



Next Generation Processes for Achieving Operationally Responsive Ground Systems

**Barry Boehm, USC-CSSE
GSAW 2008; LA SPIN
April 2, 2008**

Charts include explanatory notes

A major challenge for achieving operationally responsive ground systems is the slowness of current processes for adapting complex software-intensive systems to increasingly rapid change. A recent analysis of change processing times for two complex, high-assurance software-intensive systems, one of which included a ground station, showed average times of 27 workdays for within-group changes, 48 workdays for cross-group changes, and 141 workdays for changes involving contract modifications.

This talk will present a next-generation synthesis of the spiral model and other leading process models into the Incremental Commitment Model (ICM) being piloted or considered for adoption in some parts of DoD. The ICM emphasizes architecting systems to encapsulate subsystems undergoing the most rapid change and having them implemented by agile developers; and architecting the incremental development process by having agile systems engineers handling longer-range change traffic to rebaseline the plans for future increments while largely plan-driven teams develop and continuously V&V the current increment. Further information on the ICM in the context of integrating systems and software engineering can be found at <http://csse.usc.edu/events/2008/ARR/pages/material.html>



Thanks to Affiliates for Your USC-CSSE Support

- **Commercial Industry (12)**
 - Bosch, Cisco, Cost Xpert Group, Galorath, Group Systems, IBM, Intelligent Systems, Microsoft, Motorola, Price Systems, Softstar Systems, Sun
- **Aerospace Industry (8)**
 - BAE Systems, Boeing, General Dynamics, Lockheed Martin, Northrop Grumman(2), Raytheon, SAIC
- **Government (7)**
 - DACS, NASA-Ames, NSF, OSD (AT&L) - S&SE, US Army Research Labs, US Army TACOM, USAF Cost Center
- **FFRDC's and Consortia (6)**
 - Aerospace, FC-MD, IDA, JPL, SEI, SPC
- **International (2)**
 - Institute of Software, Chinese Academy of Sciences, Samsung

03/19/2008



Outline

- ➔ • **Motivation and context**
 - Emerging challenges for future space and ground systems
 - Software key to rapid operational responsiveness
 - Difficulties in integrating systems and software engineering

- **State of systems and software engineering (S&SE) integration**
 - Current SysE guidance: outdated assumptions; inhibitors to software best practices

- **Incremental Commitment Model (ICM) overview**
 - ICM nature, origin, and principles
 - ICM process views and applicability to Young memo

- **Conclusions: IS&SE with ICM**
 - References; Acronyms; Backup charts

03/19/2008

©USC-CSSE

3

The context of this presentation includes a summary of emerging challenges for future space and ground systems. Key among these challenges is the need for rapid operational responsiveness, which often depends on the responsiveness of the software. A study performed in 2007 by USC-CSSE for the U.S. Department of Defense (DoD) on integrating systems and software engineering found significant shortfalls of current DoD systems engineering guidance in integrating systems and software engineering for both current and likely future DoD systems. It also evaluated the Incremental Commitment Model (ICM) for integrating systems, software, and human factors engineering recommended in a recent DoD-sponsored National Research Council study [Pew and Mavor, 2007].

As recommendations, it presented a mixed strategy of incremental improvements to current DoD acquisitions, pilot projects to explore new approaches such as the ICM, and incremental improvement of DoD guidance documents and education and training initiatives. One element of the mixed strategy underway is an effort led by USC-CSSE to develop a guidebook for the use of the ICM to integrate systems and software engineering on DoD projects, in concert with several current DoD projects piloting all or parts of the ICM.

Concurrently, USC-CSSE will be developing commercial and educational versions of the ICM, developing with IBM an OpenUP electronic process guide for the ICM, and testing the educational version in Fall 2008 in its annual series of 20 real-client, 6-8 person, MS-level student team, e-services projects.



Challenges for Future Space and Ground Systems

- **Parts of multi-owner, net-centric systems of systems**
 - Frequent priority negotiation; security needs
- **Multi-mission ground systems**
 - Further priority negotiation
- **Dynamic recomposability from microsats**
 - Interoperability at every network layer
- **Intelligent, adaptive, knowledge-based, autonomous**
 - Software/agent-intensive, V&V challenges
- **All of the above plus rapid operational responses**
 - In asymmetric warfare context

03/19/2008

©USC-CSSE

4

Future space and ground systems will less and less frequently be dedicated and self-contained. Most will be parts of several multi-owner, net-centric systems of systems creating conflicting new demands on the space and ground systems. This will require rapid negotiation of change priorities; even more for multi-mission ground systems. Being net-centric implies that the systems will be potential causes and recipients of security vulnerabilities, placing heavier demands on security than before.

Future satellite system concepts also include dynamic satellite configurations with new capabilities arriving and leaving, implying the need for an interoperability framework and rapid reconfiguration not only of the satellite components but of their support systems and software. Further goals of more intelligent and autonomous satellites create further challenges in verifying and validating their performance.

Combining these with the need for rapid operational responsiveness creates even more stringent challenges, especially in situations of asymmetric warfare, as discussed next.



Asymmetric Conflict and OODA Loops

•Adversary

- Picks time and place
- Little to lose
- Lightweight, simple systems and processes
- Can reuse anything

•Defender

- Ready for anything
- Much to lose
- More heavy, complex systems and processes
- Reuse requires trust



03/19/2008

©USC-CSSE

5

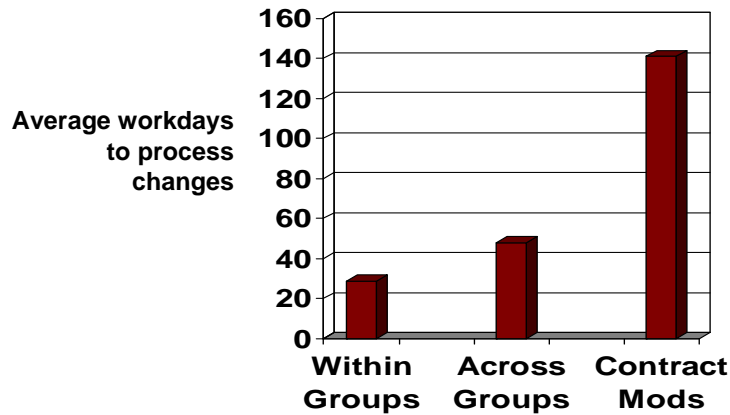
In the early stage of the Iraq war, DoD forces were phenomenally successful. They could pick the time and place of attacks, and their highly superior command, control, communications, computing, intelligence, surveillance, and reconnaissance (C4ISR) capabilities enabled them to perform observe-orient-decide-act (OODA) loops well inside those of their conventional Iraqi army adversaries.

However, in the later stages of the Iraq war, DoD has faced much more serious challenges that are representative of many future conflict situations. Their Iraqi adversaries are diffuse. They can pick the time and place of an attack. They have little to lose, and can use very agile, lightweight systems and processes, while being able to use powerful but insecure open-source software at relatively little risk.

On the other hand, DoD forces must be ready for anything and be able to defend much more valuable assets. This requires the more heavyweight systems and processes currently required to develop and operate high-assurance systems, and restricts the reuse of untrusted components. Even for individual systems, this causes significant challenges in turning OODA loops inside those of one's adversaries, but as seen next, the challenges will be even higher for complex systems of systems.



Average Change Processing Time: Two Complex Systems of Systems



Incompatible with turning within adversary's OODA loop

03/19/2008


©USC-CSSE

6

This chart shows data from two complex systems of systems that illustrates the challenge of executing tight OODA loops that involve coordinating changes across closely coupled systems of systems [Boehm, 2007]. The average time in workdays (longer than in calendar days) was 27 workdays to close a change request within an individual platform or capability group; 48 workdays if the change required coordination across multiple platform or capability groups; and 141 workdays if the change involved a change in performers' contracts. These were typical high-content U.S. build-to-specification contracts, which [Hall, 1976] found averaged 10 times as long as high-context French contracts. Corroborative data was provided at the workshop in [Schroeder, 2007], who found that the length of such changes was proportional to the number of contracts requiring changes. Other Workshop attendees reported even longer contract change turnaround times for large complex systems. Clearly, improvements are needed in change-facilitating architectures, processes, and contracts if DoD is to stand any chance of keeping its change-adapting OODA loops within those of its adversaries.



Outline

- **Motivation and context**
 - Emerging challenges for future space and ground systems
 - Software key to rapid operational responsiveness
 - Difficulties in integrating systems and software engineering
-  • **State of systems and software engineering (S&SE) integration**
 - Current SysE guidance: outdated assumptions; inhibitors to software best practices
- **Incremental Commitment Model (ICM) overview**
 - ICM nature, origin, and principles
 - ICM process views and applicability to Young memo
- **Conclusions: IS&SE with ICM**
 - References; Acronyms; Backup charts

03/19/2008

©USC-CSSE

7

The context of this presentation includes a study performed in 2007 by USC-CSSE for the U.S. Department of Defense (DoD) on integrating systems and software engineering. The study evaluated the strengths and shortfalls of current DoD systems engineering guidance in integrating systems and software engineering for both current and likely future DoD systems. It also evaluated the Incremental Commitment Model (ICM) for integrating systems, software, and human factors engineering recommended in a recent DoD-sponsored National Research Council study [Pew and Mavor, 2007].

As recommendations, it presented a mixed strategy of incremental improvements to current DoD acquisitions, pilot projects to explore new approaches such as the ICM, and incremental improvement of DoD guidance documents and education and training initiatives. One element of the mixed strategy underway is an effort led by USC-CSSE to develop a guidebook for the use of the ICM to integrate systems and software engineering on DoD projects, in concert with several current DoD projects piloting all or parts of the ICM.

Concurrently, USC-CSSE will be developing commercial and educational versions of the ICM, developing with IBM an OpenUP electronic process guide for the ICM, and testing the educational version in Fall 2008 in its annual series of 20 real-client, 6-8 person, MS-level student team, e-services projects.



Current SysE Guidance: Outdated Assumptions

Example: SysE Plan Preparation Guide

Sometimes OK for hardware; generally not for software

- An omniscient authority pre-establishes the requirements, preferred system concept, etc. (A4.1, 4.2, 4.4)
- Technical solutions are mapped and verified with respect to these (A4.1, 4.2, 4.4)
- They and technology do not change very often (A4.2, 4.5, 6.2)
 - Emphasis on rigor vs. adaptability
- The program has stable and controllable external interfaces (A4.5)
- Project organization is function-hierarchical and WBS-oriented (A3.1, 3.2)
- All requirements are equally important (A4.2)
- Reviews event/product-based vs. evidence-based (A5.2)
- The program's critical path can be defined up front (A6.1)
- Can achieve optimum readiness at minimum life-cycle cost (A6.5)
 - No slack for contingencies

03/19/2008

©USC-CSSE

8

The SEP guidelines make a number of implicit assumptions which are increasingly invalid as requirements become more emergent than prespecifiable, as requirements and technology undergo more rapid change, and as capabilities and key performance parameters become at least as success-critical as functions and products.

Even before Milestone A, Section 4 assumes that requirements are in place as a basis for traceability, verification, and allocation, even before key technologies and COTS products are fully evaluated. This tends to encourage premature specification of requirements, some of which will be found later to be infeasible or inappropriate.

Project organization and Integrated Product Teams (IPTs) are assumed to follow functional and Work Breakdown Structure (WBS) hierarchies. As discussed in charts 8 and 9, functional hierarchies are incompatible with layered software structures and inhibit integration of systems and software engineering. IPTs are also needed for non-functional cross-cutting issues such as safety, security, and end-to-end performance.

Event- and product-based reviews are better than schedule-based reviews, but they frequently overfocus on presenting functional diagrams without providing any evidence that a product built to satisfy the functions would have adequate performance over a range of stressing scenarios. Evidence of feasibility should also be produced, evaluated, and reviewed for adequacy.



System/Software Architecture Mismatches - Maier, 2006

- | | |
|--|---|
| <ul style="list-style-type: none">• System Hierarchy<ul style="list-style-type: none">– Part-of relationships; no shared parts– Function-centric; single data dictionary– Interface dataflows– Static functional-physical allocation | <ul style="list-style-type: none">• Software Hierarchy<ul style="list-style-type: none">– Uses relationships; layered multi-access– Data-centric; class-object data relations– Interface protocols; concurrency challenges– Dynamic functional-physical migration |
|--|---|

03/19/2008

©USC-CSSE

9

A valuable perspective on the mismatches between traditional hardware-oriented systems engineering architectural structures and modern software architectural structures has been provided in [Maier, 2006]. First, traditional systems engineering methods functionally decompose the systems architecture into one-to-many “part-of” or “owned-by” relationships, while modern software methods organize system capabilities as layers of many-to-many service-oriented relationships. An example of the difficulties this causes is given in the next chart.

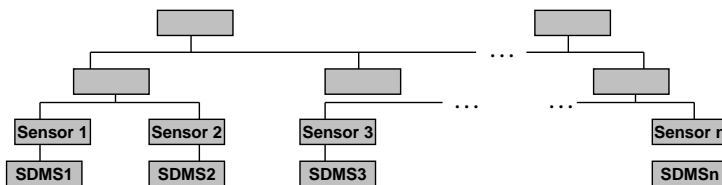
Second, traditional systems engineering approaches collect data into a single data dictionary and store it for common use. This was OK for sequential software, but for interrupt-driven concurrent software, it could cause data being used by a program to be modified while the program was interrupted, causing losses in software integrity. Modern object-oriented software approaches address this problem by using “information-hiding” techniques to store local data inside objects, requiring more up-front data engineering.

Third, hardware interfaces tend to be static: sensor data flows down a wire, and the sensor-system interface can be represented by its message content, indicating the data’s form, format, units of measure, precision, frequency, etc. In a net-centric world, interfaces are much more dynamic: a sensor entering a network must go through a number of protocols to register its presence, perform security handshakes, publish and subscribe, etc. The next chart describes how this can result in integration problems.

Fourth, hardware relations are assumed to be static and subject to static functional-physical allocation: if the engines on one wing fail, an engine cannot migrate from the other wing to rebalance the propulsion. But in software, modules frequently migrate from one processor to another to compensate for processor failures or processing overloads.

Examples of Architecture Mismatches

- **Fractionated, incompatible sensor data management**



- **“Touch Football” interface definition earned value**
 - Full earned value taken for defining interface dataflow
 - No earned value left for defining interface dynamics
 - Joining/leaving network, publish-subscribe, interrupt handling, security protocols, exception handling, mode transitions
 - Result: all green EVMS turns red in integration

03/19/2008

©USC-CSSE

10

A frequent consequence of traditional systems engineering methods that functionally decompose the systems architecture into one-to-many “part-of” or “owned-by” relationships is to create incompatible and hard-to-modify software. This is exacerbated by the owned-by relations being propagated into work breakdown structures (WBS) and WBS-oriented management structures. For example, an aircraft has wings as parts, which have ailerons as parts, which have aileron controls as parts, which have sensors as parts, which have sensor software as parts. Further, an aircraft using active controls to stabilize an otherwise unstable aircraft will rely on atmospheric, propulsion and attitude sensors owned by other parts of the aircraft hierarchy to accomplish the stabilization objectives.

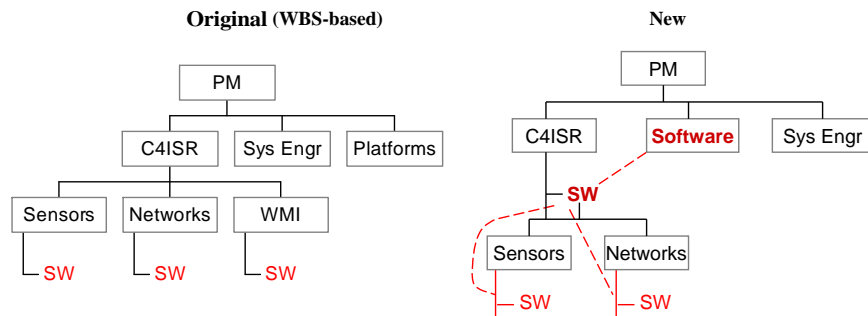
In such situations, best-of-breed hardware subsystem contractors are often selected whose sensor data management software is incompatible, causing major problems as compared to layered software providing unified data management services. And if sensor data management changes need to be coordinated across a wide variety of subsystem management chains, very slow OODA loops result. A recent program’s addressal of this problem is shown in the next chart.

