

# Chapter 8

## A Process for Interoperability

**By**  
**Cadet Pamela A. Sanders, US Army**  
**Dr. John A. Hamilton, Jr., Auburn University**

*The views expressed in this chapter are the opinions of the authors, and do not reflect the official opinions of any U.S. government agency.*

### 8.1 INTRODUCTION

Commercial industry has learned that proprietary technology items are often relegated to either niche markets or oblivion. The free market can often drive interoperability in consumer goods. Unfortunately, strictly economic factors do not drive the high reliability requirements within Defense applications. As Rear Admiral John A. Gauss, USN said when he was the interoperability chief at DISA, “We have built a bunch of state-of-the-art, open systems, TAFIM-compliant stovepipes [Bass, Clements and Kazman 1998].” Economic factors alone are not sufficient to drive interoperability in Defense applications. It is clear that in the long term, the DoD would enjoy significant economic as well as military advantages from interoperable command and control systems. Unfortunately, short-term budgetary, regulatory and political factors make it difficult to expend resources against promises of interoperability in the future. For this reason, the authors propose an interoperability process to help guide program managers in their efforts to achieve long-term interoperability.

When the issue of system interoperability is discussed in the realm of The United States Armed Forces there is one central area of concern: mission critical systems. Mission critical systems upon failure “may have catastrophic results, such as serious injury, loss of life or property, mission failure, or breach of security [Croll 2000].” Examples of mission critical systems include: weapons systems, avionics systems, command and control systems, intelligence systems, and communication

systems [Croll 2000]. Furthermore, the most frequently mentioned national issues directly related to mission critical systems include homeland defense, international security, and biological warfare.

In the areas of technology and communications, mission critical systems depend heavily upon system interoperability of hardware, software, and the interaction between hardware and software components. However, acknowledging this fact does not investigate the true nature of how to achieve this goal with both new and legacy systems. There have been many definitions, discussions, and categorizations of system interoperability. The evolution of this topic is to discuss the process of achieving system interoperability. Specifically, the authors suggest a System Interoperability Software Model to achieve system interoperability aimed at addressing issues involving mission critical systems focused on technology and communications.

## **8.2 SYSTEM INTEROPERABILITY: CONCEPTUAL DEFINITION**

### **8.2.1 IEEE**

A conceptual definition of system interoperability involves describing the term through its industry standard meanings and further embellishing these ideas for a comprehensive definition. The *IEEE Standard Glossary of Software Engineering Terminology* defines interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged. *See also: compatibility* [IEEE 1990].” Compatibility is defined as “(1) The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment. (2) The ability of two or more systems or components to exchange information [IEEE 1990].” As seen from these definitions, the concept of interoperability begins with a theoretical idea of the following structures having symmetry: software components, hardware components, the interaction between software and hardware components, and the communication between systems exchanging information.

### **8.2.2 DEPARTMENT OF DEFENSE DIRECTIVE**

According to the *Department of Defense Directive*, information technology (IT) is defined as “any equipment, or interconnected system or subsystem of equipment, that is used in the automatic acquisition, storage, manipulation, management, movement, control, display, switching, interchange, transmission, or reception of data or information by the executive agency...The term ‘IT’ also includes computers, ancillary equipment, software, firmware and similar procedures,

services (including support services), and related resources... The term 'IT' includes National Security Systems (NSS) [Department 2002]." Furthermore, the *Department of Defense Directive* defines interoperability as "the ability of systems, units or forces to provide data, information, materiel, and services, to and accept the same from other systems, units, or forces and to use the data, information, materiel, and services so exchanged to enable them to operate effectively together."

These two industry standard meanings of system interoperability provide a definition concerning software compatibility, hardware compatibility, the interaction between software and hardware components, and the reliable communication and exchange of information. These combined definitions are suitable for a rudimentary understanding in exploring the concept of system interoperability. There are other issues that widen this perspective in discussing system interoperability when analyzing the realistic applicability of this term. System interoperability involves the efficiency of producing and refining software to interact with new and existing hardware to reliably exchange information between systems. More specifically, system interoperability encompasses the idea of software process and software modeling involving legacy systems and new hardware/software that must interact to provide reliable exchange of information between systems.

### **8.3 SYSTEM INTEROPERABILITY PROCESS MODEL**

#### **8.3.1 DISTINGUISHING SOFTWARE PROCESSES AND SOFTWARE MODELS**

What is the difference between a software process and a software model? To begin with, the term "process" is defined as "a series of actions, changes, or functions bringing about a result [American 2000]." The term "model" is defined as "one serving as an example to be imitated or compared [American 2000]." From these basic definitions it is clear that a process involves "actions" taken to succeed a goal while a model is the actual implementation of the process. A "process model" is a model of a process or simply an example of actions to be taken.

For example, an OPORD structure is a process because it consists of a set of actions that lead to a successful goal (communication) which always consists of the following successive elements (subprocesses): situation, mission, execution, service support, and command and signal. Furthermore, the execution element in the OPORD is a model because it contains the actual implementation of the mission to be completed involving the concept of operation, tasks, coordinating

instructions, and safety. In this example the process (OPORD structure) remains the same and the model of execution may change due to the mission. Essentially, a goal of these authors is to present the idea that processes and models must be both flexible and synchronous in order for efficient engineering to occur.

### **8.3.2 STANDARDS: DOD-STD-2167A, IEEE/EIA 12207, LISI**

Many believe that simply defining standards for interoperability makes it easy to achieve. On the surface that makes sense: If an organization defines a technical parameter and all systems must comply with it, then it follows that the systems will be interoperable. In reality, this is much more difficult than it seems, because as technology changes so (naturally) do standards. Given the complexity of systems and the constant push to acquire the latest technology, defining standards for interfacing the new with the old or even the new with the new has proven tremendously complicated [Faughn 2001].

This statement briefly summarizes the difficulty in establishing standards that meet the expectations of organizational demands while synchronously meeting the criteria of system interoperability involving legacy and new systems, software and hardware concerns, and multitudes of subjective definitions... “What standards will we follow in this particular project?” In discussing the System Interoperability Process Model, there are three established standards for software modeling mentioned throughout the literature: DOD-STD-2167A, IEEE/EIA 12207, and LISI. These standards are discussed briefly because they are resources that support and enhance the implementation of a generic System Interoperability Process Model that is not project specific.

#### **8.3.2.1 DOD-STD-2167A**

In the section entitled “4.1.1 Software development process,” the Department of Defense states “the software development process shall include the following major activities, which may overlap and may be applied iteratively or recursively:

- a. System Requirements Analysis/Design
- b. Software Requirements Analysis
- c. Preliminary Design
- d. Detailed Design
- e. Coding and CSU Testing
- f. CSC Integration and Testing
- g. CSCI Testing
- h. System Integration and Testing

[\*See Appendix A for abbreviations] [Military 1988]

Therefore, the process involved in developing software is a flexible system which includes the previous elements. As far as software development methods are concerned, section 4.2.1 states “the contractor shall use systematic and well documented software development methods to perform requirements analysis, design, coding, integration, and testing of the deliverable software. The contractor shall implement software development methods that support the formal reviews and audits required by the contract [Military 1988].” In summary, both the software development process and the software development method include certain “general elements” that are recommended for efficient software engineering practices.

Although DOD-STD-2167A is no longer in force, it still occupies a prominent place in software engineering literature. Although many criticized DOD-STD-2167A for a variety of reasons, it still represented a blueprint for software development and acquisition. It is not clear that abandonment of DOD-STD-2167A has improved either software procurements or interoperability in the DoD.

### **8.3.2.2 IEEE/EIA 12207**

IEEE/EIA 12207 is a standard that lists specific process activities in the software lifecycle that enhance unique adaptation in different projects. The “Life Cycle Processes” are broken down into five primary processes, eight supporting processes, and four organizational processes. The five primary life cycle processes are acquisition, supply, development, operation, and maintenance. The eight supporting life cycle processes are documentation, configuration management, quality assurance, verification, validation, joint review, audit, and problem resolution. The four organizational life cycle processes are management, infrastructure, improvement, and training [See Figure 1][See Appendix B for further detailed descriptions]. The primary, supporting, and organizational life cycle processes are then broken down into activities with assigned tasks for each activity.

