, and Supannika Koolmanojwong
University of Southern California Center for Systems and Software Engineering
{boehm, jolane, koolmano} at usc.edu

Abstract. The Incremental Commitment Model (ICM) organizes systems engineering and acquisition processes in ways that better accommodate the different strengths and difficulties of hardware, software, and human factors engineering approaches. As with other models trying to address a wide variety of situations, its general form is rather complex. However, its risk-driven nature has enabled us to determine a set of twelve common risk patterns and organize them into a decision table that can help new projects converge on a process that fits well with their particular process drivers. For each of the twelve special cases, the decision table provides top-level guidelines for tailoring the key activities of the ICM, along with suggested lengths between each internal system build and each external system increment delivery. This paper elaborates on each of the twelve cases and provides examples of their use.

Introduction

The Incremental Commitment Model (ICM), developed in a recent National Research Council study on integrating human factors into the systems development process, organizes systems engineering and acquisition processes in ways that better accommodate the different strengths and difficulties of hardware, software, and human factors engineering approaches. It also provides points at which they can synchronize and stabilize. At these points, the risks of going forward can be better assessed and fitted into a risk-driven stakeholder resource commitment process.

As with other models trying to address a wide variety of situations, its general form is rather complex. However, its risk-driven nature has enabled us to determine a set of common risk patterns and organize them into a decision table that can help new projects converge on a process that fits well with their particular process drivers.

The process drivers used as inputs to the decision table include the system's size and complexity; its rate of change; its mission-criticality; the extent of non-developmental item (NDI) support for its desired capabilities; and the available organizational and personnel capability for developing the system.

The decision table includes twelve common risk-driven special cases of the ICM. With representative examples in parentheses, these are (1) Use NDI (Small accounting system); (2) Agile (Small e-services); (3) Architected agile (Business data processing); (4) Formal methods (Large-Scale Integration (LSI) chip; security kernel); (5) Software-embedded hardware component (Multi-sensor control device); (6) Indivisible initial operational capability (Complete vehicle platform); (7) NDI-intensive system (Supply chain management); (8) Hybrid agile/plan-driven system (Command, Control, Computing, Communications, Intelligence, Surveillance, Reconnaissance (C4ISR) system); (9) Multi-owner system of systems (SoS) (Net-centric emergency services); (10) Family of systems (Medical device product line), (11) Brownfield (Incremental legacy phaseout); (12a) Net-centric services—community support (Community services); and (12b) Net-centric services—quick response decision support (Response to competitor intiative). The following sections of this paper elaborate on each of the twelve cases and provide examples of their use.

ICM Background

The ICM, illustrated in Figure 1, is a risk-driven framework for tailoring system life-cycle processes. The ICM uses risk to determine how much process agility or rigor is enough to satisfy the system's objectives subject to its constraints. This may vary across different parts of the system, depending on the risks associated with the various parts. And for SoSs, where the component systems are owned and maintained by different organizations, this variation is often unavoidable.

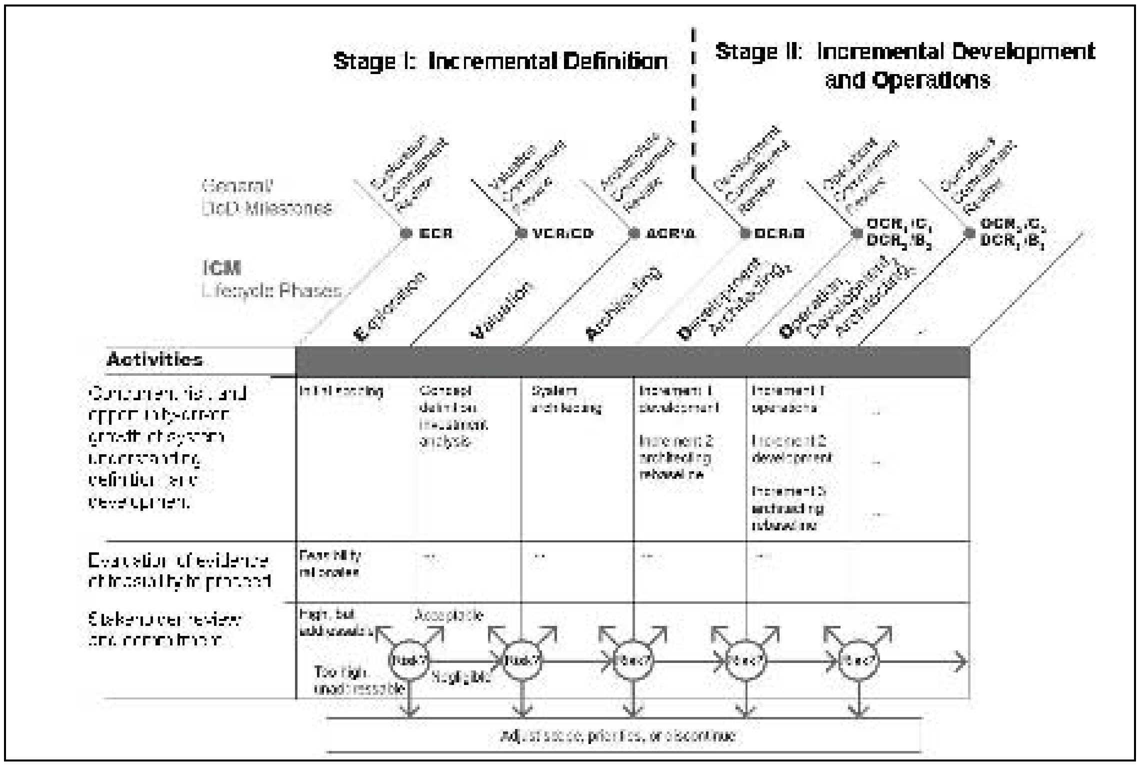
Stage I: Incremental Definition
Stage II: Incremental Development and Operations
General/ DoD Milestones
ECR
VCR/CD
ACR/A
DCR/B
OCR1/C1 DCR2/B2
OCR2/C2 DCR3/B3
ICM Lifecycle Phases
Exploration
Valuation
Architecting
Development Architecting
Operations Development Architecting
Activities
Concurrent risk- and opportunity-driven growth of system understanding, definition, and development
Initial scoping
Concept definition, investment analysis
System architecting
Increment 1 development
Increment 2 architecting rebaseline
Increment 1 operations
Increment 2 development
Increment 3 architecting rebaseline
Evaluation of evidence of feasibility to proceed
Feasibility rationales
Stakeholder review and commitment
High, but addressable
Acceptable
Risk?
Too high, unaddressable
Negligible
Adjust scope, priorities, or discontinue

