Software Engineering Institute

Beyond Technology Readiness Levels for Software: U.S. Army Workshop Report

Stephen Blanchette, Jr.
Cecilia Albert
Suzanne Garcia-Miller

December 2010

TECHNICAL REPORT
CMU/SEI-2010-TR-044
ESC-TR-2010-109

Acquisition Support Program
Research, Technology, and System Solutions Program

Unlimited distribution subject to the copyright.

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Copyright 2010 Carnegie Mellon University.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Internal use. Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

External use. This document may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors are grateful to Linda Levine, Michael Bandor, and Linda Northrop of the SEI, whose detailed reviews and insightful comments improved the quality of this report.

Executive Summary

On August 10–11, 2010, the Carnegie Mellon® Software Engineering Institute (SEI) facilitated an Army workshop, Beyond Technology Readiness Levels for Software, at Picatinny Arsenal in New Jersey. The workshop was part of the ongoing Army Strategic Software Improvement Program (ASSIP) under the oversight of the Office of the Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA(ALT)).

Technology readiness levels (TRLs) are standard measures used to determine whether a technology is sufficiently mature to be incorporated into a system. Software readiness has become an important issue for several reasons. Though software often makes up only a small percentage of overall program budgets, software issues are often an overriding constraint to delivery of Army systems to the field. For embedded systems, software is the "nervous system" that enables overall system functionality. Yet, software still is not routinely considered early in program formulation.

Further, experience with a number of technology readiness assessments (TRAs) of systems involving software has shown several shortcomings in the TRA process or its application, including

- risk of distraction by "glamorous" technologies, causing mundane but time-tested technology to be overlooked
- lack of distinction among software types (newly developed software, reused software, and commercial-off-the-shelf software)
- loss of experience and knowledge base when moving from a laboratory environment to a "relevant" environment, negatively affecting readiness scores
- inconsistencies between the TRA process and the Department of Defense Instruction (DoDI) 5000.02 acquisition life cycle
- inconsistent definition of what represents a new software technology
- incomplete consideration of life-cycle maintenance and support
- external influences on technology choices that may cause an implied critical technology element (CTE)—that is, one that does not meet the definition of CTE but nevertheless has the same effect on a program
- lack of guidance for handling technologies that are started in one increment of the software but finished in a later increment

Many of the noted problems with the TRA process and its application are rooted in the fact that the DoD's software TRLs were derived from the original set of hardware TRLs rather than being developed independently from a software perspective. The question, then, is what "real" information do program managers need to understand the risk of moving forward past a major milestone when software is involved in a complex acquisition?

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

The Army's own systemic analysis of software-intensive acquisition programs shows that failings (such as poor oversight and flawed acquisition strategy) during development cause particular difficulty for weapon systems with embedded software as they pass initial deployment and must be maintained. Often, it is the Army's software engineering centers that must attempt to compensate for these failings.

Workshop participants agreed that the knowledge represented by a readiness indicator, at any level, should provide assurance that the software is mature enough to proceed to the next acquisition phase. Such knowledge has many facets. Within the time constraints of the workshop, attendees focused on defining a candidate set of five "software readiness levels" (SRLs)—concept, architecture, design/prototype, developed, and ready for deployment—and mapping those SRLs to the acquisition life cycle. It turns out that the mapping is highly dependent on the type of system being developed. Mapping SRLs to weapon systems with embedded software is more straightforward than mapping to systems in which the software effectively is the system, such as enterprise resource planning (ERP) systems.

It is interesting to note that the SRLs that occur earlier in development (concept, architecture, and design/prototype) tended to emphasize product insight while the later SRLs tended to have more of a process focus. In addition, participants noted that the proposed SRLs did not treat satisfactorily the programmatic aspects of software development, post-deployment issues such as software sustainment and evolution, and systems of systems.