This standard is important to mention since the relationships between individual life cycle processes have been predetermined and well defined, which is an essential task that must be implemented in efficient software engineering. However, this model of the software life cycle is based upon “activities” which can easily become a breeding ground for system non-interoperability involving all aspects of software, hardware, and reliable information transmission if the primary processes are not in synch with the supporting life cycle process and the organizational life cycle processes. For example, a legacy system needs a software upgrade to transfer sensitive information overseas with specific hardware. The supplier must know the limitations of developing the software

before contracting with the acquirer, thus discussions with the developer and management must be the portal for this discussion. Management may or may not have the knowledge or understanding of system interoperability at the developer level. Furthermore, the acquirer may not understand this concept either when relating to the supplier what software is initially to be developed.

These “communication” difficulties are most likely to occur with any software lifecycle development based upon “activity” developmental processes; and, system non-interoperability is highly likely due to so many “activities” (primary process parties) being directly involved in the software lifecycle. In conclusion, the System Interoperability Process Model encompasses the “development” aspect included in IEEE/EIA 12207 since these factors directly involve resolution to system interoperability in a software process model [See Appendix C].

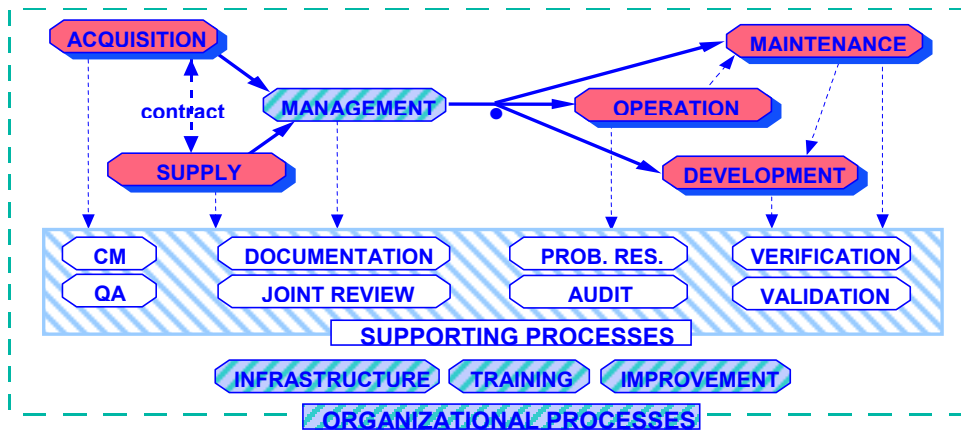


Figure 8-1 *IEEE/EIA 12207 Software Life Cycle Processes*

### 8.3.2.3 LISI

The purpose of Levels of Information Systems Interoperability (LISI) is “to provide DOD with a maturity model and a process for determining joint interoperability needs, assessing the ability of our information systems to meet those needs, and selecting pragmatic solutions and a transition path for achieving higher states of capability and interoperability [C4ISR 1998].” LISI is divided into two main elements: LISI Assessment Basis and LISI Assessment Products. LISI Assessment Basis consists of four parts: the Interoperability Maturity Model, the Reference Model, the Capabilities Model, and the Implementation Options Tables. LISI Assessment Products consists of five parts: Interoperability Profiles,

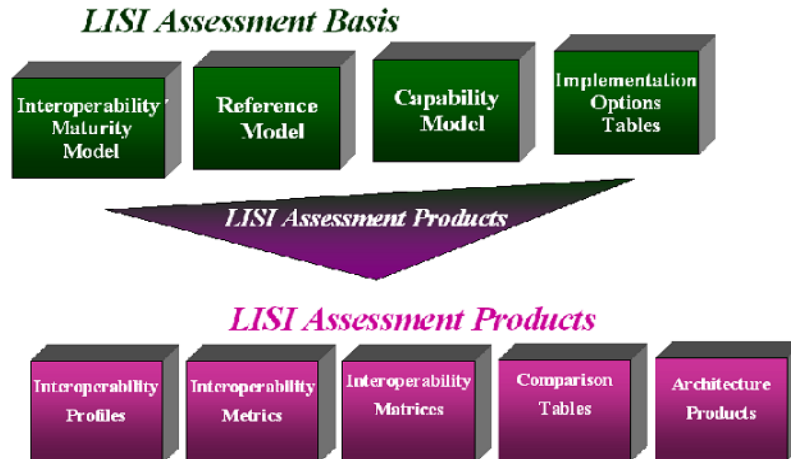


Figure 8-2 Levels of Information Systems Interoperability (LISI)

Interoperability Metrics, Interoperability Matrices, Comparison Tables, and Architecture Products [C4ISR 1998][See Figure 2].

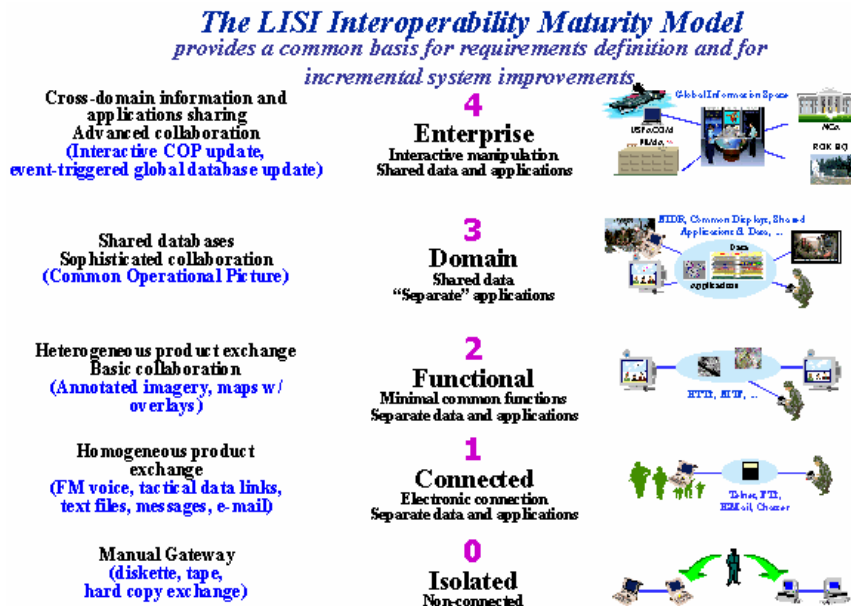


Figure 8-3 LISI Interoperability Maturity Model

The Interoperability Maturity Model classifies five levels of “the general nature of the interoperability” as Isolated, Connected, Functional, Domain and Enterprise [C4ISR 1998][See Figure 3] [See Appendix D for further descriptions]. The Reference Model embellishes the five levels of interoperability by describing each with four attributes (PAID): procedures, applications, infrastructure, and data [C4ISR 1998] [See Figure 4].

Nature of Operational Information Interaction	Corresponding Computing Environment	Level Code	Implications			
			P	A	I	D
Cross-Domain Interactive Manipulation	Universal	4	Enterprise Level	Interactive	Multiple Topologies	Enterprise Model
Shared Applications & Databases	Integrated	3	Domain Level	Groupware	World Wide Networks	Domain Model
Complex Media Exchange	Distributed	2	Program Level	Desktop Automation	Local Networks	Program Model
Simple Electronic Exchange	Connected	1	Local/Site Level	Standard System Drivers	Simple Connection	Local
Manual Gateway	Isolated	0	Access Control	N/A	Independent	Private

Figure 8-4 *The LISI Reference Model*

The specific advantages of LISI are as follows:

- The *suite of capabilities* associated with procedures, applications, infrastructure, and data that must be inherent in an information system to achieve each level of interoperability
- The *implementation options* that are available for each prescribed capability, including clear distinctions between those options that conform with current DoD technical criteria (e.g., JTA, DII COE, SHADE,...) and those that do not
- A practical *assessment process* for determining the interoperability maturity level of a given system or system pair, capabilities that may be lacking, implementations that are not compatible, and options available for resolving deficiencies and for achieving progressively higher levels of maturity [Architectures 98]

In essence, LISI provides a complete, descriptive model of classification with levels of interoperability based on individual, unique project specifications. In fact, LISI has a wide community of users including DISA, USMC, USAF, ARMY, SPAWAR, NAVY, J6, Coast Guard, and many others outside the DoD

[Levels 2002]. LISI has provided a benchmark for system interoperability “level” descriptions though the benchmark is softer than is initially apparent as the LISI model essentially compares things that are not mutually commensurable. However, LISI is not a process model of development as is suggested by developers of LISI.