A frequent consequence of static interface definitions represented by their message content is that system engineering will credit itself with full interface-definition earned value when it defines the interface message and its content. If no further earned value can be generated for defining the dynamic software interface aspects, they will tend to lose management priority and be deferred until the deficiencies are found during integration, when numerous incompatible subsystem software interface assumptions are found that are expensive and time-consuming to fix. This is a frequent explanation for all-green earned value management system (EVMS) indicators turning red during integration.



Effect of Software Underrepresentation

- Software risks discovered too late
- Slow, buggy change management
- Recent large project reorganization



03/19/2008


©USC-CSSE

11

A recent large, software-intensive system found itself encountering the type of problems discussed on the previous chart, and effectively reorganized to address them. It retained its functional part-of hierarchy for its hardware components, but created a sub-program manager for software and a dotted-line hierarchy of software chief engineers who could rapidly mobilize to address time-critical software change management issues. It also provided a software staff organization that was able to diagnose and address risks such as the touch-football interface definition problem discussed on the previous chart.



Outline

- **Motivation and context**
 - Emerging challenges for future space and ground systems
 - Software key to rapid operational responsiveness
 - Difficulties in integrating systems and software engineering
- **State of systems and software engineering (S&SE) integration**
 - Current SysE guidance: outdated assumptions; inhibitors to software best practices
-  • **Incremental Commitment Model (ICM) overview**
 - ICM nature, origin, and principles
 - ICM process views and applicability to Young memo
- **Conclusions: IS&SE with ICM**
 - References; Acronyms; Backup charts

03/19/2008

©USC-CSSE

12

The context of this presentation includes a study performed in 2007 by USC-CSSE for the U.S. Department of Defense (DoD) on integrating systems and software engineering. The study evaluated the strengths and shortfalls of current DoD systems engineering guidance in integrating systems and software engineering for both current and likely future DoD systems. It also evaluated the Incremental Commitment Model (ICM) for integrating systems, software, and human factors engineering recommended in a recent DoD-sponsored National Research Council study [Pew and Mavor, 2007].

As recommendations, it presented a mixed strategy of incremental improvements to current DoD acquisitions, pilot projects to explore new approaches such as the ICM, and incremental improvement of DoD guidance documents and education and training initiatives. One element of the mixed strategy underway is an effort led by USC-CSSE to develop a guidebook for the use of the ICM to integrate systems and software engineering on DoD projects, in concert with several current DoD projects piloting all or parts of the ICM.

Concurrently, USC-CSSE will be developing commercial and educational versions of the ICM, developing with IBM an OpenUP electronic process guide for the ICM, and testing the educational version in Fall 2008 in its annual series of 20 real-client, 6-8 person, MS-level student team, e-services projects.



ICM Nature and Origins

- **Integrates hardware, software, and human factors elements of systems engineering**
 - Concurrent exploration of needs and opportunities
 - Concurrent engineering of hardware, software, human aspects
 - Concurrency stabilized via anchor point milestones
- **Developed in response to DoD-related issues**
 - Clarify “spiral development” usage in DoD Instruction 5000.2
 - Initial phased version (2005)
 - Explain Future Combat System of systems spiral usage to GAO
 - Underlying process principles (2006)
 - Provide framework for human-systems integration
 - National Research Council report (2007)
- **Integrates strengths of current process models**
 - **But not their weaknesses**

03/19/2008

©USC-CSSE

13

The ICM provides a framework for concurrently engineering the various aspects of a system. A major challenge in concurrent engineering is the synchronization and stabilization of the concurrently engineered artifacts. The ICM includes a set of anchor point milestone criteria for achieving this synchronization and stabilization. Its initial phased version of the spiral model was developed to help DoD clarify “spiral development” usage in DoD Instruction 5000.2. It was refined significantly in the DoD-sponsored National Research Council study on Human-System Integration in the System Development Process [Pew and Mavor, 2007]. As shown in the next chart, its intent has been to integrate the strengths of current process models, while avoiding their weaknesses.



ICM integrates strengths of current process models But not their weaknesses

- **V-Model: Emphasis on early verification and validation**
 - But not ease of sequential, single-increment interpretation
- **Spiral Model: Risk-driven activity prioritization**
 - But not lack of well-defined in-process milestones
- **RUP and MBASE: Concurrent engineering stabilized by anchor point milestones**
 - But not software orientation
- **Lean Development: Emphasis on value-adding activities**
 - But not repeatable manufacturing orientation
- **Agile Methods: Adaptability to unexpected change**
 - But not software orientation, lack of scalability

03/19/2008

©USC-CSSE

14

The V-model was a significant step forward from the pure-sequential waterfall model in focusing effort on early verification and validation, but it remained too easy to interpret as a waterfall model, particularly for non-experts using waterfall acquisition and contracting instruments. The spiral model has also been too easy to misinterpret as a series of waterfall increments. The Rational Unified Process (RUP) and Model-Based (System) Architecting and Software Engineering (MBASE) process used the same anchor point milestones to stabilize concurrent engineering, but have been highly focused on software development processes.

As with MBASE, Lean Development takes a value-based approach to its process framework, but Lean Development has largely continued its primary focus on repeatable manufacturing processes, although an agile version has emerged for software. Overall, agile methods have provided much more adaptability to rapid change. They still focus primarily on software, and have recently found repeatable ways to scale up to about 100-person projects by focusing on more thorough architecture definition and validation, but still have scalability problems beyond there.



Process Model Principles

Principles trump diagrams

1. **Commitment and accountability**
2. **Success-critical stakeholder satisficing**
3. **Incremental growth of system definition and stakeholder commitment**
- 4, 5. **Concurrent, iterative system definition and development cycles**

Cycles can be viewed as sequential concurrently-performed phases or spiral growth of system definition

6. **Risk-based activity levels and anchor point commitment milestones**

Used by 60-80% of CrossTalk Top-5 projects, 2002-2005

Process Model Principles

The critical success factor principles underlying the ICM were derived from studies of successful and failed projects and the National Research Council's Human-System Integration study group discussion of particularly important human-system design and development principles. They are not specific to the ICM, but key considerations in any life-cycle process. The primary rationale for each principle is provided below.

1. Without stakeholder commitment and accountability for the system under development, trust and commitment of the development organizations are at risk.
2. A system's success-critical stakeholders include users, acquirers, developers, maintainers, and potentially others. If stakeholders' key value propositions are not satisfied by a proposed or delivered system, they will refuse to use it or find ways to undermine it.
3. Single-increment, big-bang system developments take too long to develop and generally produce obsolete, non-responsive systems. Or, they become swamped by change traffic in trying to fix premature commitments.
- 4,5. Similarly, sequential, non-iterative processes take too long. They also must make premature commitments to poorly-understood requirements, leading to expensive rework or unusable systems.
6. Risk management provides effective ways to prioritize activities and to determine how much of an activity is enough. It also avoids risks turning into expensive or serious problems. Anchor point milestones provide effective ways to synchronize and stabilize concurrent activities.



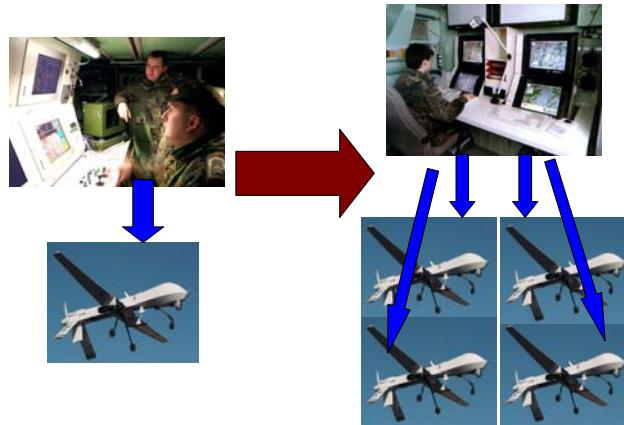
Incremental Commitment in Gambling

- **Total Commitment: Roulette**
 - Put your chips on a number
 - E.g., a value of a key performance parameter
 - Wait and see if you win or lose
- **Incremental Commitment: Poker, Blackjack**
 - Put some chips in
 - See your cards, some of others' cards
 - Decide whether, how much to commit to proceed

Incremental Commitment in Gambling

A simple metaphor to help understand the ICM is to compare ICM and incremental-commitment gambling games such as poker and blackjack, to single-commitment gambling games such as Roulette. Many system development contracts operate like Roulette, in which a full set of requirements is specified up front, the full set of resources is committed to an essentially fixed-price contract, and one waits to see if the bet was a good one or not. With the ICM, one places a smaller bet to see whether the prospects of a win are good or not, and decides whether to increase the bet based on better information about the prospects of success.

Scalable remotely controlled operations



03/19/2008

©USC-CSSE

17

Scalable remotely controlled operations – ICM Case Study

An example to illustrate ICM benefits is the Unmanned Aerial Vehicle (UAV) (or Remotely Piloted Vehicles (RPV)) system enhancement discussed in Chapter 5 of the NRC HSI report [Pew and Mavor, 2007]. The RPVs are airplanes or helicopters operated remotely by humans. These systems are designed to keep humans out of harm's way. However, current RPV systems are human-intensive, often requiring two people to operate a single vehicle. If there is a strong desire to modify the 2:1 (2 people to one vehicle) ratio to allow for a single operator and 4 aircraft (e.g., a 1:4 ratio), based on a proof-of principle agent-based prototype demo showing 1:4 performance of some RPV tasks, how should one proceed?



Total vs. Incremental Commitment – 4:1 RPV

- **Total Commitment**
 - Agent technology demo and PR: Can do 4:1 for \$1B
 - Winning bidder: \$800M; PDR in 120 days; 4:1 capability in 40 months
 - PDR: many outstanding risks, undefined interfaces
 - \$800M, 40 months: “halfway” through integration and test
 - 1:1 IOC after \$3B, 80 months
- **Incremental Commitment [number of competing teams]**
 - \$25M, 6 mo. to VCR [4]: may beat 1:2 with agent technology, but not 4:1
 - \$75M, 8 mo. to ACR [3]: agent technology may do 1:1; some risks
 - \$225M, 10 mo. to DCR [2]: validated architecture, high-risk elements
 - \$675M, 18 mo. to IOC [1]: viable 1:1 capability
 - 1:1 IOC after \$1B, 42 months

03/19/2008

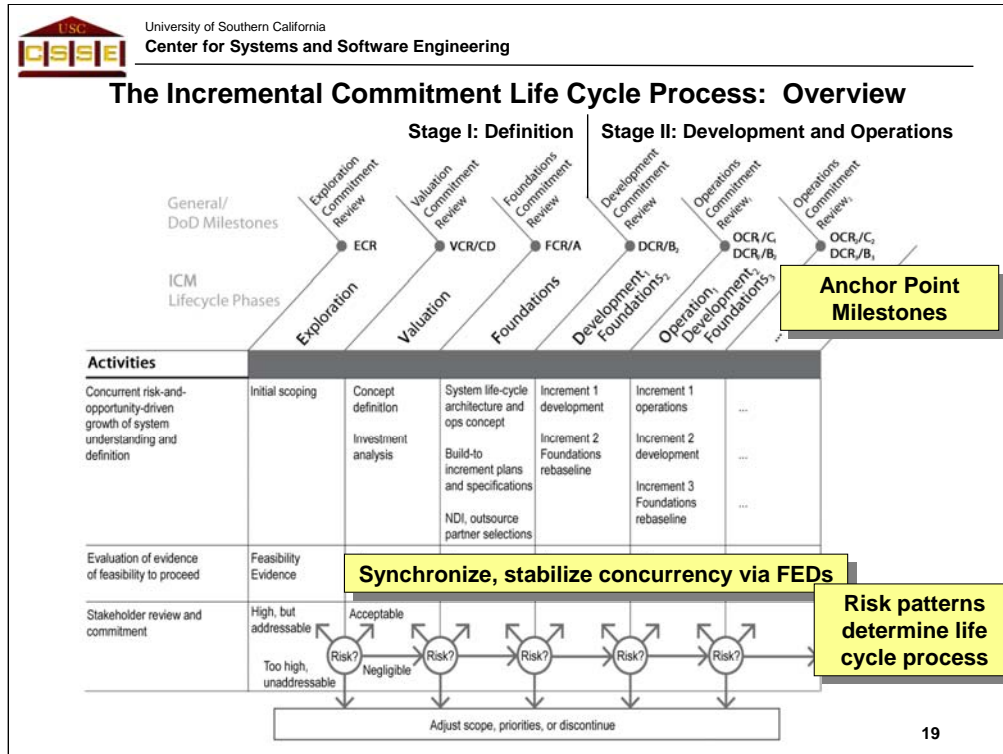
©USC-CSSE

18

Total vs. Incremental Commitment -- 4:1 RPV

This slide outlines two approaches to the RPV question: total commitment and incremental commitment. While this is a hypothetical case for developing a solution to the RPV manning problem, it shows how a premature total commitment without significant modeling, analysis, and feasibility assessment will often lead to large overruns in costs and schedule, and a manning ratio that is considerably less than initially desired. However, by “buying information” early and validating high-risk elements, technologically viable options are identified much earlier and can be provided for a much lower cost and much closer to the desired date. The ICM approach leads to the same improved manning ratio as the total commitment approach, but sooner and at a much reduced cost.

The ICM approach also employs a competitive downselect strategy, which both reduces risk and enables a buildup of trust among the acquirers, developers, and users, as emphasized in the recent UASD(AT&L) John Young memo, “Prototyping and Competition.”



The Incremental Commitment Life Cycle Process: Overview

This slide shows how the ICM spans the life cycle process from concept exploration to operations. Each phase culminates with an anchor point milestone review. At each anchor point, there are 4 options, based on the assessed risk of the proposed system. Some options involve go-backs. These options result in many possible process paths.

The life cycle is divided into two stages: Stage I of the ICM (Definition) has 3 decision nodes with 4 options/node, culminating with incremental development in Stage II (Development and Operations). Stage II has an additional 2 decision nodes, again with 4 options/node.

One can use ICM risk patterns to generate frequently-used processes with confidence that they fit the situation. Initial risk patterns can generally be determined in the Exploration phase. One then proceeds with development as a proposed plan with risk-based evidence at the VCR milestone, adjusting in later phases as necessary. For complex systems, a result of the Exploration phase would be the Prototyping and Competition Plan discussed above.

Risks associated with the system drive the life cycle process. Information about the risk(s) (feasibility assessments) supports the decision to proceed, adjust scope or priorities, or cancel the program.