The problem of characterizing software readiness is not a simple one, and many questions must still be answered. Nearly 70% of attendees expressed interest in continuing discussions on this topic.

Abstract

The Carnegie Mellon® Software Engineering Institute (SEI) facilitated an Army workshop, Beyond Technology Readiness Levels for Software, on August 10-11, 2010. The workshop, part of the ongoing Army Strategic Software Improvement Program (ASSIP), was an attempt to develop an Army perspective on the "right" software information to gather and analyze at significant program decision points (especially Milestones A, B, and C) to determine readiness to proceed to the next acquisition phase. This report synthesizes the workshop presentations, discussions, and outcomes.

1 Introduction

On August 10-11, 2010, the Carnegie Mellon® Software Engineering Institute (SEI) facilitated an Army workshop entitled Beyond Technology Readiness Levels for Software at Picatinny Arsenal in New Jersey. The workshop was part of the ongoing Army Strategic Software Improvement Program (ASSIP) under the oversight of the Office of the Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA(ALT)).

1.1 About this Report

This document summarizes the presentations and discussions from the workshop. This section introduces the problems the services have encountered using DoD Software TRLs and lists the workshop participants. Section 2 reviews the glossary that attendees used to achieve a common understanding during discussions. Section 3 summarizes presentations made at the beginning of the workshop. Section 4 summarizes the discussions and findings of each of the working group sessions. Section 5 presents conclusions and identifies potential future work.

The appendices include a list of acronyms and the current DoD TRL definitions. References to cited works may be found at the end of the report.

1.2 DoD Hardware TRLs and Software TRLs

At the System and Software Technology Conference (SSTC) in April 2010, a panel drawn from the services, the Office of the Secretary of Defense (OSD), and the SEI presented summaries of studies and activities conducted over the past few years on the application of Department of Defense (DoD) technology readiness levels (TRLs) for software (see Table 1). In general, those studies concluded that the use of the software TRLs is problematic, to the point that many who have participated in technology readiness assessments (TRAs) and subsequent program decisions believe that TRLs are an incorrect basis for evaluating either software technologies or software products throughout the course of an acquisition.

Table 1: DoD Technology Readiness Levels[1]

Definitions		
Level	**Hardware**	**Software**
1	Basic principles observed and reported	Basic principles observed and reported
2	Technology concept and/or application formulated	Technology concept and/or application formulated
3	Analytical and experimental critical function and/or characteristic proof of concept	Analytical and experimental critical function and/or characteristic proof of concept
4	Component and/or breadboard validation in a laboratory environment	Module and/or subsystem validation in a laboratory environment (i.e., software prototype development environment)

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

[1] See Appendix B for additional details regarding the hardware and software TRLs.

Definitions		
Level	**Hardware**	**Software**
5	Component and/or breadboard validation in a relevant environment	Module and/or subsystem validation in a relevant environment
6	System/subsystem model or prototype demonstration in a relevant environment	Module and/or subsystem validation in a relevant end-to-end environment
7	System prototype demonstration in an operational environment	System prototype demonstration in an operational high-fidelity environment
8	Actual system completed and qualified through test and demonstration	Actual system completed and mission qualified through test and demonstration in an operational environment
9	Actual system proven through successful mission operations	Actual system proven through successful mission-proven operational capabilities

To date, efforts by the services to interpret the DoD software TRLs have been constrained by the requirement to retain the basic definitions when creating guidance. Though marginally useful, these efforts have only confirmed for the participants the futility of continuing to base readiness decisions for software aspects of systems on the DoD software TRLs.

This workshop focused primarily on Army software-related interests but was informed by ongoing efforts in other DoD areas. Participants were asked to ignore current constraints and consider: "What is the right information to gather and analyze related to software at significant program decision points (especially Milestones A, B, and C) to determine readiness to proceed into the next acquisition phase?"