LISI has similar structural difficulties to the IEEE/EIA 12207 Software Lifecycle Process Model where too many contributing “parties” are involved in the process and the model. First, the “processes” involve a questionnaire, metric calculation, and a descriptive profile based on tables such as Requirements Comparisons and Data Comparisons Tables. Also, LISI speculates on timely technology updates entered into the query database along with updated profiles. In simplified terms, the end result of these “processes” is a categorization output derived from a questionnaire input. Second, the “model” aspect is also suspect since it should be based upon the “process” itself (which is not robust).

In summary, dynamic systems engineering requires that both processes and models be flexible to allow reengineering and reevaluation of the processes and models themselves. LISI does not provide this flexibility through either processes or model since the execution of processes is in a linear, chronological order: questionnaire, metrics, and description. Nevertheless, LISI is invaluable in the System Interoperability Process Model where categorizations of “levels” and scalability concerns are always requirements; and, LISI can be fully incorporated into the System Interoperability Process Model as a facet of the model itself.

### **8.3.3 SYSTEM INTEROPERABILITY PROCESS MODEL: BOEHM'S SPIRAL MODEL**

Boehm's Original Spiral Model [Figure 5] is based upon risk analysis, which is the trademark of later developed spiral models such as the Win-Win Spiral Model. General characteristics of a spiral model of software process are summarized best by Kotonya and Sommerville who state, “a spiral model of the requirements engineering process illustrates that the process is iterative and involves repetition of elicitation, analysis and validation activities [Kotonya and Sommerville 1998].” In other words, a spiral model of software process involves several cycles of reanalysis, reengineering, and flexibility to changes inherent in the maturity of the software's lifetime. Moreover, this flexibility in the software development process is extremely important in discussing interoperability of both hardware and software, since this allows for improvement of both new and legacy systems. Specifically, a waterfall or checklist process of software development leaves little room for reevaluation of the successes and failures of previous implementations due to the structure of the process itself. A spiral lifecycle process model allows for reengineering, and documentation of such, so that the actual product of labor

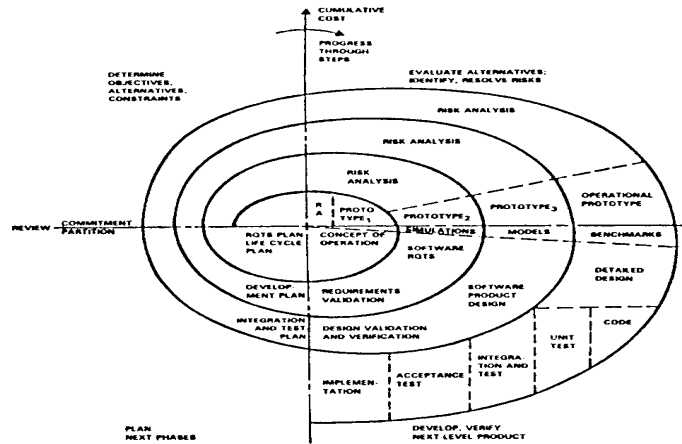


Figure 8-5 *The Original Spiral Model*

can be compared to the intentional goal of completion, otherwise known as validation.

There are also several other benefits to choosing a spiral model of software development that specifically address the system interoperability issues at present with military communication systems. To give one example, legacy systems are not extinct if a level of interoperability can be achieved through changing the process of “how” these systems are redesigned for efficient communication. Software engineers implement a widely used design method to “bridge” the gap between non-interoperable systems by creating an “interface.” This idea is very similar to having a translator interpret two languages and relay the communication between two sources. In linear or checklist software process models there is simply no way to create this interface without overhauling the entire process from the very beginning. This is extremely costly, inefficient, and unnecessary. A spiral model of software process allows for this flexibility to occur at the next iteration of development, and this will be easier and more efficient with requirements, standards, and a process already in tact. Therefore, the System Interoperability Process Model begins with a Spiral Model backbone [See Figure 6].

#### 8.3.4 SYSTEM INTEROPERABILITY PROCESS MODEL: PDSA CYCLE

In 1950, William Deming introduced the first version of the PDSA Cycle, which he revised several times until his death in 1993. The PDSA Cycle consists of four steps: Plan, Do, Study, and Act. In the table below, Deming presents definitions

of each step to represent improvement for process or product in the cycle [Deming 2000].

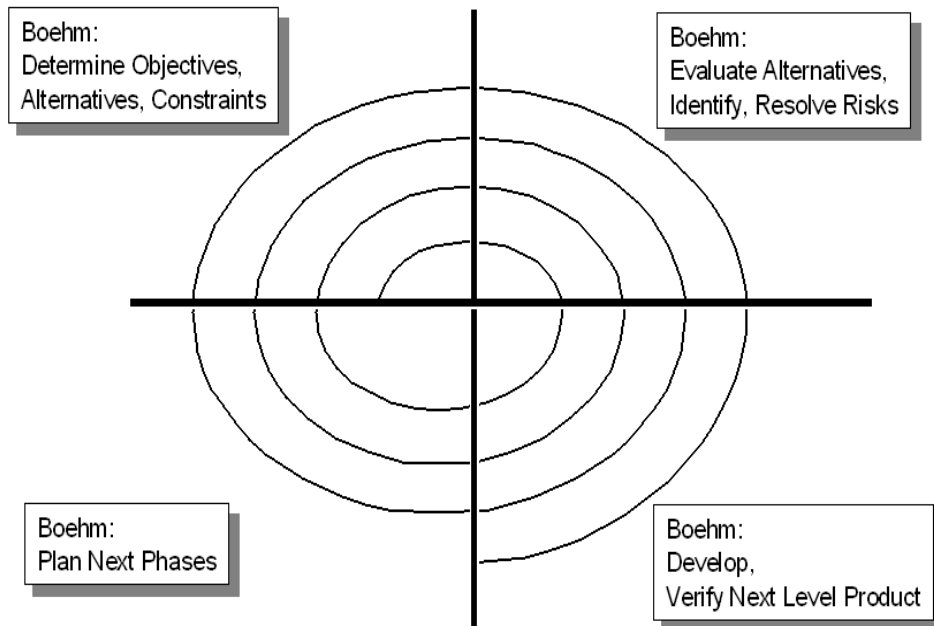


Figure 8-6 *System Interoperability Process Model (backbone)*

#### 8.3.4.1 THE PDSA CYCLE

Deming's Plan, Do, Study, Act model [Deming 2000] is summarized in Table 1 and Figure 7. Military readers familiar with Colonel John Boyd's "OODA Loop" (Observe, Orient, Decide, Act) will immediately recognize the same underlying principles in Deming's model.

Due to the popularity of Deming's PDSA Cycle, other similar models have been developed through the years based on the four key steps: Plan, Do, Study, and Act. The PDCA Cycle is a typical example of a cycle based upon Deming's work; however, the expansion is upon specific activities that occur in each step. For example, In Deming's cycle the "Study" step consists of seeing what went wrong and if expectations were fulfilled while Koehler and Pankowski explicitly

cite “get feedback” and “communicate with suppliers and customers” in their renamed, equivalent “Check” step.