Anchor Point Feasibility Rationales

- **Evidence** provided by developer and validated by independent experts that:

If the system is built to the specified architecture, it will

- Satisfy the requirements: capability, interfaces, level of service, and evolution
 - Support the operational concept
 - Be buildable within the budgets and schedules in the plan
 - Generate a viable return on investment
 - Generate satisfactory outcomes for all of the success-critical stakeholders
- All major risks resolved or covered by risk management plans
 - Serves as basis for stakeholders' commitment to proceed

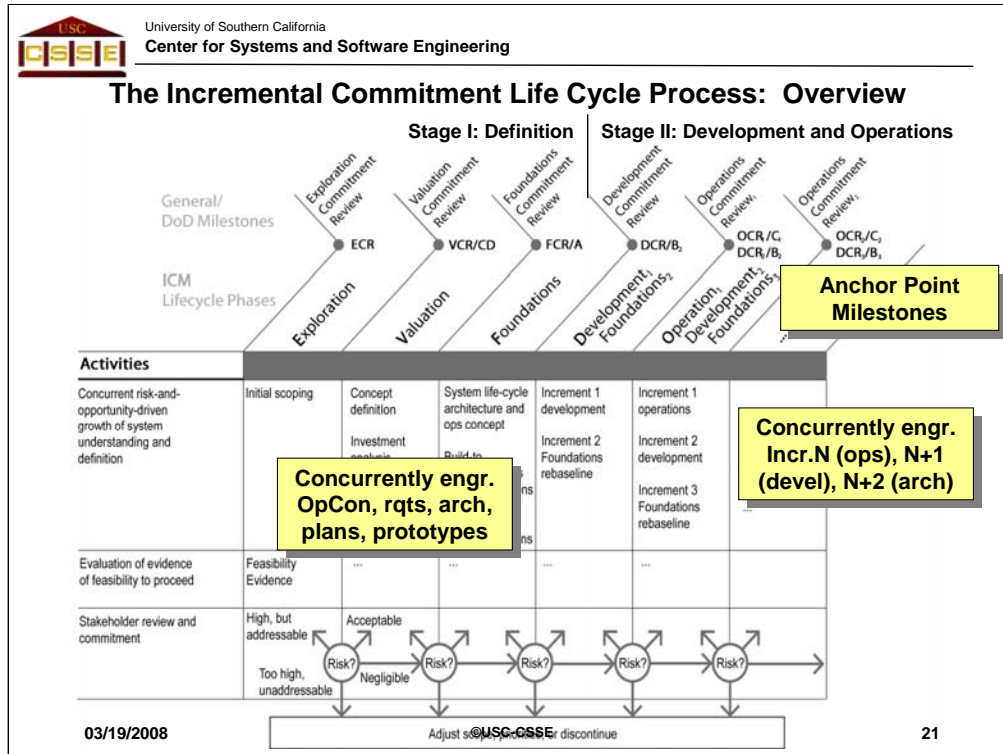
Can be used to strengthen current schedule- or event-based reviews

Anchor Point Feasibility Rationales

To make ICM 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 is focused on developer-produced *evidence*, documented in a Feasibility Evidence Description (FED), to help the key stakeholders determine the next level of commitment. At each program milestone/anchor point, feasibility assessments and the associated evidence are reviewed and serve as the basis for the stakeholders' commitment to proceed.

The FED is not just a document, a set of PowerPoint charts, or Unified Modeling Language (UML) diagrams. It is based on evidence from simulations, models, or experiments with planned technologies and detailed analysis of development approaches and projected productivity rates. The detailed analysis is often based on historical data showing reuse realizations, software size estimation accuracy, and actual developer productivity rates.

It is often not possible to fully resolve all risks at a given point in the development cycle, but known, unresolved risks need to be identified and covered by risk management plans.

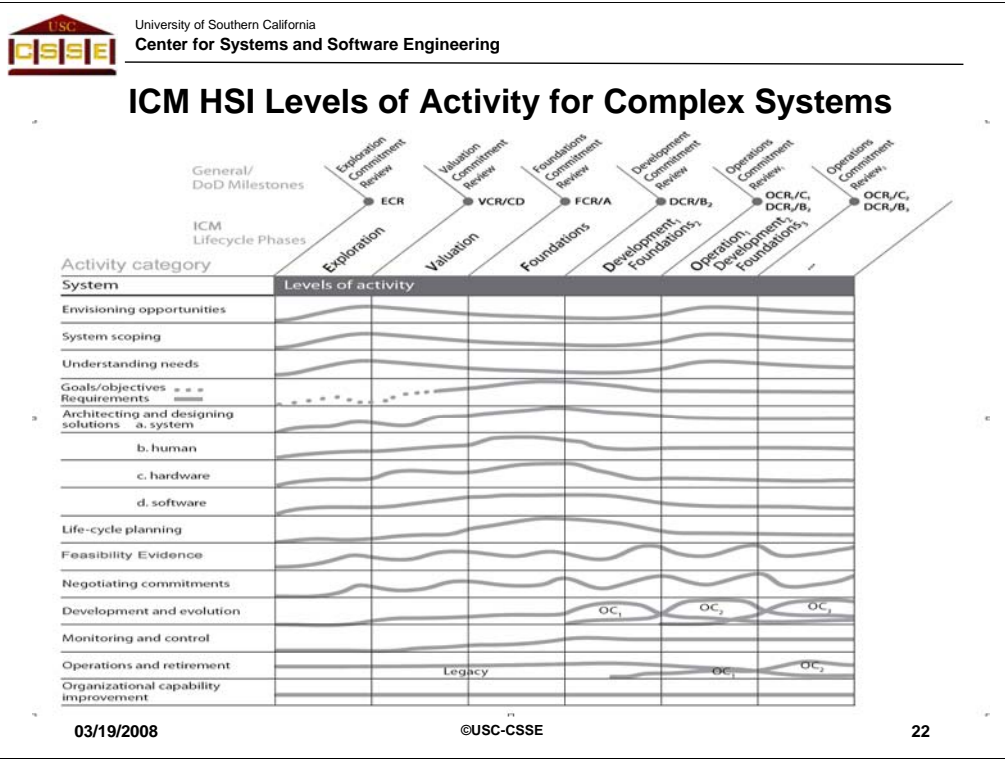


The Incremental Commitment Life Cycle Process: More on the Overview

Stage II of the Incremental Commitment Life Cycle provides a framework for concurrent engineering and development of multiple increments. More on this concurrency follows on the next slides.

Note: The term “concurrent engineering” fell into disfavor when behind-schedule developers applied it to the practice of proceeding into development while the designers worked on finishing the design. Not surprisingly, the developers encountered a high rework penalty for going into development with weak architecture and risk resolution.

“Concurrent engineering” as applied in the ICM is much different. It is focused on doing a cost-effective job of architecture and risk resolution in Stage I; and on performing stabilized development, verification, and validation of the current system increment while concurrently handling the systems change traffic and preparing a feasibility-validated architecture and set of plans for the next increment in Stage II.

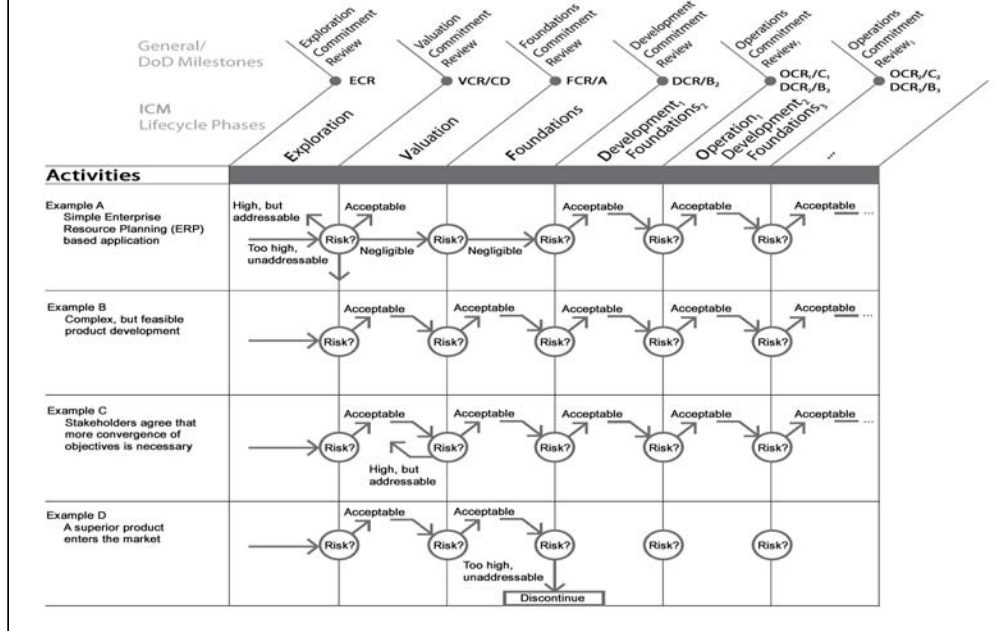


ICM HSI Levels of Activity for Complex Systems

As mentioned earlier, with the ICM, a number of system aspects are being concurrently engineered at an increasing level of understanding, definition, and development. The most significant of these aspects are shown in this slide, an extension of a similar view of concurrently engineered software projects developed as part of the RUP (shown in a backup slide).

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 this slide. The main intent of this view is to emphasize the necessary concurrency of the primary success-critical activities shown as rows. 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.

Different Risk Patterns Yield Different Processes



Different Risk Patterns Yield Different Processes

As illustrated in the four example paths through the Incremental Commitment Model in this slide, the ICM is not a single monolithic one-size-fits-all process model. As with the spiral model, it is a risk-driven process model generator, but the ICM makes it easier to visualize how different risks create different processes.

In Example A, a simple business application based on an appropriately-selected Enterprise Resource Planning (ERP) package, there is no need for a Valuation or Foundations activity if there is no risk that the ERP package and its architecture will not cost-effectively support the application. Thus, one could go directly into the Development phase, using an agile method such as a Scrum/Extreme Programming combination would be a good fit. There is no need for Big Design Up Front (BDUF) activities or artifacts because an appropriate architecture is already present in the ERP package. Nor is there a need for heavyweight waterfall or V-model specifications and document reviews. The fact that the risk at the end of the Exploration phase is negligible implies that sufficient risk resolution of the ERP package’s human interface has been done.

Example B involves the upgrade of several incompatible legacy applications into a service-oriented web-based system. Here, one could use a sequential waterfall or V-model if the upgrade requirements were stable, and its risks were low. However, if for example the legacy applications’ user interfaces were incompatible with each other and with web-based operations, a concurrent risk-driven spiral, waterfall, or V-model that develops and exercise extensive user interface prototypes and generates a Feasibility Evidence Description (described on chart 20) would be preferable.

In Example C, the stakeholders may have found during the Valuation phase that their original assumptions about the stakeholders having a clear, shared vision and compatible goals with respect the proposed new system’s concept of operation and its operational roles and responsibilities were optimistic. In such a case, it is better to go back and assure stakeholder value proposition compatibility and feasibility before proceeding, as indicated by the arrow back into the valuation phase.

In Example D, it is discovered before entering the Development phase that a superior product has already entered the marketplace, leaving the current product with an infeasible business case. Here, unless a viable business case can be made by adjusting the project’s scope, it is best to discontinue it. It is worth pointing out that it is not necessary to proceed to the next major milestone before terminating a clearly non-viable project, although stakeholder concurrence in termination is essential.



Common Risk-Driven Special Cases of the ICM

Special Case	Example	Size, Complexity	Change Rate % /Month	Criticality	NDI Support	Org. Personnel Capability	Key Stage I Activities : Incremental Definition	Key Stage II Activities: Incremental Development, Operations	Time per Build: per Increment
1. Use NDI	Small Accounting				Complete		Acquire NDI	Use NDI	
2. Agile	E-services	Low	1 – 30	Low-Med	Good; In place	Agile-ready Med-high	Skip Valuation , Architecting phases	Scrum plus agile methods of choice	<= 1 day; 2-6 weeks
3. Architected Agile	Business data processing	Med	1 – 10	Med-High	Good; most in place	Agile-ready Med-high	Combine Valuation, Architecting phases. Complete NDI preparation	Architecture-based Scrum of Scrums	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	Precise formal specification	Formally-based programming language; formal verification	1-5 days; 1-4 weeks
5. HW component with embedded SW	Multi-sensor control device	Low	0.3 – 1	Med-Very High	Good; In place	Experienced; med-high	Concurrent HW/SW engineering. CDR-level ICM DCR	IOC Development, LRIP, FRP, Concurrent Version N+1 engineering	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	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	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	Thorough NDI-suite life cycle cost-benefit analysis, selection, concurrent requirements/ architecture definition	Pro-active NDI evolution influencing, NDI upgrade synchronization	SW: 1-4 weeks; System: 6-18 months
9. Hybrid agile / plan-driven system	C4ISR	Med – Very High	Mixed parts: 1 – 10	Mixed parts; Med-Very High	Mixed parts	Mixed parts	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	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	Full ICM; extensive multi-owner team building, negotiation	Full ICM: large ongoing system/software engineering effort	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	Full ICM; Full stakeholder participation in product line scoping. Strong business case	Full ICM. Extra resources for first system, version control, multi-stakeholder support	1-2 months; 9-18 months

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: Hard ware. IOC: Initial Operational Capability. LRIP: Low-Rate Initial Production. NDI: Non-Development Item. SW: Software

03/19/2008

Common Risk-Driven Special Cases of the ICM: 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. This table contains a list of the special cases of the ICM and an example of each case. Each of the special cases is described in further detail in a companion briefing.



Example ICM Commercial Application: Symbiq Medical Infusion Pump

Winner of 2006 HFES Best New Design Award
Described in NRC HSI Report, Chapter 5



03/19/2008

©USC-CSSE

25

Example ICM HCI Application: Symbiq Medical Infusion Pump

During the National Research Council Human-System Integration study, the ICM was refined by analyzing and formalizing examples of best commercial practices of human-system integration. The Symbiq Medical Infusion Pump shown above was developed by Abbott Laboratories and won the Best New Design of 2006 Award from the Human Factors and Ergonomics Society. As evident from the views above, such devices present significant challenges in integrating high-safety hardware, software, and human factors engineering

This next-generation infusion pump is a general-purpose intravenous infusion pump (IV pump) designed primarily for hospital use with secondary, limited-feature use by patients at home. The device is intended to deliver liquid medications, nutrients, blood, and other solutions at programmed flow rates, volumes, and time intervals via intravenous and other routes to a patient. The device offers medication management features, including medication management safety software through a programmable drug library. The infuser also has sufficient memory to support extensive tracking logs and the ability to communicate and integrate with hospital information systems. The infuser is available as either a single-channel pump or a dual-channel pump. The two configurations can be linked together to form a 3- or 4-channel pump. The infuser includes a large touchscreen color display and can be powered by either A/C power or rechargeable batteries. More detail on its use of the ICM is in the backup charts and in Chapter 5 of [Pew and Mavor, 2007].



Young Memo: Prototyping and Competition

- **Discover issues before costly SDD phase**
 - Producing detailed designs in SDD
 - Not solving myriad technical issues
- **Services and Agencies to produce competitive prototypes through Milestone B**
 - Reduce technical risk, validate designs and cost estimates, evaluate manufacturing processes, refine requirements
- **Will reduce time to fielding**
 - And enhance govt.-industry teambuilding, SysE skills, attractiveness to next generation of technologists
- **Applies to all programs requiring USD(AT&L) approval**
 - Should be extended to appropriate programs below ACAT I

03/19/2008

©USC-CSSE

26

Core of Young Memo: USD(AT&L), 19 September 2007

Lessons of the past, and the recommendations of multiple reviews, including the Packard Commission Report, emphasize the need for, and benefits of, quality prototyping. The Department needs to discover issues before the costly System Design and Development (SDD) phase. During SDD, large teams should be producing detailed manufacturing designs – not solving myriad technical issues. Government and industry teams must work together to demonstrate the key knowledge elements that can inform future development and budget decisions.

To implement this approach, Military Departments and Defense Agencies will formulate all pending and future programs with acquisition strategies and funding that provide for two or more competing teams producing prototypes through Milestone (MS) B. Competing teams producing prototypes of key system elements will reduce technical risk, validate designs, validate cost estimates, evaluate manufacturing processes, and refine requirements.

Based on these considerations, all acquisition strategies requiring USD(AT&L) approval must be formulated to include competitive, technically mature prototyping through MS B. The Component Acquisitions Executives will review all existing programs and all programs in the initial stages of development for the potential to adopt this acquisition strategy. It is the policy of the Department of Defense that this acquisition strategy should be extended to all appropriate programs below ACAT I.