Figure 1. Overview of the ICM.

Figure 1 identifies the concurrently engineered life cycle phases, the stakeholder commitment review points and their use of feasibility rationales to assess the compatibility, feasibility and risk associated with the concurrently-engineering artifacts; and the major focus of each life cycle phase. There are a number of alternatives at each commitment point. These are: (1) the risks are negligible and no further analysis and evaluation activities are needed to complete the next phase; (2) the risk is acceptable and work can proceed to the next life cycle phase: (3) the risk is addressable but requires backtracking; or (4) the risk is too great and the development process should be rescoped or halted. These risks are assessed by the system's success-critical stakeholders, whose commitment will be based on whether the current level of system definition gives sufficient evidence that the system will satisfy their value propositions. Thus there are many risk-driven paths through the life cycle. A more risk-seeking set of stakeholders will tend to go forward or skip phases at a decision point; for the same level of risk, a more risk-averse set of stakeholders may choose to extend the previous phase, rescope, or discontinue the project.

The ICM pulls together and integrates a) agile processes for assessing the system environment and user needs and then planning for the implementation of new and modified system capabilities, b) plan-driven (often time-boxed) processes to develop and field new capabilities, and c) continuous verification and validation (V&V) to provide high assurance of the requisite system qualities. The ICM also strives to integrate key engineering disciplines (e.g., systems, software, human factors) to develop desired systems and system capabilities in a cost/schedule-effective manner and to support the evolution of these systems over time to meet changing user needs. The ICM is based upon the premises that many systems of today contain a significant amount of software, the requirements for these systems cannot be specified up front, and the requirements associated with these systems can and do change over time. A key to success is building in system adaptability and flexibility—and software is often the enabler for this needed adaptability and flexibility.

Essential to the ICM are its six core principles: 1) commitment and accountability of system sponsors, 2) success-critical stakeholder satisficing, 3) incremental growth of system definition and stakeholder commitment, 4) concurrent engineering, 5) iterative development cycles, and 6) risk-based activity levels and milestones.

The overall lifecycle process divides naturally into two major stages. Stage I, Incremental Definition, covers the up-front growth in system understanding, definition, feasibility assurance, and stakeholder commitment leading to a larger Stage II commitment to a feasible set of specifications and plans for Incremental Development and Operations.

To focus on all of the key aspects of a given software-intensive system requires that a great deal of concurrent activity occurs within and across the various ICM phases (Pew and Mavor, 2007). Figure 2 illustrates some of this concurrency and the associated levels of effort.

In order to rapidly and successfully adapt to increasing rates of change, projects need to be able to concurrently rather than sequentially assess and manage opportunities and risks; requirements, solutions, plans, and business cases; and hardware, software and human factors. Figure 2 builds on the Rational Unified Process "hump diagram" in [Kruchten 1999] to show how these are concurrently pursued with the ICM [Pew and Mavor 2007; Boehm and Lane 2007].

As with the RUP version, it should be emphasized that the magnitude and shape of the levels of effort will be risk-driven and likely to vary from project to project. In particular, they are likely to have mini risk/opportunity-driven peaks and valleys, rather than the smooth curves shown for simplicity in Figure 2. The main intent of this view is to emphasize the necessary concurrency of the primary success-critical activities shown as rows in Figure 2. Thus, in interpreting the Exploration column, although system scoping is the primary objective of the Exploration phase, doing it well involves a considerable amount of activity in understanding needs, envisioning opportunities, identifying and reconciling stakeholder goals and objectives, architecting solutions, life cycle planning, evaluation of alternatives, and negotiation of stakeholder commitments.