**Table 8-1** Deming’s PDSA Cycle Steps: Improvement of a Product or a Process

<p><b>Step 1: Plan</b></p>	<p>Somebody has an idea for improvement of a product or of a process.</p> <p>This is the 0-th stage embedded in Step 1.</p> <p>It leads to a plan for a test, comparison, experiment.</p> <p>Step 1 is the foundation of the whole cycle.</p> <p>A hasty start may be ineffective, costly, and frustrating.</p> <p>People have a weakness to short-circuit this step.</p> <p>They cannot wait to get into motion, to be active, to look busy, move into Step 2.</p> <p>The planning stage may start with a choice between several suggestions.</p> <p>Which one can we test? What may be the result?</p> <p>Compare the possible outcomes of the possible choices.</p> <p>Of the several suggestions, which one appears to be most promising in terms of new knowledge or profit? The problem may be how to achieve a feasible goal.</p>
<p><b>Step 2: Do</b></p>	<p>Carry out the test, comparison, or experiment, preferably on a small scale, according to the layout decided in Step 1.</p>
<p><b>Step 3: Study</b></p>	<p>Study the results. Do they correspond with hopes and expectations?</p> <p>If not, what went wrong?</p> <p>Maybe we tricked ourselves in the first place, and should make a fresh start.</p>
<p><b>Step 4: Act</b></p>	<p>Adopt the change. OR</p> <p>Abandon it. OR</p> <p>Run through the cycle again, possibly under different environmental conditions, different materials, different people, different rules.</p>

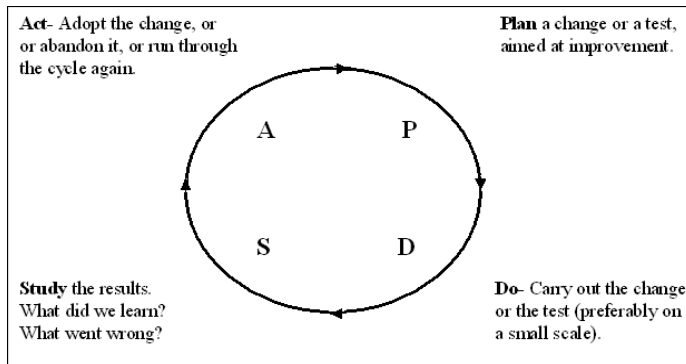


Figure 8-7 Deming’s Flow Diagram: Improvement of a Product or a Process

The System Interoperability Model Process Model incorporates Deming’s PDCA Cycle as a complement to Boehm’s Spiral Model in dividing the Axes into conceptual, separate aspects [See Figure 9].

**8.3.4.2 The PDCA Cycle**

Koehler and Pankowski’s modification of the Deming model is shown in Figure 8 and described in Table 2 [Koehler and Pankowski 1996].

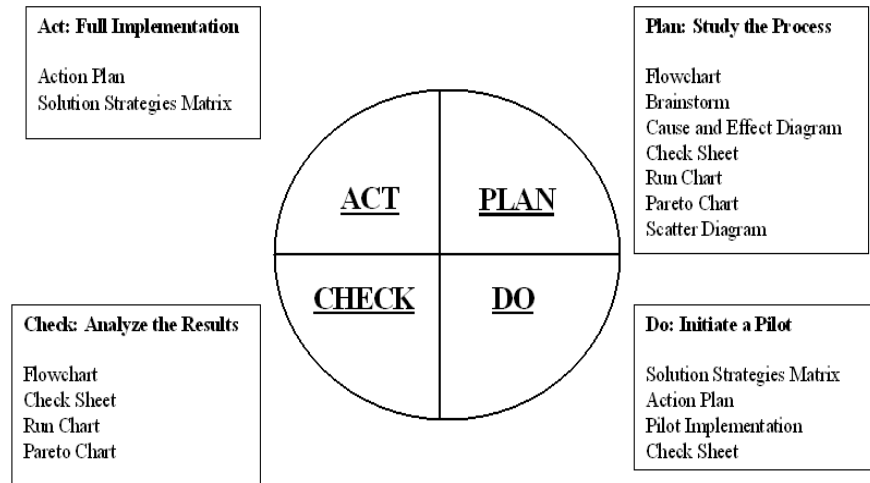


Figure 8-8 Koehler and Pankowski’s Flow Diagram

**Table 8-2** Koehler and Pankowski’s PDCA Cycle Steps

<i>Plan</i>	Communicate with agency leaders, especially the ones who will empower you in a particular team Develop awareness of process improvement needs among leaders Team leader and team facilitator jointly plan early meetings carefully; study how to conduct successful meetings Plan for inputting data to statewide team tracking system Develop plan for particular roles and how to select individuals Have knowledge of process improvement steps Develop process improvement supports: (1) values, (2) vision, (3) mission, and (4) goals Keep sharp focus: Study Deming’s fourteen points; watch and discuss videos on how to
-------------	---

<i>Plan (cont.)</i>	conduct successful meetings. Analyze process, using TQM goals. Flowchart (get large picture first, then refine to more detail later) Identify measuring points Benchmark with data collected; use checklist if needed Use cause and effect diagram Use Pareto chart Choose solution; identify elements of process change Modify flowchart
<i>Do</i>	Apply solutions; put process improvements to work Use control chart; narrow the variation
<i>Check</i>	Monitor changes Get feedback Communicate with suppliers and customers
<i>Act</i>	Make personnel adjustments if necessary Write policy revisions based upon the changes Modify agency procedures and/or rules based upon changes

The end result of this is a simplified comparison and contrast between Boehm's spiral model and Deming's model as shown in Figure 9. This is the basis for our proposed System Interoperability Model Process Model.

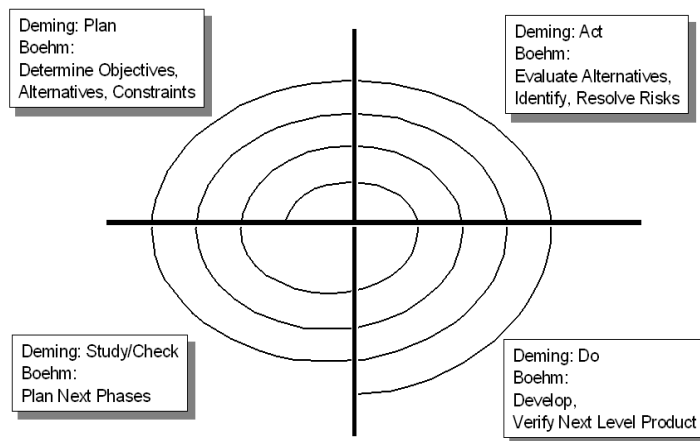


Figure 8-9 *System Interoperability Process Model (Deming Cycle Addition)*

### 8.3.5 SYSTEM INTEROPERABILITY PROCESS MODEL: SYSTEM ENGINEERING PROCESS

The concept of system engineering is similar to system interoperability in the respect that software, hardware, and the interaction of these components must always be considered in the analysis and development of a system. The system engineering process consists of using this analysis to develop, reengineer, and speculate upon the functionality of a system(s) for specific goals. These specific goals are always user dependent and generally systems fall into three categories to fulfill these specific goals: information systems, embedded systems, and command and control systems [Kotonya & Sommerville 1998]. The System Interoperability Process Model focuses upon command and control systems to which Kotonya and Sommerville clearly state, “requirements engineering for these systems includes hardware and software specification and a specification of the operational procedures and processes.”

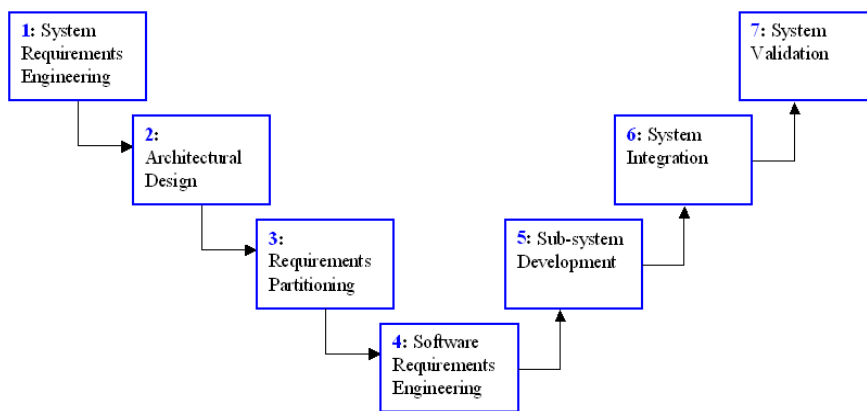


Figure 8-10 *The Systems Engineering Process* [Kotonya & Sommerville 1998]

The System Interoperability Model Process includes Kotonya and Sommerville’s Systems Engineering Process Activities to create a more structured, in-depth model of process by subdividing three Axes (Incubation, Hatching, and Maturation) into sub-Axes. Furthermore, the System Interoperability Process Model requires this structure, so that at each stage of cyclical development there is a core, systems engineering practice associated with specific activities. The process is diagrammed in Figure 11. An explanation of the elements in Figure 11 is included in Table 3.