Implementing the Young Memo

- **Need way of assuring continuity of funding, but with off-ramps**
 - Prototyping and Competition Plan
 - Incremental off-ramp milestones
- **Need criteria for assessing feasibility to proceed**
 - Avoid overfocus on prototyping
 - Feasible, scalable technology and management infrastructure
 - Adequate budget, schedule, critical skills
 - Key stakeholders committed to proceed
 - Shortfalls in evidence are risks, need risk management plans
- **ICM provides desired processes and criteria**
 - Anchor Point milestones and Feasibility Evidence deliverable
 - Incremental phase entry and exit criteria

03/19/2008

©USC-CSSE

27

At a recent Government/industry/academia workshop at USC, a working group discussed the acquisition implications of the Young memo. Industry personnel indicated that the traditional DoD staged competition approach had too many delays between stages to keep top technical teams together, and too many uncertainties about later-stage funding to justify commitment of scarce personnel and resources. Government personnel indicated the need to have assured multi-stage funding at the program manager level, but also to have incremental off-ramps if it became clear that the technology and knowledge base was too immature. An attractive concept to address these issues is the use of a Prototyping and Competition Plan, that identifies the competition strategy, required budgets and schedules, and incremental off-ramps if none of the competitors can produce feasible and scalable solutions.

Some key criteria for assessing evidence of implementation feasibility and scalability are evidence of technical capability to execute missions under realistic mission conditions, including ranges of off-nominal conditions, adversary countermeasures, representative users, and environmental parameters. These should include the ability to provide satisfactory tradeoffs among such key performance parameters as safety, security, performance, usability, and interoperability.

Although the title of the Young memo is “Prototyping and Competition,” it is important to avoid overfocus on prototyping. Evidence from program assessments indicates that the words in the Young memo emphasizing the need to “validate cost estimates, evaluate manufacturing processes, and refine requirements” in preparing the planning, staffing, and management foundations for system development are at least as important to program success as technology maturity.

As discussed next, the ICM provides a process framework for implementing a Prototyping and Competition Plan incorporating the desired characteristics. It includes anchor point milestones to serve as phase gates and off-ramps; a Feasibility Evidence deliverable providing evidence of technical and management feasibility and scalability; and criteria for specializing the ICM to common acquisition situations.



Outline

- **Motivation and context**
 - Emerging challenges for future space and ground systems
 - Software key to rapid operational responsiveness
 - Difficulties in integrating systems and software engineering
- **State of systems and software engineering (S&SE) integration**
 - Current SysE guidance: outdated assumptions; inhibitors to software best practices
- **Incremental Commitment Model (ICM) overview**
 - ICM nature, origin, and principles
 - ICM process views and applicability to Young memo
- ➔ • **Conclusions: IS&SE with ICM**
 - References; Acronyms; Backup charts

03/19/2008

©USC-CSSE

28

The context of this presentation includes a study performed in 2007 by USC-CSSE for the U.S. Department of Defense (DoD) on integrating systems and software engineering. The study evaluated the strengths and shortfalls of current DoD systems engineering guidance in integrating systems and software engineering for both current and likely future DoD systems. It also evaluated the Incremental Commitment Model (ICM) for integrating systems, software, and human factors engineering recommended in a recent DoD-sponsored National Research Council study [Pew and Mavor, 2007].

As recommendations, it presented a mixed strategy of incremental improvements to current DoD acquisitions, pilot projects to explore new approaches such as the ICM, and incremental improvement of DoD guidance documents and education and training initiatives. One element of the mixed strategy underway is an effort led by USC-CSSE to develop a guidebook for the use of the ICM to integrate systems and software engineering on DoD projects, in concert with several current DoD projects piloting all or parts of the ICM.

Concurrently, USC-CSSE will be developing commercial and educational versions of the ICM, developing with IBM an OpenUP electronic process guide for the ICM, and testing the educational version in Fall 2008 in its annual series of 20 real-client, 6-8 person, MS-level student team, e-services projects.



Conclusions: IS&SE with ICM

- **Current DoD SysE guidance seriously inhibits SwE best practices**
 - Largely sequential definition, design, development, integration, and test
 - Slows agility, ability to turn inside adversaries' OODA loop
 - Functional “part-of” vs. layered “served by” product hierarchy
 - Static vs. dynamic interfaces: messages vs. protocols
 - One-size-fits-all process guidance inhibits balancing agility and assurance
- **Incremental Commitment Model (ICM) enables better IS&SE**
 - Integrates hardware, software, human factors aspects of SysE
 - Concurrent exploration of needs, opportunities, solutions
 - Successfully addresses issues above in commercial practice
 - Only partially proven in DoD practice (examples: CrossTalk Top-5 projects)
 - Key practices applied to help major programs (example: Future Combat Systems)
 - Being adopted by major programs (example: Missile Defense Agency)
 - Effort underway to work with adopters to develop DoD ICM Guidebook, in coordination with other USD(AT&L)SSE initiatives

03/19/2008

©USC-CSSE

29

As further elaborated in the backup charts, when systems engineering was initially defining itself, maturing, and setting standards in the 1960s and 1970s, its focus was on hardware artifacts. People were not part of “the system” but users of it. In the 1980s, some pioneers began to experience that systems were getting more software-intensive and human-intensive, and formulated initial approaches such as Soft Systems Engineering [Checkland, 1980] and Systems Architecting [Rechtin, 1991] to integrate these factors into systems engineering.

Due to the fact that general changes in standards and guidelines take a good deal of time, the DoD systems engineering standards area is still in the process of assimilating these perspectives. The results of the 2007 IS&SE study and associated results such as the [Maier, 2006] system and software architectures paper and the NDIA Software Summit identification of critical software issues indicate that software engineering best practices are being seriously inhibited by current DoD systems engineering guidance, and that improvement avenues are available in such areas as acquisition, architecture, and processes.

The ICM provides an avenue for adapting current best commercial IS&SE practices for use on DoD projects. USC-CSSE’s current effort to work with pilot adopters to develop a DoD ICM Guidebook, in coordination with other USD(AT&L)SSE initiatives, affords opportunities for further candidate pilot adopters to collaborate in this effort.



2008 SOW: DoD ICM Guidebook

- **Develop a draft Guidebook for next-generation DoD IS&SE based on the ICM**
 - In collaboration with DoD and industry participants
- **Collaborate with selected DoD and industry parties in experimental tailoring of draft Guidebook elements**
- **Hold a series of Guidebook workshops**
 - Mar 19-20 (USC), July 15-17 (DC area), Oct 29-30 (USC)
- **Coordinate Guidebook with other IS&SE initiatives**
 - **Systems of systems engineering, systemic analysis, acquisition, education, assessment, research and technology**

03/19/2008

©USC-CSSE

30

Task 1. In collaboration with DoD and industry participants, develop a draft Guidebook for next-generation DoD integrated systems and software engineering (IS&SE) based on the Incremental Commitment Model (ICM). The guidebook will include:

- General principles, practices, and DoD milestone and SE technical review criteria common to all projects using the ICM;
- Detailed guidelines for using the ICM key practices to integrate systems and software engineering, and to prepare for DoD milestone and SE technical reviews during each DoD acquisition phase;
- Process decision criteria enabling early determination of project-appropriate, risk-driven special cases of the ICM;
- Guidelines for ICM-compatible interpretations of current DoD regulations, specifications, and standards;
- Examples of good and bad interpretations of ICM principles and practices;
- Guidelines for enhancing current projects with selected ICM principles and practices.

Task 2. Collaborate with selected DoD and industry projects in experimental tailoring of early draft Guidebook elements for application to existing, new, and experimental pilot projects. Candidate projects include the Missile Defense Agency, the Army Future Combat Systems program, a TBD Air Force space program, and a TBD Navy program.

Task 3. Hold a series of government-industry workshops to obtain feedback on evolving drafts of Guidebook material. Candidate times and places would be in March at USC; in July on the east coast, and in October at USC.

Task 4. Coordinate development of the draft Guidebook with other OSD-AT&L IS&SE initiatives (e.g., acquisition, education, assessment, systemic analysis, systems of systems engineering, research and technology agendas).

Task 5. In particular, coordinate with the SSE software systemic analysis initiative.

Task 6. In particular, coordinate with the SSE systems of systems engineering initiative.



References - I

- Beck, K., *Extreme Programming Explained*, Addison Wesley, 1999.
- Boehm, B., "Some Future Trends and Implications for Systems and Software Engineering Processes", *Systems Engineering* 9(1), pp. 1-19, 2006.
- Boehm, B., Brown, W., Basili, V., and Turner, R., "Spiral Acquisition of Software-Intensive Systems of Systems, *CrossTalk*, Vol. 17, No. 5, pp. 4-9, 2004.
- Boehm, B. and Lane J., "21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems." *CrossTalk*: Vol. 19, No. 5, pp.4-9, 2006.
- Boehm, B., and Lane, J., "Using the ICM to Integrate System Acquisition, Systems Engineering, and Software Engineering," *CrossTalk*, October 2007, pp. 4-9.
- Boehm, B., and Lane, J., "A Process Decision Table for Integrated Systems and Software Engineering," *Proceedings, CSER 2008*, April 2008.
- Boehm, B., "Future Challenges and Rewards for Software Engineers," *DoD Software Tech News*, October 2007, pp. 6-12.
- Boehm, B. et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
- Boehm, B., *Software Engineering Economics*, Prentice Hall, 2000.
- Carlock, P. and Fenton, R., "System of Systems (SoS) Enterprise Systems for Information-Intensive Organizations," *Systems Engineering*, Vol. 4, No. 4, pp. 242-26, 2001.
- Carlock, P., and J. Lane, "System of Systems Enterprise Systems Engineering, the Enterprise Architecture Management Framework, and System of Systems Cost Estimation", *21st International Forum on COCOMO and Systems/Software Cost Modeling*, 2006.
- Checkland, P., *Systems Thinking, Systems Practice*, Wiley, 1980 (2nd ed., 1999).
- Department of Defense (DoD), *Defense Acquisition Guidebook, version 1.6*, <http://akss.dau.mil/dag/>, 2006.
- Department of Defense (DoD), *Instruction 5000.2, Operation of the Defense Acquisition System*, May 2003.
- Department of Defense (DoD), *Systems Engineering Plan Preparation Guide, USD(AT&L)*, 2004.
- Electronic Industries Alliance (1999); EIA Standard 632: Processes for Engineering a System
- Galorath, D., and Evans, M., *Software Sizing, Estimation, and Risk Management*, Auerbach, 2006.
- Hall, E.T., *Beyond Culture*, Anchor Books/Doubleday, 1976.



References -II

- Highsmith, J., *Adaptive Software Development*, Dorset House, 2000.
- International Standards Organization, *Information Technology Software Life Cycle Processes*, ISO/IEC 12207, 1995
- ISO, *Systems Engineering – System Life Cycle Processes*, ISO/IEC 15288, 2002.
- Jensen, R. "An Improved Macrolevel Software Development Resource Estimation Model," *Proceedings, ISPA 5*, April 1983, pp. 88-92.
- Krygiel, A., Behind the Wizard's Curtain; CCRP Publication Series, July, 1999, p. 33
- Lane, J. and Boehm, B., "System of Systems Cost Estimation: Analysis of Lead System Integrator Engineering Activities", *Information Resources Management Journal*, Vol. 20, No. 2, pp. 23-32, 2007.
- Lane, J. and Valerdi, R., "Synthesizing SoS Concepts for Use in Cost Estimation", *Proceedings of IEEE Systems, Man, and Cybernetics Conference*, 2005.
- Lientz, B., and Swanson, E.B., *Software Maintenance Management*, Addison Wesley, 1980.
- Madachy, R., Boehm, B., Lane, J., "Assessing Hybrid Incremental Processes for SISOS Development", USC CSSE Technical Report USC-CSSE-2006-623, 2006.
- Maier, M., "Architecting Principles for Systems-of-Systems"; *Systems Engineering*, Vol. 1, No. 4 (pp 267-284).
- Maier, M., "System and Software Architecture Reconciliation," *Systems Engineering* 9 (2), 2006, pp. 146-159.
- Northrop, L., et al., *Ultra-Large-Scale Systems: The Software Challenge of the Future*, Software Engineering Institute, 2006.
- Pew, R. W., and Mavor, A. S., *Human-System Integration in the System Development Process: A New Look*, National Academy Press, 2007.
- Putnam, L., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Trans SW Engr.*, July 1978, pp. 345-361.
- Rechtin, E. *Systems Architecting*, Prentice Hall, 1991.
- Schroeder, T., "Integrating Systems and Software Engineering: Observations in Practice," *OSD/USC Integrating Systems and Software Engineering Workshop*, <http://csse.usc.edu/events/2007/CIIForum/pages/program.html>, October 2007.



List of Acronyms

B/L	Baselined
C4ISR	Command, Control, Computing, Communications, Intelligence, Surveillance, Reconnaissance
CD	Concept Development
CDR	Critical Design Review
COTS	Commercial Off-the-Shelf
DCR	Development Commitment Review
DI	Development Increment
DoD	Department of Defense
ECR	Exploration Commitment Review
EVMS	Earned Value Management System
FCR	Foundations Commitment Review
FDN	Foundations, as in FDN Package
FED	Feasibility Evidence Description
FMEA	Failure Modes and Effects Analysis
FRP	Full-Rate Production
GAO	Government Accountability Office
GUI	Graphical User Interface



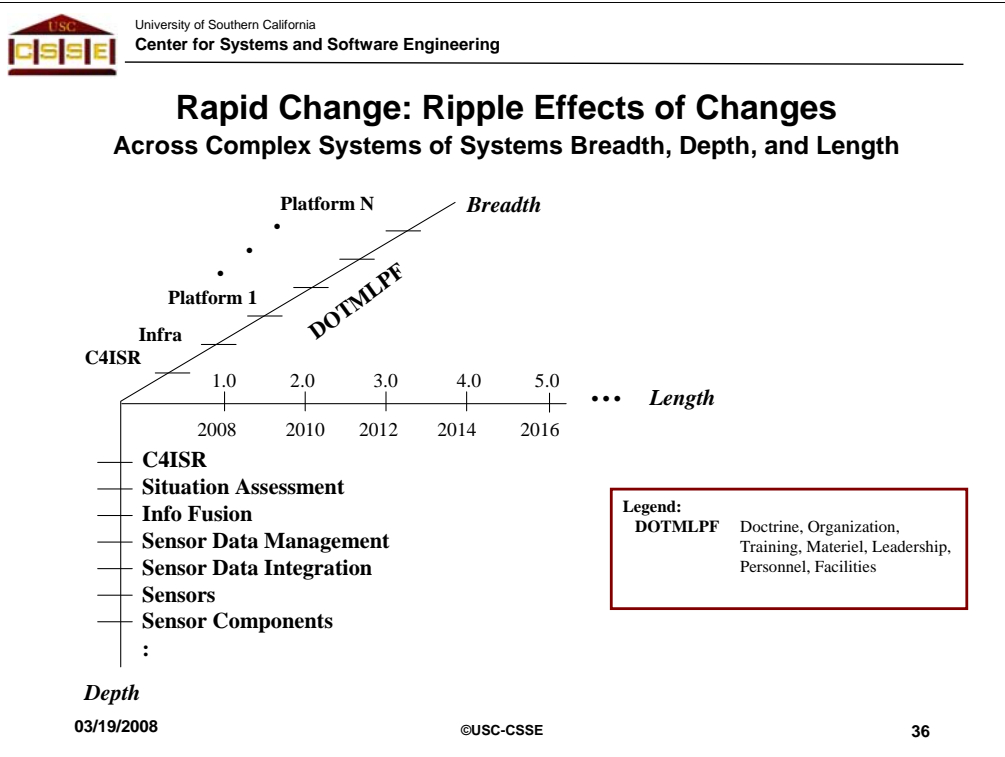
List of Acronyms *(continued)*

HMI	Human-Machine Interface
HSI	Human-System Interface
HW	Hardware
ICM	Incremental Commitment Model
IOC	Initial Operational Capability
IRR	Inception Readiness Review
IS&SE	Integrating Systems and Software Engineering
LCA	Life Cycle Architecture
LCO	Life Cycle Objectives
LRIP	Low-Rate Initial Production
MBASE	Model-Based Architecting and Software Engineering
NDI	Non-Developmental Item
NRC	National Research Council
OC	Operational Capability
OCR	Operations Commitment Review
OO&D	Observe, Orient and Decide
OODA	Observe, Orient, Decide, Act
O&M	Operations and Maintenance