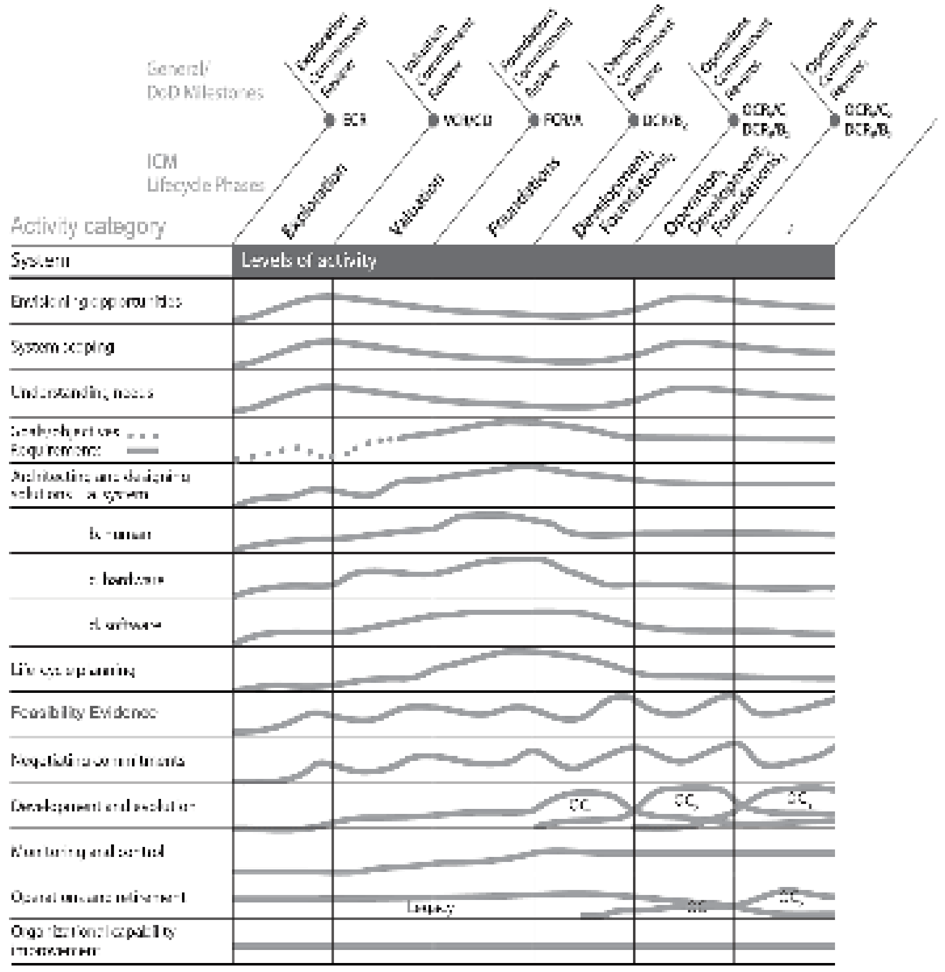

Figure 2. ICM Concurrent Activities and Level of Effort.

For example, if one were exploring the initial scoping of an SoS for a metropolitan area's disaster relief, one would not just interview a number of stakeholders and compile a list of their expressed mission needs: One would also envision and explore opportunities for reusing (parts of) other metropolitan-area disaster relief systems; for obtaining development funds from federal agencies; and for applying maturing virtual collaboration technologies. In the area of understanding needs, one would concurrently assess the capability and compatibility of existing disaster relief systems in the metropolitan area to determine which would need the most work to re-engineer into a SoS. One would also assess the scope of authority and responsibility of each existing system to determine whether the best approach would be a truly integrated and centrally-managed SoS or a best-effort interoperable set of systems. And one would explore alternative architectural concepts for developing and evolving the system; develop alternative phased plans to determine which improvements would provide the best early benefits and foundations for future growth; evaluate their relative feasibility, benefits, and risks for stakeholders to review; and negotiate commitments of further resources to proceed into a Valuation phase. Similar concurrency is needed all the way down to small time-constrained web applications.

To make this concurrency work, the anchor point milestone reviews are the mechanism by which the many concurrent activities are synchronized, stabilized, and risk-assessed at the end of each phase. Each of these anchor point milestone reviews, labeled at the top of figures 1 and 2, is focused on developer-produced *evidence*, instead of PowerPoint charts and Unified Modeling Language (UML) diagrams, to help the key stakeholders determine the next level of commitment. For the Exploration Commitment Review (ECR), the focus is on a review of an Exploration Phase plan with the proposed scope, schedule, deliverables, and required resource commitment, by a key subset of stakeholders. The plan content is risk-driven, and could therefore be put on a single page for a small and non-controversial Exploration phase since there is minimal risk at this point—a much riskier Exploration phase would require a more detailed plan outlining how the risks will be re-evaluated and managed going forward. For the Valuation Commitment Review (VCR), the risk-driven focus is similar; the content includes the Exploration phase results and a valuation phase plan; and a review by all of the stakeholders involved in the Valuation phase. The Foundations Commitment Review (FCR) and the Development Commitment Review (DCR) reviews are based on the highly successful AT&T Architecture Review Board procedures described in (Marenzano et al., 2005). For the FCR, only high-risk aspects of the Operational Concept, Requirements, Architecture, and Plans are elaborated in detail. At this point, typically multiple options have been investigated and it is sufficient to provide evidence that at least one combination of the possible options is feasible. At the DCR, feasibility is demonstrated for a particular the option set selected for development.