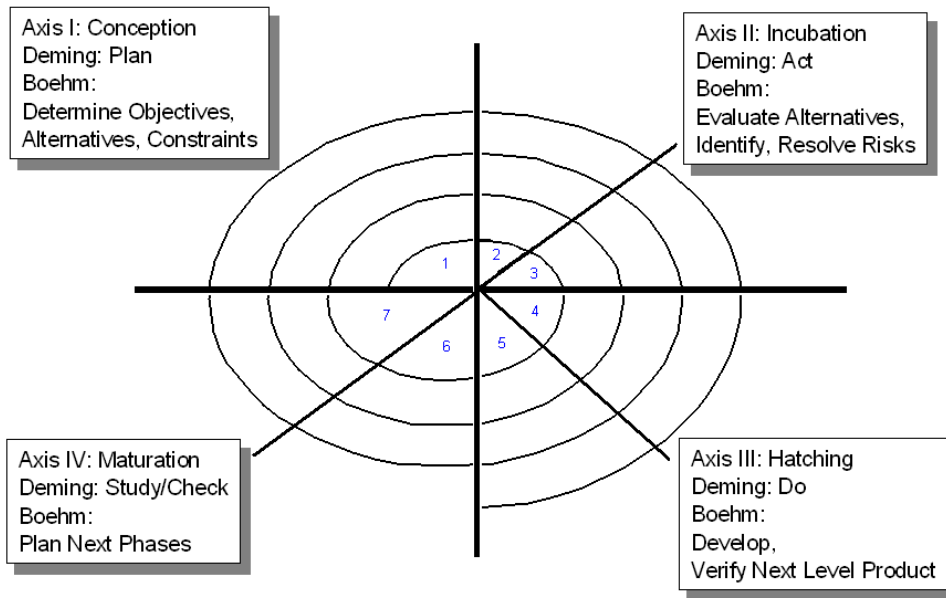


Figure 8-11 *System Interoperability Model*  
(System Engineering Process Activities Addition)

Table 8-3 The Systems Engineering Process Activities

<u>Activity</u>	<u>Description</u>
1. System requirements engineering	The requirements for the system as a whole are established. These will usually be expressed in a fairly high-level fashion and written in natural language. Some detail constraints (such as compatibility constraints) may be included is these are critical for the success of the system.
2. Architectural design	The system is decomposed into a set of independent sub-systems.
3. Requirements partitioning	The requirements are partitioned to these sub-systems. At this stage, decisions may be made about whether requirements should be hardware or software requirements.
4. Software requirements engineering	The high-level software requirements are decomposed into a more detailed set of requirements for the software components of the system.
5. Sub-system development	The hardware and the software subsystems are designed and implemented in parallel.
6. System integration	The hardware and software subsystems are put together to complete the system.
7. System validation	The system is validated against its requirements.

### 8.3.6 SYSTEM INTEROPERABILITY SOFTWARE MODEL: C4ISR ARCHITECTURE FRAMEWORK

The Joint Technical Architecture (JTA) was developed to provide DOD systems with the basis for the seamless interoperability necessary to ensure that we can truly conduct joint operations [Hamilton, Murtagh, and Deal 1999].

In relation to the Joint Technical Architecture, the C4ISR Architecture Framework dominates the design of the System Interoperability Software Model consisting of three intrinsic tiers as follows: the Operational Architecture View, the Systems Architecture View, and the Technical Architecture View [Hamilton and Deal 1998]. The Operational Architecture view involves the main ideas of “which units communicate, which data to, which other units, via which systems.” In a software engineering model, this represents an object-oriented approach to designing implementations. The Systems Architecture View “answers the ‘How?’ question in response to the ‘Why?’ from the Operational Architecture View” in a connectivity diagram [Deal 1997]. In an electrical engineering model, this represents the physical wires and equipment required to accomplish the model

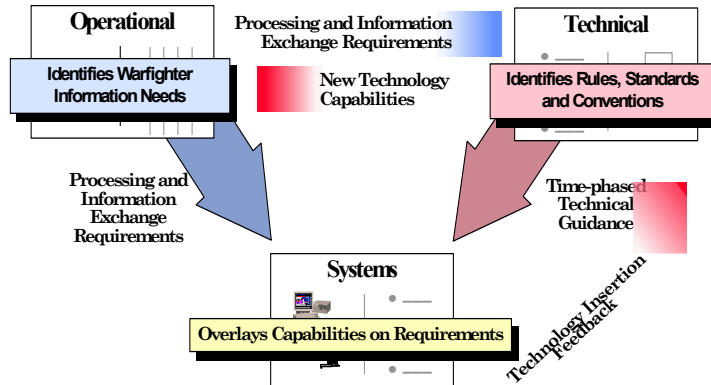


Figure 8-12 *C4ISR Architecture Views*

designed in the Operational Architecture View. The Technical Architecture View creates a “set of implementation guidance standards” such as DOD-STD-2167A, IEEE/EIA 12207, LISI, WIN32 API, TCP/IP, and X-windows (X11R5) [Hamilton 1998] [See Figure 12]. Further detailed descriptions of the C4ISR Architecture Views are directly quoted from the *C4ISR Architecture Framework Version 2.0* in Appendix E.

Legacy systems will continue to dominate the initiation choice for a process model of development and enforce the idea that the C4ISR Architecture Views will be heavily relied upon in relation to software issues handling existing hardware and software incompatibilities. Therefore, waterfall and checklist process models fail to reevaluate the system interoperability issues that arise in software engineering projects, as development is an ongoing process of change, reanalysis, and reengineering. In the System Interoperability Process Model, the C4ISR Architecture Views focus upon all stages of development involving system interoperability, specifically any software and hardware issues that arise in each Axis of development. In summary, the C4ISR Architecture Views are necessary in every stage of the System Interoperability Process Model, a model of creating and reengineering software specifically designed to enhance system interoperability based upon these three tiers of analysis [See Figure13].

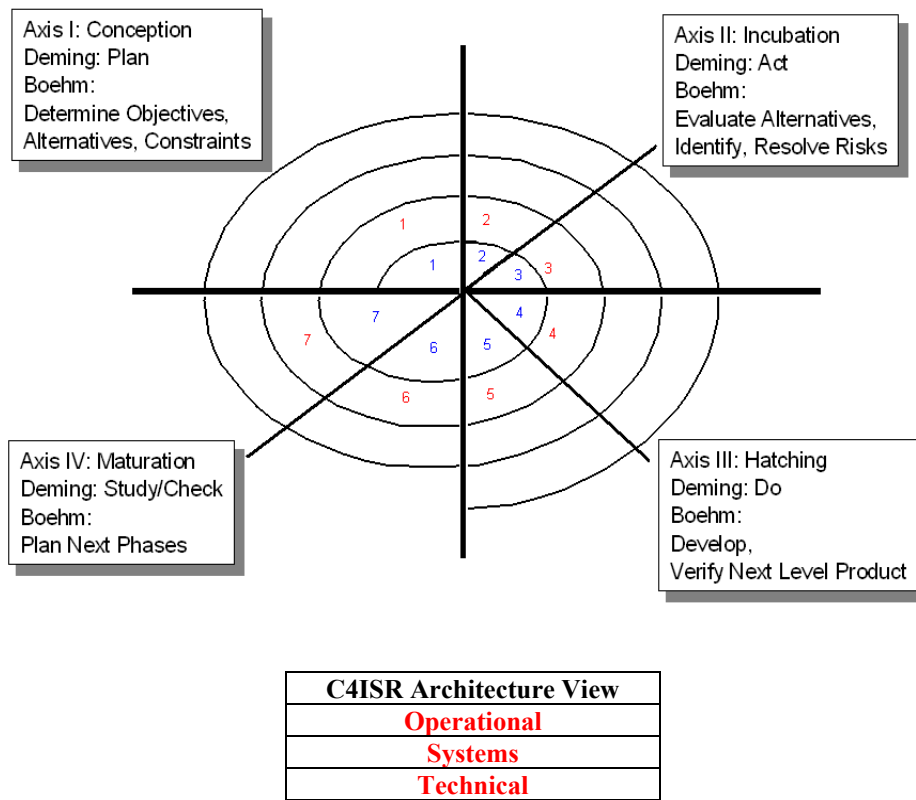


Figure 8-13 *System Interoperability Process Model (C4ISR Addition)*

### 8.3.7 SYSTEM INTEROPERABILITY PROCESS MODEL

#### 8.3.7.1 AXIS I: CONCEPTION

*Deming: Plan*

*Boehm: Determine Objectives, Alternatives, Constraints*

Systems Engineering Process Activity: [1. System Requirements Engineering](#)

C4ISR Architecture Views: [Operational](#), [Systems](#), [Technical](#)

Operational View:

Achieve a comprehensive analysis of the records containing information about which systems communicate data to what other systems. Attention should be paid to groupings (such as sub-system design) since Axis II will require the dissection of the system into sub-systems. Note any signs of system interoperability constraints (or suspicion of such).