List of Acronyms *(continued)*

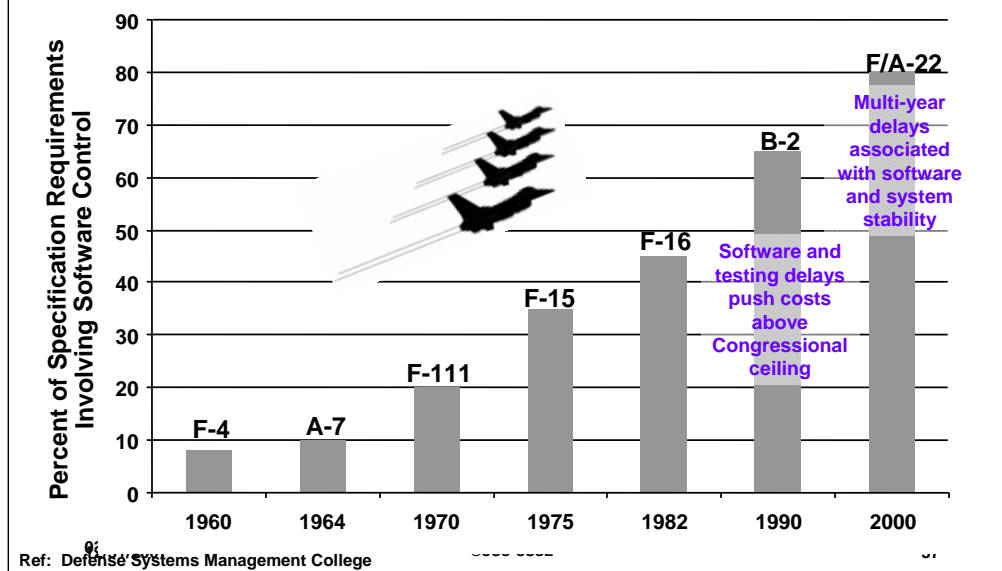
PDR	Preliminary Design Review
PM	Program Manager
PR	Public Relations
PRR	Product Release Review
RUP	Rational Unified Process
SoS	System of Systems
SoSE	System of Systems Engineering
SSE	Systems and Software Engineering
SW	Software
SwE	Software Engineering
SysE	Systems Engineering
Sys Engr	Systems Engineer
S&SE	Systems and Software Engineering
USD (AT&L)	Under Secretary of Defense for Acquisition, Technology, and Logistics
VCR	Validation Commitment Review
V&V	Verification and Validation
WBS	Work Breakdown Structure
WMI	Warfighter-Machine Interface



DoD's C4ISR successes have come by transitioning from a set of stovepiped forces with incompatible interfaces into a more network-centric set of interoperable forces able to turn more rapid and accurate OODA loops. This has been done largely by providing tighter coupling among the force components. However, as software and high-performance-computing people have found via bitter experience, tight coupling buys faster operational performance at the cost of slower adaptability to change. Each change creates extensive ripple effects requiring coordinated changing across the overall complex of components, along with second-order effects on the system of system's doctrine, operations, training, materiel, leadership, personnel, and facilities (DOTMLPF). Not only does this happen across a wide breadth of systems and platforms, but each system such as C4ISR has a deep supplier chain of subcontractors and sub-subcontractors sustaining the system of system's situation awareness, information fusion, sensor data management, sensor data integration, individual sensors, and sensor components. Beyond this, a change may require some capabilities to be deferred to later increments across the length of the system's life cycle, causing further dependency ripple effects. The magnitude of this challenge of change management across the breadth, depth, and length of complex systems of systems using traditional systems architectures and contracting mechanisms was seen in chart 5.



Systems Engineering Is Evolving from its Hardware Origins



As illustrated by this chart, when systems engineering was initially defining itself, maturing, and setting standards in the 1960s and 1970s, its focus was on hardware artifacts such as the aircraft above. People were not part of “the system” but users of it. In the 1980s, some pioneers began to experience that systems were getting more software-intensive and human-intensive, and formulated initial approaches such as Soft Systems Engineering [Checkland, 1980] and Systems Architecting [Rechtin, 1991] to integrate these factors into systems engineering.

Due to the fact that general changes in standards and guidelines take a good deal of time, the systems engineering field is still in the process of assimilating these perspectives, as will be illustrated in the review below of current DoD systems engineering guidance. If the techniques of hardware, software, and human factors engineering were highly similar, this would not be a problem. However, as shown in the next few charts, there are significant differences among these three disciplines that can cause serious difficulties in using hardware-intensive guidelines to engineer software-intensive or human-intensive systems.



Underlying HwE, SwE, HFE Differences

Difference Area	Hardware	Software	Human Factors
Major Life-cycle Cost Source	Development, manufacturing	Life-cycle evolution	Training and operations labor
Ease of Changes	Generally difficult	Good within architectural framework	Very good, but people-dependent
Nature of Changes	Manual, labor-intensive, expensive	Electronic, inexpensive	Need personnel retraining, can be expensive
User-tailorability	Generally difficult, limited options	Technically easy; mission-driven	Technically easy; mission-driven
Indivisibility	Inflexible lower limit	Flexible lower limit	Smaller increments easier to introduce
Underlying Science	Physics, chemistry, continuous mathematics	Discrete mathematics, linguistics	Behavioral sciences
Testing	By test organization; much analytic continuity	By test organization; little analytic continuity	By users

03/19/2008

©USC-CSSE

38

The major sources of life cycle cost in a hardware-intensive system are during development and manufacturing, particularly for systems having large production runs. For software-intensive systems, manufacturing costs are essentially zero, and typically about 70% of the life cycle cost goes into post-development maintenance and upgrades [Lientz-Swanson, 1980; Boehm, 1981]. For human-intensive systems, the major costs are staffing and training, particularly for defense systems requiring continuous 24/7 operators. A primary reason for this difference is indicated in rows 2 and 3 of the table. Particularly for widely-dispersed hardware such as ships, submarines, satellites, and some ground vehicles, making hardware changes across a fleet can be extremely difficult and expensive. As a result, many hardware deficiencies are handled via software or human workarounds that save money overall but shift the life-cycle costs toward the software and human parts of the system.

As can be seen when buying hardware such as cars or TVs, there is some choice of options, but they are generally limited. It is much easier to tailor software or human procedures to different classes of people or purposes. It is also much easier to deliver useful subsets of most software and human systems, while delivering a car without braking or steering capabilities is infeasible.

The science underlying most of hardware engineering involves physics, chemistry, and continuous mathematics. This often leads to implicit assumptions about continuity, repeatability, and conservation of properties (mass, energy, momentum) that may be true for hardware but not true for software or human counterparts. An example is in testing. A hardware test engineer can generally count on covering a parameter space by sampling, under the assumption that the responses will be a continuous function of the input parameters. A software test engineer will have many discrete inputs, for which a successful test run provides no assurance that the neighboring test run will succeed. And for humans, the testing needs to be done by the operators and not test engineers.



Implications for Integrating SysE and SwE: Current SysE Guidelines Emphasize Hardware Issues

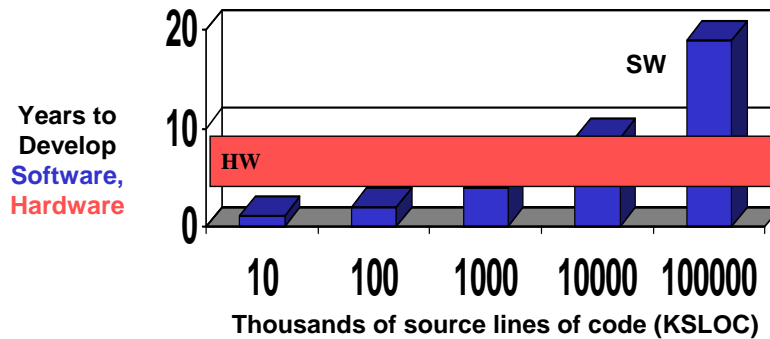
- **Focus on early hardware decisions may lead to**
 - **Selecting hardware components with incompatible software**
 - **Inadequate hardware support for software functionality**
 - **Inadequate human operator capability**
 - **Late start of software development**
- **Difficulty of hardware changes may lead to**
 - **High rate of change traffic assigned to software without addressing critical-path risks**
- **Indivisibility may lead to single-increment system acquisition**
- **Different test phenomena may lead to inadequate budget and schedule for testing software and human factors**

Focusing early systems engineering decisions on hardware cost-effectiveness can lead to serious problems with the software and human aspects of the system. Frequently, source selection of best-of-breed hardware suppliers will lead to a system with incompatible software commercial-off-the-shelf (COTS) products and incompatible user interfaces that are difficult if not impossible to reconcile. A wireless system with limited size, weight, and power constraints may skimp on computer capacity (inflating software costs) or display capabilities (making the system effectively unusable). Deferring the start of software development until all of its requirements are ready will often put the software on the system delivery critical path, thereby causing critical path slippages when using software workarounds to cover hardware shortfalls.

Acquiring indivisible hardware systems often leads to acquiring all of the software at once also. This increases critical-path risks and loses the opportunity to obtain user feedback on early software increments. It also delays the start of infrastructure and test software, often delaying the start of integration and test. And assuming analytic continuity when test planning and budgeting will generally leave insufficient resources for software and operational testing.

Software Development Schedule Trends

#Years $\sim 0.4 * \text{cube root (KSLOC)}$



Delaying software start increasingly risky
Need to find ways to compress software schedules
- Timeboxing; architecting for decoupled parallel development

03/19/2008

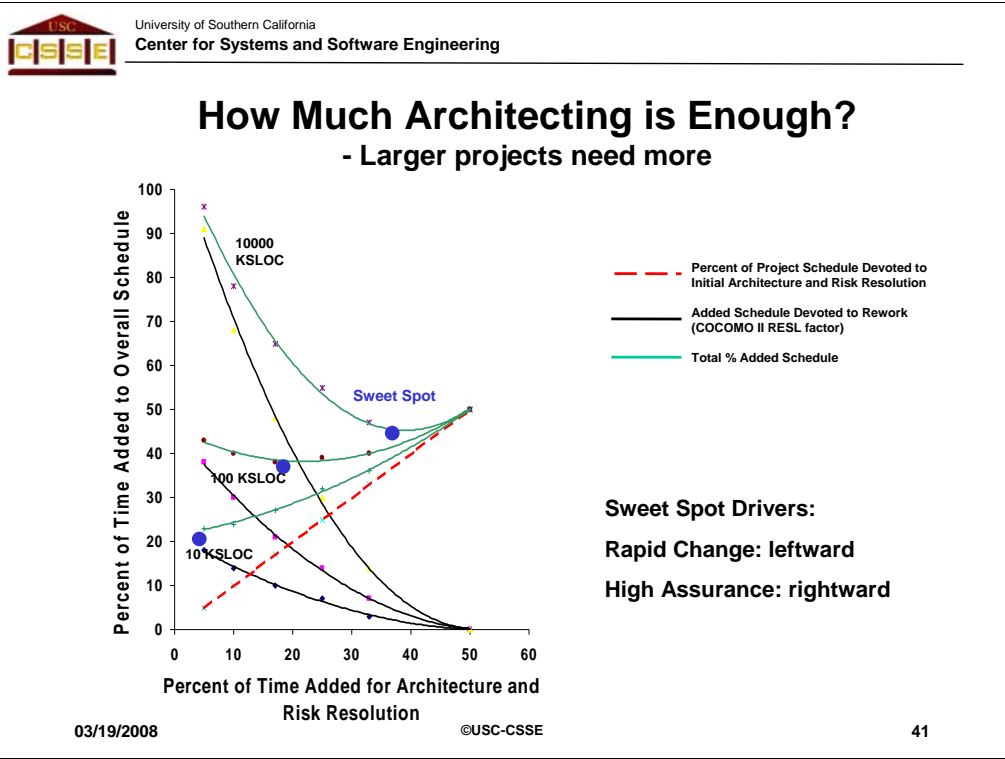
©USC-CSSE

40

A persistent relationship between software size and development schedule found in most software estimation models [Putnam, 1978; Boehm, 1981; Jensen, 1983, Galorath-Evans, 2006] is that the number of years a software project will take to complete is proportional to roughly 0.4 times the cube root of the project's size in thousands of equivalent (reuse-adjusted) source lines of code (KSLOC). For 1000 KSLOC or a million lines of source code, the cube root of 1000 is 10, and the development time is likely to be around $0.4 * 10 = 4$ years, or at the lower end of a 4-8 year range of typical hardware system development times. However, around 10 million lines of code, the software development time begins to exceed typical hardware development times, and many envisioned complex systems of systems have much more software than this to develop.

For one thing, this means that delaying the start of software development on such projects is extremely risky, and it is better to accept some risk in starting parts of the software, particularly the infrastructure and integration support software, before the total system is fully defined (often, starting the software will find system problems such as those described in the previous charts and save much downstream rework).

Another risk mitigation strategy is timeboxing: if schedule is indeed the independent variable, it is best to prioritize features and architect the system to be able to add or drop borderline-priority features to meet the schedule. A further mitigation strategy is to architect the system to enable extensive parallel development: if the architecture enables a set of million-line components developed in 4 years to plug and play, a great deal of integration and rework time is saved. But as discussed on the next chart, this requires an increasingly large investment in up-front architecting time.



Suppose that the mitigation strategy discussed on the previous chart is applied to a 10 million line software system normally requiring 9 years to develop. If the system is architected to enable 10 1-million-line components to be concurrently developed in 4 years to plug and play, a great deal of integration and rework time is saved.

The figure above shows how this tradeoff between architecting time and rework time can be analyzed by the well-calibrated COCOMO II Architecture and Risk Resolution Factor [Boehm et al., 2000]. It shows that for a 10,000-KSLOC system, the “sweet spot” minimizing the sum of architecting time and rework time occurs at about 37% of the development time for architecting, with a relatively flat region between 30% and 50%. Below 30%, the penalty curve for premature issuance of supplier specifications is steep: a 17% investment in architecting yields a rework penalty of 48% for a total delay of 65% compared with a rework penalty of 20% and a total delay of 50% for the 30% architecting investment.

For the 10,000-KSLOC system above, investing 33% of a 9-year development schedule for architecting would use 3 years. If the 10 components developed in 4 years were then able to plug and play, the system would be delivered in 7 years vs. 9. Note on the chart that rapid change would require some effort and delay in revising the architecture, but this could add up to 2 years and the project would still be ahead.

This figure and its implications were convincing enough to help one recent very large software-intensive system to add 18 months to its schedule to improve the architectural specifications before committing to development.