ICM Process Decision Table

Because of its complexity, ICM examples have been developed to show users how to use the framework to create a development process appropriate for their system of interest. These cases cover the very small to the very large as well as the use of commercial off-the-shelf (COTS) software products to the development of a large, complex custom software application or integrated sets of software applications. Each of these situations presents risks at the various stages of development, some risks more critical than others. The goal of the ICM is to identify these risks and then tailor the process to include rigor where necessary to investigate and manage the risks and to streamline the process when risks are negligible, allowing the development team to be more agile when possible. Table 1 contains a list of the special cases of the ICM and an example of each case. These cases cover a broad spectrum of software-intensive systems. In addition, Table 1 describes the characteristics of the case (or application category) with respect to size and complexity; expected change rate during the development process; the overall criticality of the application, typically as it applies to human life, security/protection of information, or financial liability; and the type of NDI support one might expect for an application in this category. It also indicates the organizational and personally capabilities that are required to be successful at building this type of application and the typical/suggested lengths for each internal system build and each external system increment delivery. Table 2 provides top-level guidelines for tailoring the key activities in Stage I (incremental definition) and Stage II (incremental development and operations) of the ICM.

Table 1: Characteristics of the Risk-Driven Special Cases of the ICM.

Special Case	Example	Size, Complexity	Change Rate (%/Month)	Criticality	NDI Support	Organizational and Personnel Capability	Time/Build; Time/Increment
1. Use NDI	Small accounting				Complete		
2. Agile	E-services	Low	1-30	Low-Med	Good; in place	Agile-ready Med-high	<= 1 day; 2-6 weeks
3. Architected Agile	Business data processing	Med	1-10	Med-High	Good; most in place	Agile-ready Med-high	2-4 weeks; 2-6 months
4. Formal Methods	Security kernel; Safety-critical LSI chip	Low	0.3	Extra High	None	Strong formal methods experience	1-5 days; 1-4 weeks
5. HW with embedded SW component	Multi-sensor control device	Low	0.3-1	Med-Very High	Good; in place	Experienced; med-high	SW: 1-5 days; Market-driven
6. Indivisible IOC	Complete vehicle platform	Med-High	0.3-1	High-Very High	Some in place	Experienced; med-high	SW: 2-6 weeks; Platform: 6-18 months
7. NDI- intensive	Supply chain management	Med-High	0.3-3	Med-Very High	NDI-driven architecture	NDI-experienced; med-high	SW: 1-4 weeks; Systems: 6-18 months
8. Hybrid agile/ plan-driven system	C4ISR system	Med-Very High	Mixed parts; 1-10	Mixed parts; Med-Very High	Mixed parts	Mixed parts	1-2 months; 9-18 months
9. Multi-owner system of systems	Net-centric military operations	Very High	Mixed parts; 1-10	Very High	Many NDIs; some in place	Related experience, med-high	2-4 months; 18-24 months
10. Family of systems	Medical device product line	Med-Very High	1-3	Med-Very High	Some in place	Related experience, med-high	1-2 months; 9-18 months
11. Brownfield	Incremental legacy phaseout	High-Very High	0.3-3	Med-High	NDI as legacy replacement	Legacy re-engineering	2-6 weeks/ refactor ; 2-6 months
12a. Net- Centric Services—Community Support	Community Services or Special Interest Group	Low-Med	0.3-3	Low-Med	Tailorable service elements	NDI-experienced	<= 1 day; 6-12 months
12b. Net-Centric Services—Quick Response Decision Suppport	Response to competitor initiative	Med-High	3-30	Med-High	Tailorable service elements	NDI-experienced	<= 1 day ; QR-driven

Legend:
C4ISR: Command, Control, Computing, Communications, Intelligence, Surveillance, Reconnaissance; **HW:** Hardware; **IOC:** Initial Operational Capability; **NDI:** Non-Development Item; **SW:** Software.

Table 2: Key Activities for Each Special Case of the ICM.