Systems View:

Create or locate the “connectivity diagrams” to get a general picture of the setup. Creation and modification of diagrams need to be as comprehensive as possible in Axis I since other Axes will depend on the thoroughness of these initial diagrams. Attention should be paid to groupings (such as sub-system design) since Axis II will require the dissection of the system into sub-systems. Note any signs of system interoperability constraints (or suspicion of such).

Technical View:

Create or locate concrete standards that were or will be adhered to during the life span of the system(s). This includes any suggested or mandatory DOD-STD-2167A, IEEE/EIA 12207, or LISI requirements that are usually project and goal dependent. Standards are both implicit and explicit; and, therefore, all standards should be documented thoroughly since all are designated as mission essential to achieve goals. Note any signs of system interoperability constraints (or suspicion of such).

Overview:

Axis I, the “Conception” stage, is named appropriately since certain benchmarks must be achieved in this process of the System Interoperability Process Model. These benchmarks include Operational, Systems, and Technical Views creating a solid basis for the “history” of legacy systems and a complete picture of the desired objective to achieve. Other benchmarks include answering questions such as the type of data transfer, at what rate (Technical), to which systems (Operational), through what paths (Systems). In other words, a conceptual idea of how the three Architecture Views relate for each unique project is essential for achieving system interoperability in advanced cycles of Axes. Discovering

disjointed, conflicting views, referred to as a “collision,” is highly likely at this Axis (i.e. diagrams that don’t match the operational view), which will cause exponential damage in system interoperability as the process model continues. In most cases, there can never be enough planning to ensure that system interoperability will be achieved in later stages.

### 8.3.7.2 AXIS II: INCUBATION

*Deming: Act*

*Boehm: Evaluate Alternatives, Identify, Resolve Risks*

Systems Engineering Process Activities: [2. Architectural Design](#)  
[3. Requirements Partitioning](#)

C4ISR Architecture Views: [Operational](#), [Systems](#), [Technical](#)

Systems Engineering Process Activity: [2. Architectural Design](#)

Operational View:

Sub-systems are categorized into sets of independent sub-systems based on the records (information) in Axis I.

Systems View:

Sub-systems are categorized into sets of independent sub-systems based on the diagrams in Axis I.

Technical View:

Determine standards that apply to “developmental” stages (Specifically IEEE/EIA 12207: Developmental Processes) and other applicable standards.

Sub-Axis Overview:

Decomposing a system into sub-systems involves the Operational View and Systems View to have already achieved a successful benchmark at Axis I. If this benchmark was achieved, then fewer collisions should occur in division of sub-systems. In other words, if there is agreement between the Operational View and the Systems View in conceptual analysis in the Axis I process, then fewer “collisions” should occur in Axis II. However, if this benchmark was unsuccessfully achieved, many collisions will occur thus signaling a regression to the “Conception” stage to rectify the differences. These differences will again increase exponentially as the views become more and more disjunctive.

Systems Engineering Process Activity: [3. Requirements Partitioning](#)

Technical View:

The Operational and Systems Views have already decomposed the system into sub-systems with an adequate amount of agreement, which is always project

dependent; therefore, the Technical View at this stage must determine the hardware or software requirements based upon the divisions previously set. Again, standards that apply to “developmental” stages and other applicable standards are noted in this stage.

Overview:

Axis II, the “Incubation” stage, is named appropriately since no “actions” are taken at this stage of process in the System Interoperability Process Model. Benchmarks include the Operational View and the Systems View categorizing independent sub-systems based on the agreements made in Axis I. The Technical View must take these agreements and assign requirements based upon the software and hardware divisions previously set. Also, if a new system is implemented then collisions should occur less in determining divisions of sub-systems at this stage.

### 8.3.7.3 AXIS III: HATCHING

*Deming: Do*

*Boehm: Develop, Verify Next Level Product*

Systems Engineering Process Activities:

4. Software Requirements Engineering

5. Sub-system Development

C4ISR Architecture Views: **Operational, Systems, Technical**

Systems Engineering Process Activity: 4. Software Requirements Engineering

Technical View:

High-level software requirements are translated into a detailed, specific set for the software components of the system.

Sub-Axis Overview:

This sub-Axis stage could have been included in the “Incubation” stage; however, there is a logical justification for keeping it in an advanced stage. Regressions to prior stages, at some point in the System Interoperability Model, are highly likely due to “incomplete” stages, which later reveal that all avenues of approach were not explored or not explored thoroughly. Regression to the previous stage, “Incubation,” would involve reviewing hardware and software requirements of sub-systems. In the “Hatching” stage, an exploratory viewpoint is taken in the sense that regression stays in one stage of process and can be reviewed as needed, without involving both hardware and software requirements from a previous process stage.

Systems Engineering Process Activity: 5. Sub-system Development

**Operational View:**

Independent sub-systems are designed and implemented based upon categorization in Axis II.

**Systems View:**

Independent sub-systems are designed and implemented based upon categorization in Axis II.

**Overview:**

The Operational View and the Systems View designed and implemented independent sub-systems based on the agreements made in Axis II. If this benchmark was achieved, then fewer collisions should occur in both design and implementation. However, if this benchmark was unsuccessfully achieved, many collisions will occur thus signaling a regression to the “Incubation” stage to rectify the differences. However, regression first begins at the sub-Axis in this stage since technical requirements for software components may be suspect. Again, these differences will increase exponentially as the views become more and more disjunctive.

**8.3.7.4 AXIS IV: MATURATION**

*Deming: Study/Check*

*Boehm: Plan Next Phases*

Systems Engineering Process Activities: [6. System Integration](#)  
[7. System Validation](#)

C4ISR Architecture Views: [Operational](#), [Systems](#), [Technical](#)

Systems Engineering Process Activity: [6. System Integration](#)

**Operational View:**

Independent sub-systems both software and hardware, based on design and implementation in Axis III, are integrated into a system.

**Systems View:**

Independent sub-systems both software and hardware, based on design and implementation in Axis III, are integrated into a system.

**Sub-Axis Overview:**

The Operational View and the Systems View designed and implemented independent sub-systems in Axis III. If this benchmark was achieved, then fewer collisions should occur in integrating the system(s). However, if this benchmark was unsuccessfully achieved, many collisions will occur thus signaling a regression to the “Hatching” stage to rectify the differences. Again, these

differences will increase exponentially as the views become more and more disjunctive.

Systems Engineering Process Activity: [7. System Validation](#)

Technical View:

The integrated system(s) is validated against collective, specified requirements recorded throughout the System Interoperability Process Model cycle. If the requirements are invalid, then the cycle must continue until these requirements are fulfilled.

Overview:

The Operational View and the Systems View designed and implemented independent sub-systems in Axis III, which in turn preempted the integration of the system(s) in Axis IV. In this same stage, the Technical View validated comprehensive requirements and determined whether or not the System Interoperability Process Model should implement further cycles to achieve valid results. Therefore, further iterations of the System Interoperability Process Model are based on specific adherence to achieving valid requirements set from the “Conception” stage onward. Furthermore, having a process model depend on valid requirements to provide insight into further iterations is a common engineering practice.

## **8.4 CONCLUSIONS**

In this chapter we have proposed an interoperability process model based on best commercial practices, current state-of-the-art software engineering research as well as US Department of Defense efforts. We have started with Professor Barry Boehm’s Spiral Model, added elements from Deming’s work and finally added checkpoints from the DOD C4ISR Architecture Framework.