Relationship Between *Systems Engineering* and *Software Engineering* in a SoS Environment

- **Focus of System of Systems (SoS) SysEs: Primarily attempting to identify set of options for incorporating functions to support desired new capabilities**
- **Core element activities do not tend to segregate hardware engineering, software engineering, and human factors engineering**
 - SysEs take more of a holistic point of view in analyses and trade-offs
 - SysEs looking for options and opportunities within the desired timeframe
- **Success in integration of systems and software engineering heavily influenced by the fact that SoS development seldom starts with a “clean sheet of paper”**
- **Current challenge: What can we learn from this for new system developments starting with a fairly clean sheet of paper?**

03/19/2008

©USC-CSSE

42

Relationship Between Systems Engineering and Software Engineering in a System of Systems (SoS) Environment

Key to making this relationship work for SoSE are:

Addressing organizational as well as technical perspectives

Focusing on areas critical to the SoS:

- Overall capability of SoS
- Continuous (“up front”) analysis which anticipates change based on a robust understanding of internal and external sources of change
- Software and system architectures that are open and loosely coupled, extensible, flexible, and persistent over time
- Design strategy that relies on trades performed upfront and throughout

A technical management approach that reflects need for transparency and trust with focused active participation



Comparison of Top-10 Risks

Software-Intensive Systems of Systems - CrossTalk, May 2004

1. Acquisition management and staffing
2. Requirements/architecture feasibility
3. Achievable software schedules
4. Supplier integration
5. Adaptation to rapid change
6. Quality factor achievability and tradeoffs
7. Product integration and electronic upgrade
8. Software COTS and reuse feasibility
9. External interoperability
10. Technology readiness

Software-Intensive Systems - CSSE 2006-07 Top 10 Survey

1. Architecture complexity, quality tradeoffs
2. Requirements volatility
3. Acquisition and contracting process mismatches
4. Budget and schedule
5. Customer-developer-user
6. Requirements mismatch
7. Personnel shortfalls
8. COTS
9. Technology maturity
10. Migration complexity

Many similarities...

Results from an informal survey conducted as part of the IS&SE workshop indicate that the above risks are still the top 10 risks of concern...

03/19/2008

©USC-CSSE

43

Comparison of Top-10 Risks

The top 4 risks in three different surveys include:

- Acquisition and contracting issues
- Architecture issues
- Budget and schedule issues
- Requirements volatility/feasibility

These same risks are also reflected in the COCOMO software, systems engineering, and system of systems cost models and are considered to be significant cost drivers as well as influences on the overall success of the development effort.



SoSE Lessons Learned to Date

- **SoSs provide examples of how systems and software engineering can be better integrated when evolving existing systems to meet new needs**
- **Net-centricity and collaboration-intensiveness of SoSs have created more emphasis on integrating hardware, software, and human factors engineering**
- **Focus is on**
 - Flexibility
 - Adaptability
 - Use of creative approaches, experimentation, and tradeoffs
 - Consideration of non-optimal approaches that are satisfactory to key stakeholders
- **SoS process adaptations have much in common with the Incremental Commitment Model (ICM)**

03/19/2008

©USC-CSSE

44

Systems of Systems Engineering Lessons Learned to Date

This slide summarizes some of the key findings from both the OSD SoSE pilot surveys and the USC CSSE SoSE cost model research. There are many instances where SoSE teams have adapted and streamlined traditional systems and software engineering processes to better integrate them and view both areas when making both strategic and tactical SoS architecture and evolution decisions. The need to be flexible, adaptable, and creative to quickly meet changing needs has driven much of this adaptation and streamlining. It is also interesting to observe that many of the adaptations that have been made to these processes are consistent with the ICM principles to be discussed in chart 36 and incorporate the ICM agile, plan-driven, and V&V team focus to be discussed in charts 47 and 48.



DoD S&SE Guidance Strengths and Shortfalls

- **Systems Engineering Plan Preparation Guide**
- **Defense Acquisition Guide Chapter 4 (SysE)**
- **DoDI 5000.2**

Overall, the DoD Systems Engineering Plan Preparation Guidelines, Chapter 4 (Systems Engineering) of the Defense Acquisition Guidebook, and DoD Instruction 5000.2 are significant advances over their predecessors, and most of their shortfalls can be overcome when being applied by experts.

Unfortunately, many programs have shortages of experts, particularly for software, and literal application of some of the guidance shortfalls can lead to trouble in integrating systems and software engineering. The identification of a shortfall is meant to be an indicator of an issue to consider carefully during systems engineering of a program, and an item to consider in future versions of the guidelines, and not as a reason to avoid use of the guidelines.



Review of SysE Plan (SEP) Guidelines: Advances Over Previous Approaches

- Tailoring to milestones A, B, C (*all*)
- Better linkages to acquisition, program management (*A2.5, 3, 6*)
- More explicit focus on risk management (*A 4.3, 6.3*)
- More explicit focus on Tech. Readiness Levels (*A2.3, 4.3*)
- More explicit addressal of full range of stakeholders (*A 3.2-5, 5.4*)
- Addressal of Family/Systems of Systems considerations (*A1.1, 3.5, B, C*)
- Focus on critical technologies, fallback strategies (*A2.3*)
- Focus on event-based vs. schedule-based milestones (*A5.1*)
- Identification of IPT critical success factors (*A3.2, 3.3, 3.4*)

03/19/2008

©USC-CSSE

46

For convenience to DoD System Engineering Plan (SEP) developers, there are individual guidelines for each of the SEPs to be reviewed at each major DoD acquisition milestone and covering its subsequent phase: Milestone A and Technology Development; Milestone B and System Development and Demonstration; and Milestone C and Production and Deployment/Operations and Support. The guidelines for each milestone and phase are highly overlapping, so this review will focus on the guidelines for Milestone A. The comments above are referenced to the relevant paragraph in Section A, which usually has a counterpart paragraph in the Milestone B and C guidelines. Most of the comments are fairly self-evident, so these notes will concentrate on items 3, 6, and 8 in blue.

Early waterfall-model approaches to systems engineering focused on achieving such properties as completeness and correctness, which often focused projects on trivial issues while high-risk issues remained unaddressed. The current SEP Guidelines appropriately emphasize risk management integration in the Milestone A, B, and C guidelines.

Previous SEP guidance was primarily formulated before there were many families of systems or systems of systems being developed. The current SEP Guidelines explicitly include these, and emphasize integration with external organizations in the Milestone A, B, and C guidelines.

Previous project management guidelines emphasized keeping on a predetermined schedule and often resulted in premature reviews. The definition and achievement of entry and exit criteria for each review are emphasized in the Milestone A, B, and C guidelines.



SEP Guidelines: Risky Assumptions I

Sometimes OK for hardware; generally not for software

- **An omniscient authority pre-establishes the requirements, preferred system concept, etc. (A4.1, 4.2, 4.4)**
- **Technical solutions are mapped and verified with respect to these (A4.1, 4.2, 4.4)**
- **They and technology do not change very often (A4.2, 4.5, 6.2)**
 - Emphasis on rigor vs. adaptability
- **The program has stable and controllable external interfaces (A4.5)**
- **MRL's and TRL's exist independent of program scale and complexity (A2.3, 4.3, 4.5)**
- **Systems do not include humans (Pref., A4.4, 3.2)**
 - Emphasis on material requirements, solutions (A3.1, 3.2, 6.4, 6.5)
- **Confusion in satisfying users vs. all stakeholders (A2.1, 3.4)**

03/19/2008

©USC-CSSE

47

The SEP guidelines make a number of implicit assumptions which are increasingly invalid as requirements become more emergent than prespecifiable, as requirements and technology undergo more rapid change, and as capabilities and key performance parameters become at least as success-critical as functions and products. Again and below, the notes will focus on the less self-evident points; here on points 1, 5, and 6.

Even before Milestone A, Section 4 assumes that requirements are in place as a basis for traceability, verification, and allocation, even before key technologies and COTS products are fully evaluated. This tends to encourage premature specification of requirements, some of which will be found later to be infeasible or inappropriate.

Readiness levels for technology (TRLs) and manufacturing (MRLs) are referred to independent of the scale and environment of the system. A technology can be at Level 7 for a modest-size system and Level 2 for a very large system.

Systems are defined in terms of products and functions, do not include people, and minimally address operational requirements and critical mission scenarios.



SEP Guidelines: Risky Assumptions II

Sometimes OK for hardware; generally not for software

- **Project organization is function-hierarchical and WBS-oriented (A3.1, 3.2)**
- **Contractual, MOA arrangements are separable from technical issues (A3.5, 6.3)**
- **All requirements are equally important (A4.2)**
- **Reviews event/product-based vs. evidence-based (A5.2)**
- **Producibility is important for manufacturing but not software (A6.1, Δ in 6.4)**
- **The program's critical path can be accurately identified up front (A6.1)**
- **Program managers only make decisions at review points (A6.2)**
- **One can achieve optimum readiness at minimum life-cycle cost (A6.5)**
 - **No slack for contingencies**
- **Most importantly, overfocus on technology maturity (A4)**
 - **Many more sources of risk in OSD/AT&L root cause analysis**

03/19/2008

©USC-CSSE

48

Here, the notes address points 1, 4, 8, and 9.

Project organization and Integrated Product Teams (IPTs) are assumed to follow functional and Work Breakdown Structure (WBS) hierarchies. As discussed in charts 8 and 9, functional hierarchies are incompatible with layered software structures and inhibit integration of systems and software engineering. IPTs are also needed for non-functional cross-cutting issues such as safety, security, and end-to-end performance.

Event- and product-based reviews are better than schedule-based reviews, but they frequently overfocus on presenting functional diagrams without providing any evidence that a product built to satisfy the functions would have adequate performance over a range of stressing scenarios. Evidence of feasibility should also be produced, evaluated, and reviewed for adequacy.

In a world of uncertainty and complex tradeoffs among multiple objectives, stakeholders, and performance criteria, words such as optimize, maximize, and minimize are misleading. For example, a minimum life-cycle cost system will have no slack and no way to be optimally ready for contingencies.

Technology maturity is an important aspect of a proposed system, but it is only one of many aspects that must be systems engineered to achieve program success. Chart 50 will elaborate on this.



DAG Chapter 4: SysE Evaluation: Summary

- **Defense Acquisition Guide Chapter 4 Strengths**
 - Major improvement over previous sequential, reductionist approach
 - Addressal of systems of systems
 - Good emphasis on early V&V, trade spaces, thorough up-front effort
- **Shortfalls**
 - Still some sequential, reductionist, hardware holdovers: complete requirements, top-down decomposition, RAM
 - Reluctance to reconcile related regulations, specifications, and standards
 - Inheritance of conflicts (people internal/external to system)
 - Minimal addressal of change analysis, external systems evolution, SysE support of acquisition management

03/19/2008

©USC-CSSE

49

As with the SEP Preparation Guide, Chapter 4 (Systems Engineering) of the Defense Acquisition Guidebook is a major improvement over its predecessors, but has some shortfalls with respect to integrating systems and software engineering that can be overcome when being applied by experts, but can lead to trouble when being applied by non-experts.

Its primary strengths are a much improved emphasis on concurrently engineering requirements and solutions, on addressal of systems of systems issues, and on early tradeoff analyses and verification and validation (V&V).

Its primary shortfalls involve residual holdovers from previous guidelines such as assumptions that emergent requirements can be prespecified, that systems can be defined purely top-down, and that reliability, availability, and maintainability (RAM) apply only to hardware and not to software.

It adopts incompatible definitions from different standards and directives: in Section 4.1.1, it follows EIA/IS 632 in defining a system as a product external to people; and in Section 4.1.3, it follows DoD Directive 5000.1 in defining a total system as including hardware, software, and human elements. And it could provide more extensive guidance in such increasingly important areas as change impact analysis and synchronization, systems engineering of adaptation to externally evolving interoperating systems, and systems engineering support of complex systems of systems acquisition.

Evaluation of its specific strengths and shortfalls is organized below in terms of four key issue areas: support of concurrent vs. sequential engineering; systems engineering satisficing among multiple missions, stakeholders, and objectives; incremental and evolutionary system definition and commitment; and risk-driven activities.



DAG Chapter 4: SysE Evaluation: Concurrent Engineering

- **Strengths**
 - Use of Integrated Product Teams, IPPD (4.1.5)
 - NDI evaluation during requirements development (4.2.4.1)
 - Emphasized for systems of systems (4.2.6)
- **Shortfalls**
 - Logical analysis overly top-down, sequential (4.2.4.2)
 - V-diagrams too sequential, heavyweight (4.3.1, 4.3.2)
 - Functional decomposition incompatible with NDI, service/object-oriented approaches (throughout)
 - Cost not considered in 4.3.1.3; highlighted in 4.3.1.4
 - Evolutionary acquisition SysE too sequential (4.3.6)
 - Some hardware-only sections (4.4.8, 4.4.9)

03/19/2008

©USC-CSSE

50

In the area of concurrent vs. sequential engineering, Chapter 4 is strong in its emphasis on Integrated Product and Process Development (IPPD) and its use of Integrated Product Teams (IPTs) to balance product and process solutions and meet both cost and performance objectives. It recognizes the increasing importance of Non-Developmental Items (NDIs) to system solutions and the need to concurrently and iteratively explore and define requirements and solutions.

It is good in addressing systems of systems issues, although it inconsistently recommends top-down approaches for addressing systems of systems and NDIs, which have strong bottom-up aspects. Its V-diagrams are improvements over previous more-sequential versions, although non-experts will still find it too easy to misinterpret them as sequential waterfall processes, particularly if their contracts and award fee structures invoke waterfall standards such as MIL-STD-1521B.

As discussed in charts 8 and 9, its strong emphasis on functional decomposition is incompatible with layered, service-oriented software architecture decomposition and related software NDIs. Section 4.3.1.3's guidance lays minimal emphasis on cost and affordability, although valid cost estimates and acceptable cost risk are highlighted as review criteria in Section 4.3.1.4.

Sections 4.4.8 and 4.4.9 have statements such as, "RAM (reliability, availability, maintainability) system requirements address all elements of the system, including support and training equipment, technical manuals, spare parts, and tools." But all of their specifics, such as spare parts, producibility, system manufacturing and assembly, sampling methods to estimate part reliability, and corrosion protection and mitigation are hardware-oriented, with no mention of system-critical software RAM and supportability issues such as deadlock avoidance, database recovery, and software COTS refresh synchronization.



DAG Chapter 4: SysE Evaluation: Stakeholder Satisficing

- **Strengths**
 - **Balanced, user-adaptive approach (4.1.1)**
 - **Use of Integrated Product Teams and HSI (4.1.5, 4.4.10)**
- **Shortfalls**
 - **Emphasis on “optimizing” (4.0, 4.1.3, 4.1.5)**
 - **Overfocus on users vs. other stakeholders (4.3.1, 4.3.2)**
 - **Owners, Maintainers, Interoperators, Sponsors**

Overall, Chapter 4 adopts a balanced approach that is “adaptive to the needs of the user,” and has a strong emphasis on IPTs and human-systems integration (HSI) techniques to achieve balance. However, it overemphasizes satisfying users and says little about satisficing with respect to other success-critical stakeholders such as owners of key component systems and interoperating systems, system maintainers, sponsors, and others such as NDI suppliers and testers. And as discussed on chart 44, it is inconsistent about whether or not humans are part of the system.



DAG Chapter 4: SysE Evaluation: Incremental/Evolutionary Definition and Commitment

- **Strengths**
 - Evolutionary acquisition (4.1.3, 4.1.4)
 - Emphasized for systems of systems (4.2.6)
 - Good lists of ASR, SRR content (4.3.1.4.2, 4.3.2.4.1)

- **Shortfalls**
 - IPPD emphasis on “optimizing” (4.1.5)
 - System Design makes all life cycle decisions (4.2.3.1)
 - ASR try to minimize future change (4.3.1.4.2)
 - SRR overemphasis on fully-defined requirements (4.3.2.4.1)

With respect to incremental and evolutionary system definition and commitment, DAG Chapter 4 explicitly endorses evolutionary acquisition overall, and explicitly endorses incremental development for systems of systems. The content of its Alternative System Review is quite strong, including development of a comprehensive rationale and risk assessment for the project’s preferred system solution. However, its attempt to minimize the number of requirements that may need to be changed in future phases is inconsistent with future trends toward increasing changes in threats and opportunities.

The content of its System Requirements Review has strong coverage of software, human factors, technical feasibility, and cost and schedule feasibility. However, it is overly focused on getting requirements fully defined and on functional decomposition.