Special Case	Example	Key Stage I Activities: Incremental Definition	Key Stage II Activities: Incremental Development, Operations
1. Use NDI	Small accounting	Acquire NDI	Use NDI
2. Agile	E-services	Skip Valuation, Architecting phases	Scrum plus agile methods of choice
3. Architected Agile	Business data processing	Combination Valuation, Architecting phases. Complete NDI preparation	Architecture-based Scrum of Scrums
4. Formal Methods	Security kernel; Safety-critical LSI chip	Precise formal specification	Formally-based programming language; formal verification
5. HW with embedded SW component	Multi-sensor control device	Concurrent HW/SW engineering. CDR-level ICM DCR	IOC Development, LRIP, FRP. Concurrent version N+1 engineering
6. Indivisible IOC	Complete vehicle platform	Determine minimum-IOC likely, conservative cost. Add deferrable SW features as risk reserve	Drop deferrable features to meet conservative cost. Strong award fee for features not dropped
7. NDI- intensive	Supply chain management	Thorough NDI-suite life cycle cost-benefit analysis, selection, concurrent requirements/ architecture definition	Pro-active NDI evolution influencing, NDI upgrade synchronization
8. Hybrid agile/ plan-driven system	C4ISR system	Full ICM; encapsulated agile in high change, low-medium criticality parts (Often HMI, external interfaces)	Full ICM, three-team incremental development, concurrent V&V, next-increment rebaselining
9. Multi-owner system of systems	Net-centric military operations	Full ICM; extensive multi-owner team building, negotiation	Full ICM; large ongoing system/software engineering effort
10. Family of systems	Medical device product line	Full ICM; Full stakeholder participation in product line scoping. Strong business case	Full ICM. Extra resources for first system, version control, multi-stakeholder support
11. Brownfield	Incremental legacy phaseout	Re-engineer/refactor legacy into services	Incremental legacy phaseout
12a. Net-Centric Services—Community Support	Community Services or Special Interest Group	Filter, select, compose, tailor NDI	Evolve tailoring to meet community needs
12b. Net-Centric Services—Quick Response Decision Suppport	Response to competitor initiative	Filer, select, compose, tailor NDI	Satisfy quick response; evolve or phase out

Legend:
C4ISR: Command, Control, Computing, Communications, Intelligence, Surveillance, Reconnaissance; **CDR:** Critical Design Review; **DCR:** Development Commitment Review; **FRP:** Full-Rate Production ; **HMI:** Human-Machine Interface ; **HW:** Hardware; **IOC:** Initial Operational Capability; **LRIP:** Low-Rate Initial Production; **NDI:** Non-Development Item; **SW:** Software; **V&V:** Verification and Validation.

Using the Process Decision Tables

Selecting and Tailoring a Decision Table Case. The general ICM chart has many decision options available, but its risk-driven approach results in most projects having simpler special cases that can usually be determined during the project's Exploration phase. The decision table indicates the most common special cases. But it is not exhaustive. If your risk pattern is not included, look at the closest approximation to it, and think through an appropriate risk-driven variant of it.

Even if your pattern is included in the table, there may be special aspects of your project that require additional tailoring (e.g., multiple cultures, distant time zones, legacy constraints). The decision table should be a stimulus to thinking, not a substitute for it. In addition, many complex projects have simpler pieces in them. Rather than apply a one-size-fits-all process to all of them, often you can architect the system to use simpler processes on the simpler parts.

The rest of this section describes each of the ICM special cases and typical risks associated with each case.

Case 1: Use NDI (Non-Development Items). Suppose you have an application for which an appropriate NDI (COTS, open source, reuse library, customer-furnished package) solution is available, and other options of developing perhaps a better version yourself or outsourcing such a development. Even if you produce a better solution (frequently not the case), you will generally incur more expense and take longer to begin to capitalize on its benefits. And you will often find that the NDI package has features that you hadn't realized you would need, but are there when you need them.

On the other hand, there are risks that may disqualify some NDIs. They may be overly complex for your needs, incompatible with your other applications, or highly volatile. See the discussion in Section 33.1 of *Software Engineering Economics* (Boehm, 1981) for more about the pros and cons of NDI solutions.

Case 2: Pure Agile Methods. If your project is small (less than 10 people) and its criticality involves the loss of discretionary vs. essential funds, a pure agile method such as Extreme Programming (Beck, 1999), Scrum (Schwaber, 2002), or Crystal (Cockburn, 2002) is generally best if you have relatively high-capability, agile-ready personnel. The risks of using a more formal, plan-driven approach are less adaptability to change and belated feedback on the product's capabilities. The biggest risk is to try to develop the application all at once for several months, and then find that it is a mismatch to the users' needs. Agile developers have found that the best way to address this risk is to organize the project into short (2-6 week) delivery increments that may be incomplete, but provide early useful capabilities. Any flaws can then be detected early and fixed in the next increment (for very high criticality applications, this would not be acceptable). On a small project it is also easy to set up a daily build and regression test structure that identifies integration problems early when they are easier to fix.

Some risks of using the agile approach is that it is harder to write a contract for what is being developed; that it may sub optimize for early success by using unscalable capabilities (fourth-generation languages, running all in main memory); or high personnel turnover. See (Boehm and Turner, 2004), pp. 121-128 for an example risk analysis of an agent-based event planning system, ending with a decision to go agile.

Case 3: Architected Agile. For medium-size (20-80 people), medium complexity (reasonably mature and scalable technology; largely compatible shareholders), agile methods can be scaled using an Architected Agile approach with early investment in a largely change-prescient architecture and user/developer/customer team building. For relatively stable projects (0.3-1% change/month), plan-driven methods can be used with low risk. But for higher rates of changes (1-10%/month), a more agile approach is less risky. A risk analysis of a 50-person, medium sized architecture-based agile supply chain management project is provided on pages 106-121 of (Boehm and Turner, 2004). A number of organizations in such areas as corporate infrastructure, medical, aerospace, and ERP applications have reported significant gains in adaptability and quality of the Architected Agile approach over plan-driven methods for such projects. However, others that had less capable and agile-ready people, less management and customer commitment, and less up-front architecture investment have not. (Boehm, 2007)

Case 4: Formal Methods. Formal methods involve the development and verification of a precise mathematical specification of the behavior of a hardware and/of software systems; an implementation of the system in formal-semantics-based languages; and a mathematical proof that the implementation is exactly equivalent to the specification (no more; no less). Such methods are expensive relative to standard commercial practice and require scarce high-capability personnel, but are inexpensive relative to a massive product recall, expensive lawsuits, or a massive loss of valuable and confidential information.