Process is not a panacea. The purpose of the proposed System Interoperability Process Model is to provide engineers in the service system acquisition commands a means to develop interoperable systems given a set of valid requirements.

## 8.5 REFERENCES

[American 00] *The American Heritage College Dictionary*, Third Edition, Houghton Mifflin Company, Boston, 2000.

[Bass, Clements and Kazman 98] Bass, L., Clements, P. and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, Reading, Mass., 1998, p. 392.

[Boehm 88] Barry Boehm, *A Spiral Model of Software Development and Enhancement*, Computer, May 1988, pp. 61-72.

[Boehm 94] Barry Boehm and P. Bose, *A Collaborative Spiral Software Process Model Based on Theory W.*, Proceedings, ICSP 3, IEEE, Reston, VA, Oct. 1994.

[C4ISR 98] Architectures Working Group, *C4ISR Architecture Working Group Final Report*, 14 April 1998.

[Chatfield 02] Chatfield, Jennifer, Enyeart, Cy, Ficks, Wayne., *New Architecture Directions*, [http://www.mitre.org/pubs/edge/january\\_98/fifth.htm](http://www.mitre.org/pubs/edge/january_98/fifth.htm), January 1998.

[Croll 00] Paul R. Croll, *The current State of National and International Standards for the Software Engineering of Mission Critical Systems*, NDIA SE&S Conference, Session 4B, Standards and Best Practices, 25 October 2000.

[Deal 97] Deal, J.C., Hamilton, J.A., Jr., Caudle, J., *Unknown Lands and Uncharted Waters, The Army Enterprise Architecture*, 3rd International Symposium on Command and Control Research and Technology, June 17 - 20, 1997, National Defense University, Fort McNair, Washington, D.C., pp 426 - 449.

[Deming 00] Edwards Deming, *The New Economics for Industry, Government, Education*, MIT Press, Cambridge, Massachusetts, 2000.

[DOD-STD-2167A] *Military Standard: Defense System Software Development*, Department of Defense, Washington, D.C., 29 February 1988.

[Faughn 01] Anthony W. Faughn, *Interoperability: Is it Achievable?*, Program on Information Resources Policy, Harvard University, SEPT 2001.

[Hamilton 98] John A. Hamilton, *Achieving HLA Compliance VIA the Joint Technical Architecture – Army*, Summer Computer Simulation Conference, July 19 - 22, 1998, Reno, Nev., pp 509 - 513.

[Hamilton 98a] Hamilton, J.A., Jr., Deal, J.C., *Software Interoperability and Distributed Simulation*, 1998 Advanced Simulation Technologies Conference April 5 - 9, 1998, Boston, Mass., pp 151 - 156.

[Hamilton 2000] John A. Hamilton, *Interoperability Education Initiative: Joint Technical Architecture Tutorial*, Book 1: Architectures and Data Issues, Presentation 2, AFIT School of Systems and Logistics, 2000.

[Hamilton, Murtagh and Deal, 99] John A. Hamilton, Jeanne L. Murtagh, and John C. Deal, *A Basis for Joint Interoperability*, 1999 Command & Control Research & Technology Symposium, US Naval War College, 29 June – 1 July

[IEEE 90] IEEE Computer Society Coordinating Committee, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standards Board, New York, 1990.

[Koehler & Pankowski 1996] Jerry Koehler and Joseph Pankowski, *Continual Improvement in Government Tools and Methods*, St. Lucie Press, Delray Beach, Florida, 1996.

[Kotonya & Sommerville 98] Gerald Kotonya and Ian Sommerville, *Requirements Engineering*, John Wiley & Sons, New York, 1997.

[Levels 02] *Levels of Information Systems Interoperability LISI*, <http://www.doi.army.mil/doi/LISI/overview/LISI%20White%20Paper.htm>

[LISI 98] *Levels of Information Systems Interoperability, C4ISR Architecture Framework Version 2.0*, C4ISR AWG Architectures Working Group, 30 MAR 1998.

[Wells 98] Jim Wells, *Overview of IEEE/EIA 12207: Standard for Information Technology*, Software Engineering Process Office at SPAWAR Systems Center San Diego, 1998.

## 8.6 FIGURES, TABLES AND APPENDICES

Figure 1 Wells 1998

Figure 2

[http://www.nca.or.kr/homepage/s\\_standard.nsf/78c7dc6c068f9138c92569980026b32b/262fb5d5cd827ff0c9256a45001216c0/\\$FILE/LISI\(010227\).ppt](http://www.nca.or.kr/homepage/s_standard.nsf/78c7dc6c068f9138c92569980026b32b/262fb5d5cd827ff0c9256a45001216c0/$FILE/LISI(010227).ppt)

Figure 3 [http://www.mitre.org/support/swec/html/01\\_sowellfinal/sld033.htm](http://www.mitre.org/support/swec/html/01_sowellfinal/sld033.htm)

Figure 4 Chatfield 1998

Figure 5 Boehm 1988

Figure 6 System Interoperability Process Model (backbone)

Table 1 Deming's PDSA Cycle Steps: Improvement of a Product or a Process

Figure 7 Deming 2000

Figure 8 Koehler and Pankowski 1996

Table 2 Koehler and Pankowski's PDCA Cycle Steps

Figure 9 System Interoperability Process Model (Deming Cycle Addition)

Table 3 The Systems Engineering Process Activities

Figure 10 Kotonya and Sommerville 1998

Figure 11 System Interoperability Process Model

(System Engineering Process Activities Addition)

Figure 12 Hamilton 2000

Figure 13 System Interoperability Process Model (C4ISR Addition)

### 8.6.1 APPENDICES

**Appendix A:** *Military Standard: Defense System Software Development*. 1988. Department of Defense, Washington, D.C.

**Appendix B:** Wells, J. 1998. *Overview of IEEE/EIA 12207: Standard for Information Technology*. Software Engineering Process Office at SPAWAR Systems Center San Diego.

**Appendix C:** Wells, J. 1998. *Overview of IEEE/EIA 12207: Standard for Information Technology*. Software Engineering Process Office at SPAWAR Systems Center San Diego.

**Appendix D:** C4ISR AWG Architectures Working Group. 1998. Levels of Information Systems Interoperability, C4ISR Architecture Framework Version 2.0.

**Appendix E:** C4ISR AWG Architectures Working Group. 1998. Levels of Information Systems Interoperability, C4ISR Architecture Framework Version 2.0.

**8.6.1.1 APPENDIX A**

## 3. DEFINITIONS

3.8 Computer Software Component (CSC). A distinct part of a computer software configuration item (CSCI). CSCs may be further decomposed into other CSCs and Computer Software Units (CSUS).

3.9 Computer Software Configuration Item (CSCI). A configuration item for computer software.

3.11 Computer Software Unit (CSU). An element specified in the design of a Computer Software Component (CSC) that is separately testable.

**8.6.1.2 APPENDIX B**

<b>The Four Organizational Life Cycle Processes</b>	
Management	Defines the basic activities of the management, including project management, related to the execution of a life cycle process.
Infrastructure	Defines the basic activities for establishing the underlying structure of a life cycle process.
Improvement	Defines the basic activities that an organization (that is, acquirer, supplier, developer, operator, maintainer, or the manager of another process) performs for establishing, measuring, controlling, and improving its life cycle process.
Training	Defines the activities for providing adequately trained personnel.

<b>The Five Primary Processes</b>	
Acquisition	Defines the activities of the <i>acquirer</i> , the organization that acquires a system, software product, or software service.
Supply	Defines the activities of the <i>supplier</i> , the organization that provides the system, software product or software service to the acquirer.
Development	Defines the activities of the <i>developer</i> , the organization that defines and develops the software product.
Operation	Defines the activities of the <i>operator</i> , the organization that provides the service of operating a computer system in its live environment for its users.
Maintenance	Defines the activities of the <i>maintainer</i> , the organization that provides the service of maintaining the software product; that is, managing modifications to the software product to keep it current and in operational fitness. This process includes the migration and retirement of the software product.