DAG Chapter 4: SysE Evaluation: Risk-Driven Activities

- **Strengths**
 - Good general approach, use of P(L), C(L) (4.2.3.5)
 - Good hierarchical risk approach (4.2.3.5)
 - Good overall emphasis on risk (4.3.1, 4.3.2)

- **Shortfalls**
 - Underemphasis on reviewing feasibility evidence as major source of risk (4.2.3.5)
 - ASR underemphasis on feasibility evidence, risk (4.3.1.4.3)
 - No risk reserve approach, identification of top risk sources
 - No approach to risk-driven level of detail, level of activity, earned value management

DAG Chapter 4 has a good overall emphasis on risk, and good recommended risk assessment and risk mitigation techniques, although its waterfall model of the risk management process in Section 4.2.3.5 is surprisingly risk-insensitive. It could say more about how to fund risk mitigation, about how to use risk to determine the appropriate level of detail for processes and products, and about how to use a more risk-based approach to earned value management.



Review of Current DoDI 5000.2 (2003)

- **Strengths**
 - Good set of commitment milestones
 - Emphasizes evolutionary and incremental acquisition
 - Including technology development strategy for next increment
 - Emphasizes maturing technology before committing to develop
 - Spiral development with risk management, demos, user feedback
- **Shortfalls**
 - Underemphasizes pre-milestone B risk management
 - Covers technology maturity, but not other feasibility issues
 - Requirements/architecture, plans, cost-schedule, staffing, acquisition, contracting, operational concept feasibility
 - Needs updating for recent issues
 - Systems of systems, time-defined acquisition, COTS-based systems, multi-timeline systems

DoDI 5000.2 provides a very strong basis for integrating systems and software engineering. Besides having a good set of commitment milestones and emphasizing risk management, evolutionary, incremental, and simulation-based acquisition, it calls for multiple architectural views. Unlike the SEP Guide and DAG Chapter 4, it does not specify functional decomposition as the preferred architectural approach, and emphasizes operational as well as system and technical architecture views. It recommends the type of concurrent increment development and next-increment architecture rebaselining described in charts 54 and 55.

As indicated in the chart, it needs updating for recent issues. But its main improvement need is to expand its Milestone B entrance criteria beyond just technology maturity, approved requirements, and funding. As shown in the next chart, there are several additional risk areas that have caused problems in System Development and Demonstration that should be addressed prior to Milestone B.



Milestone B Focus on Technology Maturity Misses Many OSD/AT&L Systemic Root Causes

- | | |
|--|---|
| 1 Technical process (35 instances)
- V&V, integration, modeling&sim. | 6 Lack of appropriate staff (23) |
| 2 Management process (31) | 7 Ineffective organization (22) |
| 3 Acquisition practices (26) | 8 Ineffective communication (21) |
| 4 Requirements process (25) | 9 Program realism (21) |
| 5 Competing priorities (23) | 10 Contract structure (20) |

- Some of these are root causes of technology immaturity
- Can address these via evidence-based Milestone B exit criteria
 - Technology Development Strategy
 - Capability Development Document
 - Evidence of affordability, KPP satisfaction, program achievability

03/19/2008

©USC-CSSE

55

USD (AT&L)/SSE's root cause analysis of system development problems has identified a number of risk sources above that are not covered by DoDI 5000.2's Milestone B entrance criteria of technology maturity, approved requirements, and funding. Some of the root causes, such as verification and validation (V&V) shortfalls, competing priorities, lack of appropriate staff, and program realism, have been root causes of technology maturity shortfalls, but causes of other shortfalls as well. There are good ways to address these additional risk sources within the scope of the current DoDI 5000.2 by placing stronger emphasis on such documents as the Technology Development Strategy and Capability Development Document, and on providing evidence of program achievability as well as technology maturity at Milestone B.



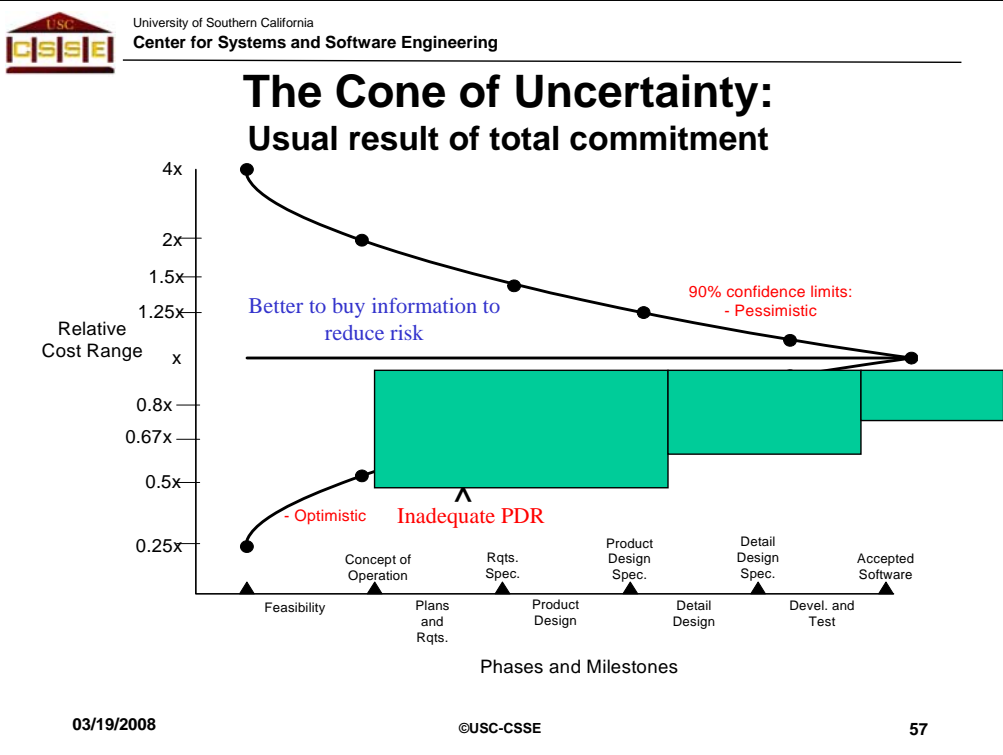
Shared Commitments are Needed to Build Trust

- **New partnerships are increasingly frequent**
 - They start with relatively little built-up trust
- **Group performance is built on a bedrock of trust**
 - Without trust, partners must specify and verify details
 - Increasingly untenable in a world of rapid change
- **Trust is built on a bedrock of honored commitments**
- **Once trust is built up, processes can become more fluid**
 - But need to be monitored as situations change
- **Competitive downselect better than cold RFP at building trust**

Shared Commitments are Needed to Build Trust

This slide discusses the role of trust in partnerships to develop large software-intensive systems. Trust is important with respect to the stakeholders of the system as well as the development organizations, particularly for rapid adaptability to change in DoD OODA loops. Efforts are needed to establish this trust with new partnerships in order to be able to respond to changing requirements and environments. This is related to the third ICM principle:

Incremental and evolutionary growth of system definition and stakeholder commitment. This characteristic captures the often incremental discovery of emergent requirements and solutions via methods such as prototyping, operational exercises, and use of early system capabilities. Requirements and commitment cannot be monolithic or fully pre-specifiable for complex, human-intensive systems; increasingly detailed understanding, trust, definition and commitment is achieved through an evolutionary process.



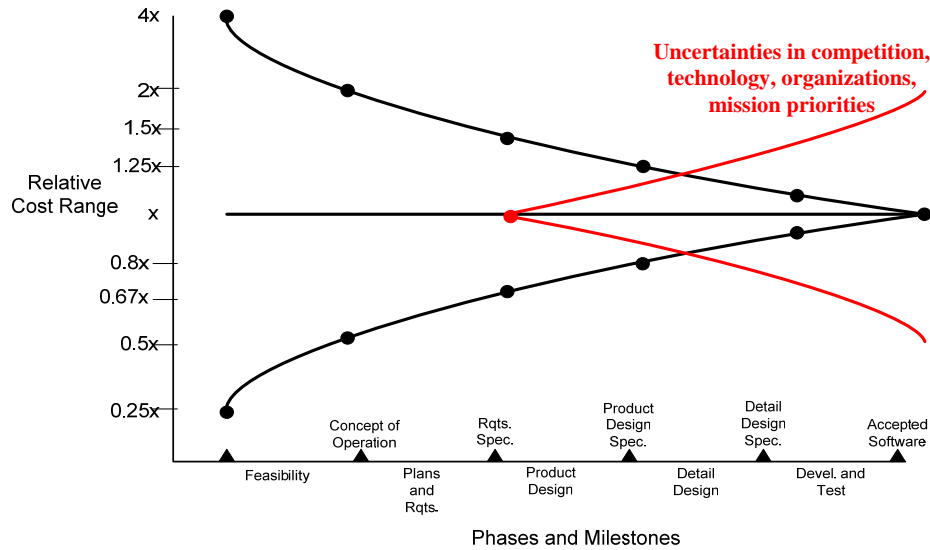
The Cone of Uncertainty: Usual result of total commitment

The Cone of Uncertainty [Boehm, 1981] used empirical data to show the degree to which “if you don’t know exactly what you’re building, you won’t be able to exactly predict its cost.” This view of the cone of uncertainty shows how the total commitment process can lead to competitive bidders making optimistic assumptions about cost. Not only does this set them and the program up for overruns, but it reduces the resources available to perform system architecture and risk resolution by the program’s Preliminary Design Review (PDR). Chart 36 showed the large resulting additional rework costs that will result from this as the development of the system proceeds.

By “buying information” early about the technology and architecture, this risk can be reduced, thereby significantly reducing the size of the rework and the cone of uncertainty much earlier in the development process.



There is Another Cone of Uncertainty: Shorter increments are better



03/19/2008

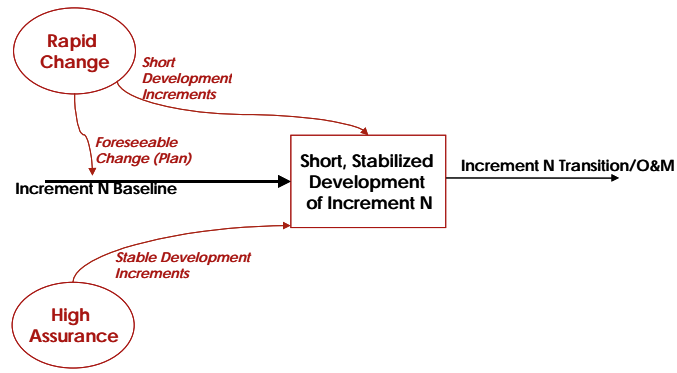
©USC-CSSE

58

There is Another Cone of Uncertainty: Shorter increments are better

Uncertainties in competition and technology evolution and changes in organizations and mission priorities, can wreak havoc with the best of system development programs. In addition, the longer the development cycle, the more likely it will be that several of these uncertainties or changes will occur and make the originally-defined system obsolete. Therefore, planning to develop a system using short increments helps to ensure that early, high priority capabilities can be developed and fielded and changes can be more easily accommodated in future increments.

ICM Stage II: Increment View



03/19/2008

©USC-CSSE

59

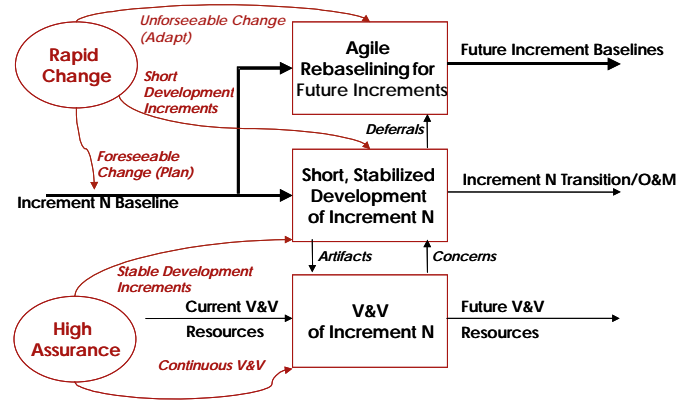
ICM Stage II: Increment View

The ICM is organized to simultaneously address the conflicting challenges of rapid change and high assurance of dependability. It also addresses the need for rapid fielding of incremental capabilities with a minimum of rework.

For high assurance, the development of each increment should be short, stable, and provided with a validated baseline architecture and set of requirements and development plans. The architecture should accommodate any foreseeable changes in the requirements; the next chart shows how the unforeseeable changes are handled.

ICM Stage II: Increment View

A radical idea?



No; a commercial best practice and part of DoDI 5000.2

03/19/2008

©USC-CSSE

60

ICM Stage II: More Detailed Increment View

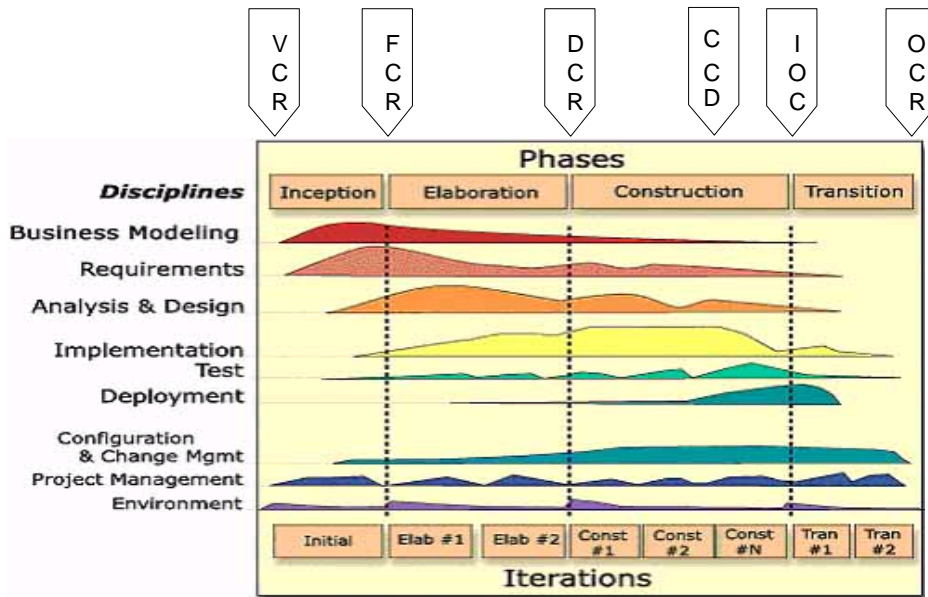
The need to deliver high-assurance incremental capabilities on short fixed schedules means that each increment needs to be kept as stable as possible. This is particularly the case for large, complex systems and systems of systems, in which a high level of rebaselining traffic can easily lead to chaos. In keeping with the use of the spiral model as a risk-driven process model generator, the risks of destabilizing the development process make this portion of the project into a waterfall-like build-to-specification subset of the spiral model activities. The need for high assurance of each increment also makes it cost-effective to invest in a team of appropriately skilled personnel to continuously verify and validate the increment as it is being developed.

However, “deferring the change traffic” does not imply deferring its change impact analysis, change negotiation, and rebaselining until the beginning of the next increment. With a single development team and rapid rates of change, this would require a team optimized to develop to stable plans and specifications to spend much of the next increment’s scarce calendar time performing tasks much better suited to agile teams.

The appropriate metaphor for addressing rapid change is not a build-to-specification metaphor or a purchasing-agent metaphor but an adaptive “command-control-intelligence-surveillance-reconnaissance” (C2ISR) metaphor. It involves an agile team performing the first three activities of the C2ISR “Observe, Orient, Decide, Act” (OODA) loop for the next increments, while the plan-driven development team is performing the “Act” activity for the current increment. “Observing” involves monitoring changes in relevant technology and COTS products, in the competitive marketplace, in external interoperating systems and in the environment; and monitoring progress on the current increment to identify slowdowns and likely scope deferrals. “Orienting” involves performing change impact analyses, risk analyses, and tradeoff analyses to assess candidate rebaselining options for the upcoming increments. “Deciding” involves stakeholder renegotiation of the content of upcoming increments, architecture rebaselining, and the degree of COTS upgrading to be done to prepare for the next increment. It also involves updating the future increments’ Feasibility Rationales to ensure that their renegotiated scopes and solutions can be achieved within their budgets and schedules.