Current formal methods generally require repeating the expensive mathematical proof process whenever the specification or implementation is changed, making the approach less viable for systems with highly volatile requirements. In general, non-developmental items are not precisely specified and verified enough to be safely used in such systems. Also, formal methods have limited scalability; almost all fully-verified software systems have less than 10,000 lines of code. However, some progress is being made toward modularizing such systems so that implementations and proofs can be built up incrementally via lower-level lemmas and theorems.

Case 5: Software Embedded Hardware Component. The application classes above have been mostly software-intensive. The differences in economic and risk patterns for hardware-intensive projects (Boehm and Lane, 2007) will create different risk-based special cases of the ICM. Once a project commits to a particular manufacturing approach and infrastructure, the hardware cost of change will be much higher. And the primary costs at risk are in development and manufacturing, particularly if the component is cheaper to replace than to fix or if fixes can be accomplished by software workarounds. Thus, the ICM Stage I activities of producing and validating detailed specifications and plans are much more cost-effective than for the agile cases above. They will often go beyond the usual level of detail (Critical Design Review versus evidence-based Preliminary Design Review) for an ICM Development Commitment Review (DCR), since so many of the details are likely to be major sources of rework expenditures and delay if wrong. And after Initial Operational Capability (IOC) development, the rework risks generally dictate an incremental low rate initial production phase before a full-rate production phase in Stage II. In case the component is planned to evolve to subsequent hardware-software reconfigurations, the ICM approach of having a concurrent systems engineering team developing specifications and plans for the "N+1st " increment could be adopted. The length of these later increments will be driven by the product's marketplace or competitive situation.

Case 6: Indivisible IOC. More complex hardware intensive systems, such as aircraft, may have a great deal of software that can be incrementally developed and tested, but may have an indivisible hardware IOC that must be developed as a unit before it can be safely tested (e.g., an automobile or aircraft's braking system or full set of safety-critical vehicle controls). Relative to the cost and duration of the software increments, the indivisible hardware IOC has two significant sources of risk:

- It cannot fit into smaller software increments
- It cannot drop required hardware features to meet IOC schedule/cost/quality as independent variable.

The first can be addressed by synchronizing the hardware IOC with the Nth software increment (e.g., (Rechtin and Maier, 1997) osculating orbits for hardware and software).

If some combination of schedule/cost/quality is truly the project IOC's independent variable, then one does not want to commit to a most-likely combination of schedule/cost/quality, as there will be a roughly 50% chance of an overrun or quality shortfall in completing the indivisible IOC. Rather, it is better to determine a conservative IOC cost and schedule for meeting the quality objective, and use the difference between the conservative cost and schedule and the most-likely cost and schedule as a risk reserve. The best way to do this is to use the conservative cost and schedule as the IOC target, and to determine a set of desired but non-essential, easy to drop software features that could be developed within the risk reserve. Then, if the most-likely indivisible IOC capability begins to overrun, some desired software features can be dropped without missing the scheduled delivery time and cost. There should also be a strong award-fee incentive for the developers to minimize the number of features that need to be dropped.

Case 7: NDI-Intensive. Our experiences in developing USC web-service applications between 1996 and 2004 was that they went from 28% of the application's functionality being delivered by NDI components to 80% (Yang et al, 2005). A similar trend was identified by the 2001 Standish Report, which reported that 53% of the functionality of commercial software applications was being delivered by NDI components in 2000 (Standish, 2001). The economics of NDI-intensive systems dictates a bottom-up versus a top-down approach to system development, in which the capability envelope of the NDI determines the affordable requirements, rather than a top-down requirements-to-capability approach. A large supply-chain management system may need to choose among several NDI candidates each for such functions as inventory control, trend analysis, supplier/customer relations management, transportation planning, manufacturing control, and financial transactions; and evaluate not only the candidates' cost/performance aspects, but also their interoperability with each other and with the corporation's legacy infrastructure. Besides NDI assessment, other significant sources of effort can be NDI tailoring, NDI integration, and effect of NDI version volatility and obsolescence; see (Yang et al, 2005).

A particular challenge in Stage II is the effect of NDI volatility and obsolescence. Surveys have indicated that commercial NDI products have a new release about every 10 months, and that old releases are supported by the vendor for about 3 releases. Some large systems have had about 120 NDI components, indicating that about 12 components will have new releases each month, and that not upgrading will leave each component unsupported in about 30 months. In such cases, a great deal of attention needs to be paid to upgrade synchronization, and to pro-active NDI evolution influencing. Some large organizations synchronize their NDI upgrades to their major re-training cycles of about 12-18 months. For additional NDI best practices, see (Wallnau et al, 2002).