<b>The Eight Supporting Life Cycle Processes</b>	
Documentation	Defines the activities for recording the information produced by a life cycle process.
Configuration management	Defines the configuration management activities.
Quality Assurance	Defines the activities for objectively assuring that the software products and processes are in conformance with their specified requirements and adhere to their established plans. Joint reviews, Audits, Verification and Validation may be used as techniques of Quality Assurance.
Verification	Defines the activities( for the acquirer, the supplier, or an independent party) for verifying the software products and services in varying depth depending on the software project.
Validation	Defines the activities (for the acquirer, the supplier, or an independent party) for validating the software products of the software project.
Joint Review	Defines the activities for evaluating the status and products of an activity. This process may be employed by any two parties, where one party (reviewing party) reviews another party (reviewed party) in a joint forum.
Audit	Defines the activities for determining compliance with the requirements, plans, and contract. This process may be employed by any two parties, when one party (auditing party) audits the software products or activities of another party (audited party).
Problem resolution	Defines a process for analyzing and removing the problems (including nonconformances), whatever their nature or source, that are discovered during the execution of development, operation, maintenance, or other processes.

## 8.6.1.3 APPENDIX C

## Overview of IEEE/EIA 12207: Standard for Information Technology

(12207.0 Clause 5, continued)

## 5.3 The Development Process

Activity	Tasks (paraphrased)	12207.1 Information Item guidelines	
5.3.1 Process Implementation	.1 Define software life cycle model .2 Document and control outputs .3 Select and use standards, tools, languages .4 Document development plans .5 Deliver all needed products	-- -- -- Plan, Desc --	-- -- -- 6.5 DPP, 6.17 SDSD --
5.3.2 System requirements analysis	.1 Specify system requirements .2 Evaluate requirements against criteria	Specification Spec, Record	6.26 SRS 6.26 SRS, 6.6 SRER
5.3.3 System architectural design	.1 Establish top-level architecture .2 Evaluate architecture against criteria	Description Desc, Record	6.25 SARAD 6.25 SARAD, 6.6 SAER
5.3.4 Software requirements analysis	.1 Document software requirements .2 Evaluate requirements against criteria .3 Conduct joint reviews iaw 6.6	Desc Desc, Record --	6.22 SRD, 6.30 UDD 6.22 SRD, 6.6 SRER --
5.3.5 Software architectural design	.1 Transform requirements into architecture .2 Document top-level design for interfaces .3 Document top-level design for database .4 Document preliminary user documentation .5 Document preliminary test requirements .6 Evaluate architecture against criteria .7 Conduct joint reviews iaw 6.6	Description Description Description Description Plan Desc, Record --	6.12 SAD 6.19 SIDD 6.4 DBDD 6.30 UDD 6.27 T/VP 6.12 SAD, 6.6 SAER --
5.3.6 Software detailed design	.1 Document design for each component .2 Document design for interfaces .3 Document design for database .4 Update user documentation .5 Document unit test requirements .6 Update integration test requirements .7 Evaluate detailed design against criteria .8 Conduct joint reviews iaw 6.6	Description Description Description Description Plan Plan Rec, Desc --	6.16 SDD 6.19 SIDD 6.4 DBDD 6.30 UDD 6.27T/VP 6.27T/VP 6.6 DDER, 6.16 SDD --
5.3.7 Software coding and testing	.1 Document each unit, database and tests .2 Conduct and document unit testing .3 Update user documentation .4 Update integration test requirements .5 Evaluate code and test results	Desc, Rec, Proc Report Description Plan Rec, Plan	6.4 DBDD, 6.24 SCR, 6.28 T/VP 6.29 T/VRR 6.30 UDD 6.27T/VP 6.7 EOCR, 6.6 SCTRE, 6.24SCR, 6.27T/VP
5.3.8 Software integration	.1 Document integration plans .2 Conduct and document integration tests .3 Update user documentation .4 Document qualification tests .5 Evaluate plans and tests against criteria .6 conduct joint reviews iaw 6.6	Plan, Proc Report Description Proc Proc, Desc Record, Plan	6.18 SIP, 6.28 T/VP 6.29 T/VRR 6.30 UDD 6.28 T/VP 6.28 T/VP, 6.30 UDD 6.6 SIER, 6.18 SIP
5.3.9 Software qualification testing	.1 Conduct and document qualification testing .2 Update user documentation .3 Evaluate tests against criteria .4 Support audits iaw 6.7 .5 Prepare product for next phase	Report Description Record -- Record	6.29 T/VRR 6.30 UDD 6.6 SIER -- 6.24 SCR
5.3.10 System integration	.1 Integrate software with hardware & others .2 Document integration tests .3 Evaluate integrated system against criteria	Report Procedure Record	6.29 T/VRR 6.28 T/VP 6.6 SQTER
5.3.11 System qualification testing	.1 Conduct and document qualification tests .2 Evaluate system against criteria .3 Support audits iaw 6.7 .4 Prepare product for installation	Report Record -- Record	6.29 T/VRR 6.6 SER -- 6.24 SCR
5.3.12 Software installation	.1 Plan installation in target environment .2 Install software iaw plan	-- --	-- --
5.3.13 Software acceptance support	.1 Support acquirer's acceptance tests .2 Deliver product per contract .3 Provide training per contract	Report Record --	6.29 T/VRR 6.24 SCR --

**8.6.1.4 APPENDIX D**

<i>Information Exchange</i>	<i>Level</i>	<i>Computing Environment</i>
Cross-domain information and application sharing Advance collaboration (Interactive COP update, event-triggered global database update)	<b>4: Enterprise</b> Interactive manipulation Shared data and application	Global Information Space USPACOM FEMA NCA ROK HQ
Shared databases Sophisticated collaboration (Common Operational Picture)	<b>3: Domain</b> Shared data “Separate” applications	NIDR, Common Displays, Shared Applications & Data
Heterogeneous product exchange Basic collaboration (Annotated imagery, map with overlays)	<b>2: Functional</b> Minimal common functions Separate data and applications	HTTP, NITF
Homogeneous product exchange (FM voice, tactical data links, text files, messages, e-mail)	<b>1: Connected</b> Electronic connection Separate data and applications	Telnet, FTP, E-mail, Chatter
Manual Gateway (diskette, tape, hard copy exchange)	<b>0: Isolated</b> Non-Connected	

### **8.6.1.5 APPENDIX E**

#### **Operational Architecture View**

*Definition*

The operational architecture view is a description of the tasks and activities, operational elements, and information flows required to accomplish or support a military operation

*Primary Purpose*

Define operational elements, activities and task, and information exchange requirements

*Tenants*

- Operational architectures incorporate doctrine and assigned tasks and activities
  
- Activities and information-exchange requirements may cross organizational boundaries
  
- Operational architectures are not generally systems-dependent
  
- Generic activity descriptions are not based on an organizational model or force structure
  
- Operational architectures should clearly identify the time phase(s) covered

#### **Systems Architecture View**

*Definition*

The systems architecture view is a description, including graphics, of systems and interconnections providing for, or supporting, warfighting functions

*Primary Purpose*

Enable or facilitate operational tasks and activities through the application of physical resources

*Tenants*

- Systems architectures map systems with their associated platforms, functions, and characteristics back to the operational architecture
  
- Systems architectures identify systems interfaces and define the connectivities between systems

- System architectures define systems constraints and bounds of system performance behavior
- Systems architectures are technology-dependent, show how multiple systems within a subject area link and interoperate, and may describe the internals of particular systems
- Systems architectures can support multiple organizations and missions
- Systems architectures should clearly identify the time phase(s) covered
- Systems architectures are based upon and constrained by technical architectures

### **Technical Architecture View**

#### *Definition*

The technical architecture view is the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements

#### *Primary Purpose*

Define the set of standards and rules that govern system implementation and system operation

#### *Tenants*

- Technical architecture views are based on associations between operational requirements and their supporting systems, enabling technologies, and appropriate interoperability criteria
- A technical architecture profile is constructed from an enterprise-wide set of standards and design rules for specific standards contained in the Joint Technical Architecture and other applicable standards documents
- The technical architecture standards and criteria should reflect multiple information system implementation paradigms
- Technical architecture profiles account for the requirements of multiplatform and network interconnections among all systems that produce, use, or exchange information electronically for a specifically bounded architecture configuration
- Technical architectures must accommodate new technology, evolving standards, and the phasing out of old technology

-Technical architectures should be driven by commercial standards and direction