A successful rebaseline means that the plan-driven development team can hit the ground running at the beginning of the “Act” phase of developing the next increment, and the agile team can hit the ground running on rebaselining definitions of the increments beyond.

RUP/ICM Anchor Points Enable Concurrent Engineering



03/19/2008

©USC-CSSE

61

RUP/ICM Anchor Points Enable Concurrent Engineering

This slide illustrates the Rational Unified Process (RUP) and its anchor points. The RUP activities and anchor points were developed to support concurrent engineering and are the basis for the ICM anchor points. In comparison to the software-intensive RUP, the ICM also addresses hardware and human factors integration. It extends the RUP phases to cover the full system life cycle: an Exploration phase precedes the RUP Inception phase, which is refocused on valuation and investment analysis. The RUP Elaboration phase is refocused on system Architecting (a term based on the [Rechtin, 1991] analogy of system architecting to the architecting of buildings, involving concurrent development of requirements, architecture, and plans, to which it adds feasibility evidence); the RUP Construction and Transition phases are combined into Development; and an additional Operations phase combines operations, production, maintenance, and phase-out.



Common Risk-Driven Special Cases of the ICM

Special Case	Example	Size, Complexity	Change Rate % /Month	Criticality	NDI Support	Org. Personnel Capability	Key Stage I Activities : Incremental Definition	Key Stage II Activities: Incremental Development, Operations	Time per Build: per Increment
1. Use NDI	Small Accounting				Complete		Acquire NDI	Use NDI	
2. Agile	E-services	Low	1 – 30	Low-Med	Good; In place	Agile-ready Med-high	Skip Valuation , Architecting phases	Scrum plus agile methods of choice	<= 1 day; 2-6 weeks
3. Architected Agile	Business data processing	Med	1 – 10	Med-High	Good; most in place	Agile-ready Med-high	Combine Valuation, Architecting phases. Complete NDI preparation	Architecture-based Scrum of Scrums	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	Precise formal specification	Formally-based programming language; formal verification	1-5 days; 1-4 weeks
5. HW component with embedded SW	Multi-sensor control device	Low	0.3 – 1	Med-Very High	Good; In place	Experienced; med-high	Concurrent HW/SW engineering. CDR-level ICM DCR	IOC Development, LRIP, FRP, Concurrent Version N+1 engineering	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	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	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	Thorough NDI-suite life cycle cost-benefit analysis, selection, concurrent requirements/ architecture definition	Pro-active NDI evolution influencing, NDI upgrade synchronization	SW: 1-4 weeks; System: 6-18 months
9. Hybrid agile / plan-driven system	C4ISR	Med – Very High	Mixed parts: 1 – 10	Mixed parts; Med-Very High	Mixed parts	Mixed parts	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	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	Full ICM; extensive multi-owner team building, negotiation	Full ICM: large ongoing system/software engineering effort	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	Full ICM; Full stakeholder participation in product line scoping. Strong business case	Full ICM. Extra resources for first system, version control, multi-stakeholder support	1-2 months; 9-18 months

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: Hard ware. IOC: Initial Operational Capability. LRIP: Low-Rate Initial Production. NDI: Non-Development Item. SW: Software

03/19/2008

Common Risk-Driven Special Cases of the ICM: 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. This table contains a list of the special cases of the ICM and an example of each case. Each of the special cases is described in further detail in a companion briefing.



Examples of Risk-Driven Special Cases

4. Software-Embedded Hardware Component Example: Multisensor control device

- **Biggest risks: Device recall, lawsuits, production line rework, hardware-software integration**
 - DCR carried to Critical Design Review level
 - Concurrent hardware-software design
 - Criticality makes Agile too risky
 - Continuous hardware-software integration
 - Initially with simulated hardware
- **Low risk of overrun**
 - Low complexity, stable requirements and NDI
 - Little need for risk reserve
- **Likely single-supplier software makes daily-weekly builds feasible**

03/19/2008

©USC-CSSE

63

Example of Risk-Driven Special Cases: Software-Embedded Hardware Component

This slide shows the typical risks associated with this type of system development and how the ICM can be used to either resolve or mitigate those risks. Where risks are low, one can quickly proceed forward, sometimes even skipping risk mitigation activities for those risks.



Examples of Risk-Driven Special Cases

5. Indivisible IOC

Example: Complete vehicle platform

- **Biggest risk: Complexity, NDI uncertainties cause cost-schedule overrun**
 - Similar strategies to case 4 for criticality (CDR, concurrent HW-SW design, continuous integration)
 - Add deferrable software features as risk reserve
 - Adopt conservative (90%) cost and schedule
 - Drop software features to meet cost and schedule
 - Strong award fee for features not dropped
 - Likely multiple-supplier software makes multi-weekly builds more feasible

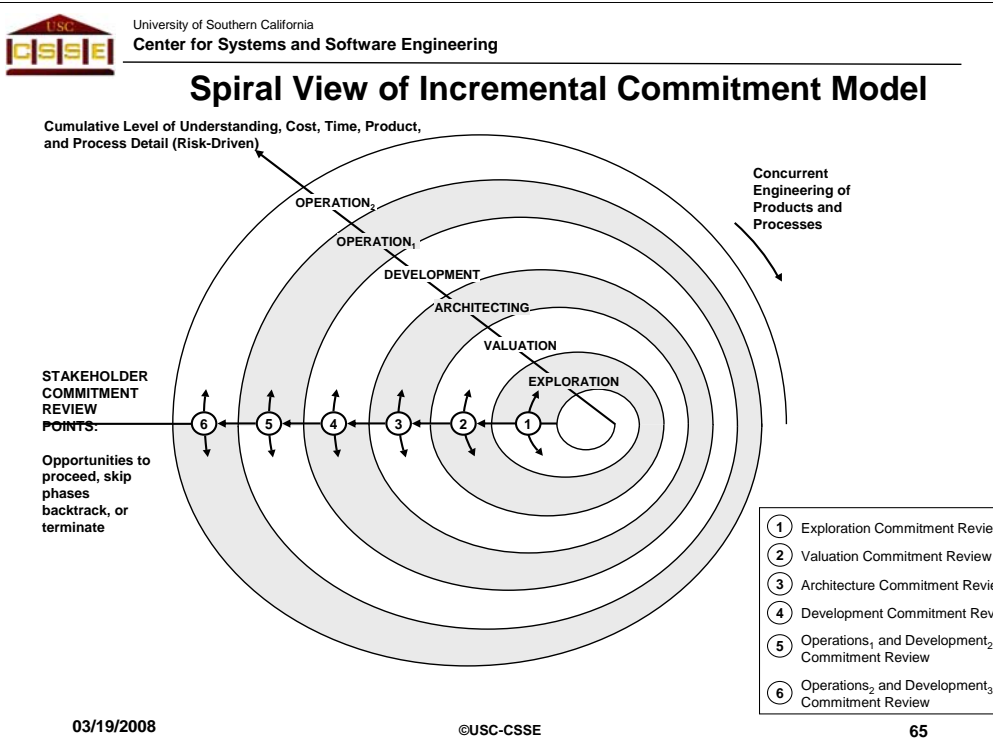
03/19/2008

©USC-CSSE

64

Examples of Risk-Driven Special Cases: Indivisible IOC

In cases such as a new vehicle platform where the project needs to provide a complete vehicle for the Initial Operational Capability (IOC), there may be no features that can be dropped to address new risks. Traditionally, a set of risk reserve funds are recommended to be held back to address such risks as they arise. However, experience has shown that unallocated funds are usually diverted to other purposes and are gone by the time they are needed for risk mitigation. An alternative approach is to allocate the risk reserve funds to the development of borderline-priority software features, that can be deferred to future increments when risk-mitigation funds are needed. To ensure that the risk reserve is judiciously used, the developer should receive a significant award fee for features not dropped.



Spiral View of the ICM

A simplified spiral model view of the ICM is provided in this slide. It avoids sources of misinterpretation in previous versions of the spiral model, and concentrates on the five key spiral development principles. Stakeholder satisficing is necessary to pass the stakeholder commitment review points or anchor point milestones. Incremental growth in system understanding, cost, time, product, and process detail is shown by the spiral growth along the radial dimension. Concurrent engineering is shown by progress along the angular dimension. Iteration is shown by taking several spiral cycles both to define and develop the system. Risk management is captured by indicating that the activities' and products' levels of detail in the angular dimension are risk-driven, and by the risk-driven arrows pointing out from each of the anchor point commitment milestones.

These arrows show that the spiral model is not a sequential, unrollable process, but that it incorporates many paths through the diagram including skipping a phase or backtracking to an earlier phase based on assessed risk. The fourth arrow pointing toward rescoping or halting in previous slides is omitted from view for simplicity; it would be pointing down underneath the plane of this diagram. Other aspects of the spiral model, such as the specific artifacts being concurrently engineered, and the use of the Feasibility Rationale are consistent with their use in the other views, where they are easier to understand and harder to misinterpret than in a spiral diagram. Also for simplicity, the concurrent operation of increment N , development of increment $N+1$, and architecting of increment $N+2$ are not shown explicitly, although they are going on.



Use of Key Process Principles: Annual *CrossTalk* Top-5 Projects

Year	Concurrent Engineering	Risk-Driven	Evolutionary Growth
2002	4	3	3
2003	5	4	3
2004	3	3	4
2005	4	4	5
Total (of 20)	16	14	15

03/19/2008

©USC-CSSE

66

Use of Key Process Principles

A good source of successful projects that have applied the critical success factor principles of the ICM is the annual series of Top-5 software-intensive systems projects published in *CrossTalk* from 2002 to 2005. The “Top-5 Quality Software Projects” were chosen annually by panels of leading experts as role models of best practices and successful outcomes. This slide summarizes each year’s record with respect to usage of four of the six principles: concurrent engineering, risk-driven activities, and evolutionary and iterative system growth (most of the projects were not specific about commitments, accountability, and stakeholder satisficing). Of the 20 top-5 projects in 2002 through 2005, 16 explicitly used concurrent engineering, 14 explicitly used risk-driven development, and 15 explicitly used evolutionary and iterative system growth, while additional projects gave indications of their partial use.



Process Principles in *CrossTalk* 2002 Top-5 Software Projects

	ICM/ Spiral Degree	Concurrent Requirements/ Solution Development	Risk-Driven Activities	Evolutionary Increment Delivery
STARS Air Traffic Control	*	Yes	HCI, Safety	For multiple sites
Minuteman III Messaging (HAC/RMPE)	*	Yes	Safety	Yes; block upgrades
FA-18 Upgrades	*	Not described	Yes	Yes; block upgrades
Census Digital Imaging (DCS2000)	**	Yes	Yes	No; fixed delivery date
FBCB2 Army Tactical C3I	**	Yes	Yes	Yes

03/19/2008

©USC-CSSE

67

Process Principles in *CrossTalk* 2005 Top-5 Software Projects

This slide provides more specifics on the 2005 top-5 projects (one star corresponds to partial use of the principles underlying the ICM and spiral models, two stars corresponds with strong application of the ICM and spiral model principles). Two-page summaries of each project are provided in the January 2002 issue of *CrossTalk*, available via the Internet.



Symbiq IV Pump ICM Process

- **Exploration Phase**
 - Stakeholder needs interviews, field observations
 - Initial user interface prototypes
 - Competitive analysis, system scoping
 - Commitment to proceed
- **Valuation Phase**
 - Feature analysis and prioritization
 - Display vendor option prototyping and analysis
 - Top-level life cycle plan, business case analysis
 - Safety and business risk assessment
 - Commitment to proceed while addressing risks

Symbiq IV Pump ICM Process – Exploration and Valuation Phases

This slide summarizes the key activities for the Symbiq IV pump Exploration and Valuation phases. It highlights the need for developing a detailed understanding of the pump capabilities up front by using a combination of field observations and user interface prototypes, then in the valuation phase, prioritizing features and understanding risks going forward.



Symbiq IV Pump ICM Process

- **Foundations Phase**
 - Modularity of pumping channels
 - Safety feature and alarms prototyping and iteration
 - Programmable therapy types, touchscreen analysis
 - Failure modes and effects analyses (FMEAs)
 - Prototype usage in teaching hospital
 - Commitment to proceed into development
- **Development Phase**
 - Extensive usability criteria and testing
 - Iterated FMEAs and safety analyses
 - Patient-simulator testing; adaptation to concerns
 - Commitment to production and business plans

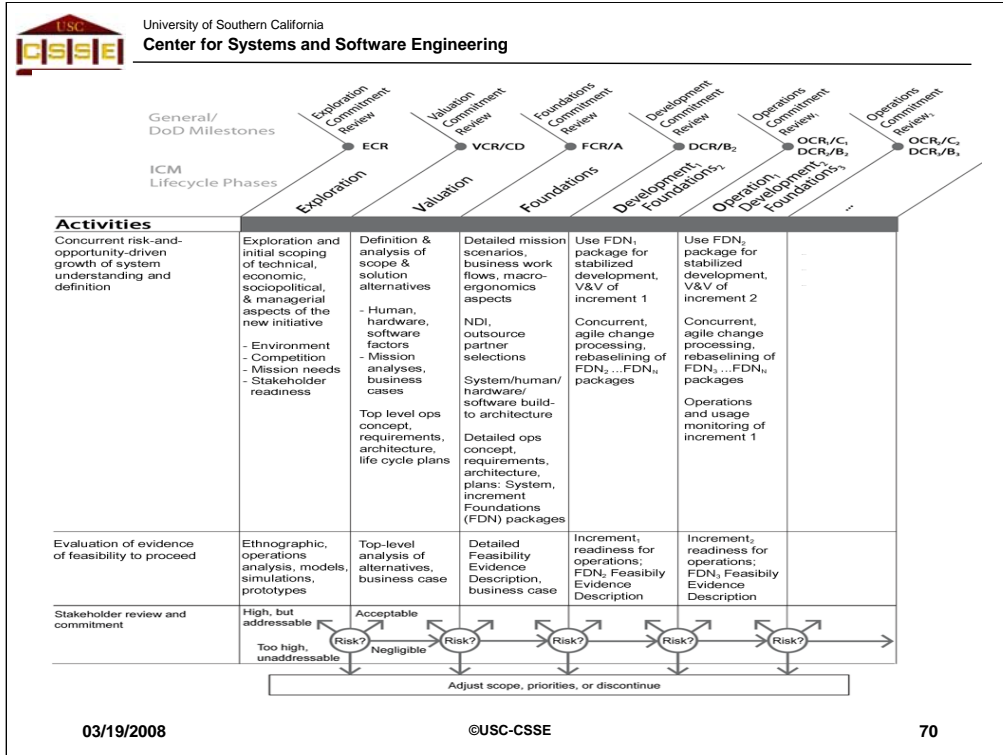
03/19/2008

©USC-CSSE

69

Symbiq IV Pump ICM Process – Foundations and Development Phases

This slide summarizes the key activities for the Symbiq IV pump Foundations and Development phases. It highlights the need for continual analyses and feasibility assessments as development proceeds and the decision to go into production is made. At each milestone, risks were assessed and a commitment to go forward was made by the key Abbott Laboratories managers and representatives of their supplier and consumer communities, based on their particular risk/reward ratios.



This slide provides more detailed descriptions of the activities concurrently going on during the various ICM phases. An FDN package contains all of the Foundations material (in risk-driven levels of detail) needed for its independent expert review preceding its Development Commitment Review: operations concept, requirements, architecture, plans, business case, feasibility evidence, and risk management plans as necessary.