Case 8: Hybrid Agile/Plan-Driven System. Some large, user-intensive systems such as C4ISR systems, air traffic control systems, and network control systems have a mix of relatively stable, high-criticality elements (sensors and communications; key business logic) and more volatile, more moderately critical elements such as GUI displays, electronic warfare countermeasures, and interfaces with externally evolving systems. As many acquisitions of this nature have shown, it is highly risky to apply one-size-fits-all processes and incentive structures to these differing classes of elements. Instead, in Stage I, it is important to determine which system elements belong in which class along with the other functions of understanding needs, envisioning opportunities, NDI assessment, scoping the system, and determining feasible requirements and increments performed for complex systems in Stage I; to architect the system to encapsulate the volatile parts for development by agile teams; and to organize to use plan-driven development for the other elements of the overall system. In Stage II, the cycles for the agile teams are likely to be significantly shorter than those for the plan-driven teams.

Case 9: Multi-Owner System of System. In this situation, your goal is to integrate a set of existing systems (or guide and evolve the integration of a set of existing systems). These systems are primarily developed, owned, and maintained by an organization other than the one that is attempting to manage and guide the set of systems as a system of systems. Because of the independence of these constituent systems, the SoS organization has little or no formal control over the processes used to maintain and evolve the constituent systems. The SoS may be an enterprise wide business SoS, with the constituents being primarily COTS products along with some legacy applications. Or it may be Department of Defense (DoD) warfighting SoS, where the constituent legacy systems are integrated to increase capabilities on the battle field.

Traditional systems engineering (SE) activities are typically tailored for the SoS case to define an SoS architecture, better coordinate the activities of multiple systems in migrating to the SoS architecture, and provide synchronization points for the SoS. In (OUSD AT&L, 2008), pilot studies have shown that many DoD SoS have re-organized the traditional SE activities into a set of seven core elements: 1) translating capability objectives, 2) understanding systems and relationships, 3) assessing performance to capability objectives, 4) developing, evolving, and maintaining SoS design, 5) monitoring and assessing changes, 6) addressing new requirements and options, and 7) orchestrating updates to SoS. Further analysis shows, that these elements map fairly well to the hybrid agile/plan-driven case (Case 8) at the SoS level.

What makes this case different from Case 8, is that each of the constituent systems is using their own processes, which could be any of the above cases, depending on the scope, complexity, and characteristics of the constituent system. What is key is that the Case 9 extends Case 8 to include information or participation of the constituent systems in the agile planning activities and lets the “battle rhythm” of the constituent system increments guide the SoS plan-driven and V&V activities.

Case 10: Family of Systems. Families of systems are typically a set of systems that belong to a product line and can be easily used to customize a solution for a given need. This might be a suite of medical devices or a suite of applications to customer support. This is often the set of systems developed by a vendor that become the NDI components for CASE 7 above. Again, the rigor required for the SoS case is present here. However, in this situation, the family of systems is typically owned and evolved by a single organization/vendor and presents a case where the owning organization has much more control over the evolution of the components of the family of systems, thus possibly reducing some risks and allowing the ICM process to be a little more streamlined.

Case 11: Brownfield: A Brownfield counterexample involved a major US corporation that used a Greenfield systems engineering and development approach to develop a new Central Corporate Financial System to replace a patched-together collection of strongly-coupled and poorly documented COBOL business data processing programs. The system included an early Enterprise Resource Planning (ERP) system that was well matched to the corporation's overall financial needs, but not to its detailed business processes, which included various workarounds to compensate for difficulties with the legacy software. The new system was well organized to support incremental implementation, but was dropped at a cost of $40 million after two failed tries to provide continuity of service, due primarily to the infeasibility of incrementally phasing out the legacy software and business processes compatibly with the new system's incremental capabilities.

The ICM can be used to organize systems engineering and acquisition processes in ways that better support Brownfield legacy software re-engineering so that organizations can better provide continuity of services. In particular, its concurrent activities of Understanding Needs, Envisioning Opportunities, System Scoping and Architecting, Feasibility Evidence Development, and Risk/Opportunity Assessment enable projects to focus specifically on their legacy system constraints and on opportunities to deal with them.

The application of the ICM to the Brownfield corporate counterexample situation would have avoided the failures of the Greenfield development. Its Understanding Needs activity would have determined the ways in which the legacy system had intertwined financial and other business services. For examples, Project Services included budgeting and scheduling, work breakdown system accounting, earned value management intertwined with requirements, version and configuration management; Contract Services included expenditure category management, billing, and receivables management intertwined with deliverables management and engineering change proposal tracking; with similar intertwining in Personnel Services and Marketing Services.

The ICM's Envisioning Opportunities activity would have identified opportunities to use large-scale refactoring methods to decouple the financial and non-financial elements of these services in ways that would make it feasible for them to interoperate with a Financial Services element. Its System Scoping and Architecting activity would have repackaged the legacy software and developed the new architecture around financial services that would support incremental phaseout, and its Feasibility Evidence Development activity would have assessed any outstanding risks and covered them with risk management plans that would be tracked to ensure project success.

A good example of a Brownfield methodology with several case study examples is provided in (Hopkins and Jenkins, 2008).

Case 12a: Net-Centric Services—Community Support. Net-Centric Services for community support tend to fall into two categories. One involves community service organizations providing needed services for child care, elder care, handicapped, homeless, or jobless people. Their information processing service needs include tracking of clients, volunteers, donors, donations, events, and provision of news and communication services. These used to require considerable programming to realize, but now can be realized by combining and tailoring NDI packages offering such services. The other category of net-centric services for community support involves special interest groups (professionals, hobbyists, committees, ethnic groups) with similar needs for tracking members, interests, subgroups, events, and provision of news and communication services. Development of such capabilities involves much more prototyping and much less documentation than is needed for more programming-oriented applications; for example, the internals of the NDI packages are generally unavailable for architectural documentation, and the resulting system capabilities are more driven by NDI capabilities than by prespecified requirements documents.

A good example of how the ICM provides support for such net-centric services is provided in (Boehm and Bhuta, 2008) and the entire journal provides additional approaches and case studies of net-centric services development.

Case 12b: Net-Centric Services: Quick Response Decision Support. Another form of net-centric services involves rapidly configuring a capability to analyze alternative decision options in response to a competitor's initiative. These might involve a competitor beginning to penetrate a business's home territory, and might involve the need to evaluate the relative costs and benefits of alternative strategies of expanding services, expanding service locations, or repricing products or services. NDI packages to help analyze geographical, logistical, human resources, and financial implications of alternative competitive response decisions can be rapidly configured to provide a quick-response capability for converging on a decision. Once the decision is made, the resulting capability may be phased out, or evolved into a more general and maintainable capability for similar future situations. Again, the November/December 2008 special issue of IEEE Software on "Opportunistic Software Systems Development" provides additional perspectives on this area of net-centric services development.

Conclusions and Future Plans

The Incremental Commitment Model described in this article builds on experience-based critical success factor principles (stakeholder satisficing, incremental definition, iterative evolutionary growth, concurrent engineering, risk management) and the strengths of existing V, concurrent engineering, spiral, agile, and lean process models to provide a framework for concurrently engineering system-specific critical factors into the systems engineering and systems development processes. It provides capabilities for evaluating the feasibility of proposed solutions; and for integrating feasibility evaluations into decisions on whether and how to proceed further into systems development and operations. Previous work has shown that the critical success factors inherent in the ICM have been found in many successful programs (Boehm and Lane, 2007). In this paper, we have described how the ICM can be applied to many types of systems. Continuing efforts are underway to evaluate the efficacy of the ICM as well as its fallibility.

References

Beck, K. 1999. *Extreme programming explained*, Addison Wesley.

Boehm, B. 1981. *Software engineering economics*, Prentice Hall.

Boehm, B. 2007. “Agility and quality”, Keynote Presentation, ICSE 2007 Workshop on Software Quality.

Boehm, B. and Bhuta, J. (2008). Balancing Risks and Opportunities in Component-Based Software Development. IEEE Software, November/December 2008, pp. 56-63.

Boehm, B. and Lane, J. 2007. Using the incremental commitment model to integrate system acquisition, systems engineering, and software engineering, *CrossTalk*, Vol. 20, No. 10.

Boehm, B. and Turner, R. 2004. Balancing agility and discipline: a guide for the perplexed. Addison-Wesley, Boston.

Cockburn, A. 2002. *Agile software development*, Addison Wesley.

Hopkins, R. and Jenkins, K. (2008). *Eating the IT Elephant: Moving from Greenfield Development to Brownfield*, IBM Press, 2008

Marenzano, J. et al. 2005. Architecture reviews: practice and experience. *IEEE Software*, March/April 2005, pp. 34-43.

Office of the Under Secretary of Defense for Acquisition, Technology and Logistics (OUSD AT&L), 2008. Systems engineering for systems of systems, Version 1.0. Washington, DC: Pentagon.

Pew, R. W., and Mavor, A. S. 2007. Human-System Integration in the System Development Process: A New Look. National Academy Press.

Schwaber, K. and Beedle, M. 2002. *Agile Software Development with Scrum*. Prentice Hall.

Standish Group 2001. *Extreme Chaos*.

Wallnau, K., Hissam, S., and Seacord, R. 2002. *Building Systems from Commercial Components*. Addison Wesley.

BIOGRAPHY

Barry Boehm, Ph.D., is the TRW professor of software engineering and co-director of the Center for Systems and Software Engineering at the University of Southern California. He was previously in software engineering, systems engineering, and management positions at General Dynamics, Rand Corp., TRW, and the Defense Advanced Research Projects Agency, where he managed the acquisition of more than $1 billion worth of advanced information technology systems. Dr. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

Jo Ann Lane is currently a USC CSSE Principal supporting software and systems engineering and research activities. In this capacity, she is currently working on a cost model to estimate the effort associated with system-of-system architecture definition and integration. She is also a part time instructor teaching software engineering courses at San Diego State University. Prior to this, she was a key technical member of Science Applications International Corporation's Software and Systems Integration Group responsible for the development and integration of software-intensive systems and systems of systems.

Supannika Koolmanojwong is currently a PhD student at the University of Southern California. Her primary research area focuses on software process improvement, software process modeling and net-centric services. She is currently developing Incremental Commitment Model Electronic Process Guide for software development and mentoring 16 student groups using Incremental Commitment Model to develop e-services projects. Prior to this, she was a RUP/OpenUp Content Developer at IBM Software Group.