The desired outcome of the workshop was to develop an Army strategic direction for tackling the shortcomings in the readiness-level concept for software. The Army hoped to share this information with the other services and to collaborate on an evaluation approach that provides DoD decision makers with relevant software-related information at key acquisition decision points.

1.3 Workshop Participants

Suzanne Garcia-Miller of the SEI led the workshop, assisted by other SEI technical professionals. Workshop participants represented the broad acquisition community, including the office of the assistant secretary of the Army for acquisition, logistics, and technology, Army program executive offices (PEOs), Army and Joint project management offices (PMOs), Army software engineering centers and directorates, the Defense Contract Management Agency (DCMA), the Army's Central Technical Support Facility (CTSF), and the MITRE Corporation. Overall, 38 individuals participated in the workshop: 30 Army participants, 2 joint PEO/PMO participants, 4 SEI participants, 1 MITRE participant, and 1 DCMA participant.

Table 2: Workshop Participants (listed alphabetically by last name)

Attendee	Organization
Cecilia Albert	SEI
Stephen Blanchette, Jr.	SEI
Tom Blenk	Armament Research, Development, and Engineering Center (ARDEC)
Michael Bonastia	Project Manager (PM) Close Combat Systems/Armament Software Engineering Center (SEC)
Michael Brown	Armament SEC
Eric Byrd	PEO Soldier
Terry Carlson	PEO Aviation
Karen Davis	PEO Ammunition
Monica Farah-Stapleton	Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA(ALT))
Cathy Fitch	PEO Ammunition
Mindy Gabbert	Communications-Electronics Command (CECOM) SEC
Tim Ganguly	Joint Project Manager Guardian
Suzanne Garcia-Miller	SEI
Tim Green	Communications-Electronics Research, Development and Engineering Center (CERDEC) Software Engineering Directorate (SED)
Michael Gully	Aviation and Missile Research, Development, and Engineering Center (AMRDEC) SED
Scott Hamma	PEO Ground Combat Systems
Mike Herrmann	PEO Enterprise Information Systems
Chris Jais	ASA(ALT)
Valarie James	Precision Fires Rocket and Missile Systems/PEO Missiles & Space
Ken Kragh	CECOM SEC
Alex Leung	PEO Ammunition
Angela Llamas-Butler	SEI
Robert Loesh	AMRDEC SED
Steve Lubash	Armament SEC
Glenda Mendiola	PEO Intelligence, Electronic Warfare & Sensors
Donna Merriman	Defense Contract Management Agency (DCMA)/Software Engineering Acquisition Management
Bryce Meyer	PEO Integration/SEI
Dana Miles	CECOM SEC
John F. Miller	MITRE
Larry Osiecki	ARDEC SEC
Richard Payne	ARDEC Quality Engineering & System Assurance (QESA)
Barbara Pemberton	PEO Simulation, Training, and Instrumentation
Josh Pressnell	Joint PEO Chemical and Biological Defense
Tracey Roberts	Central Technical Support Facility (CTSF)

Dhaval Shah	PEO Ammunition
Gagan Singh	ARDEC QESA
Gari-Lynn Smith	U.S. Army
Jack VanKirk	PEO Aviation

2 A Common Glossary

Experience has shown that different interpretations of the same words often lead to misunderstandings. Too often, people part company believing they have an agreement, only to find out later that interpretational differences have resulted in two (or more) unique perspectives, neither of which is agreeable to the other party. With that in mind, Suzanne Miller presented a proposed glossary of the terms that were important to the discussions in this workshop. The terms are presented below in the order that they were discussed.

Development
Systematic application of knowledge toward the production of useful materials, devices, and systems or methods [Meade 2003] (includes commercial-off-the-shelf, or COTS, configuration)

Maintenance
Corrective and preventive activities performed on fielded systems (and systems of systems) meant to prolong their useful life

(Pre-) Planned Product Improvement (vs. "Evolution")
Enhancements to an existing system meant to improve its ability to meet changing mission needs

System
A functionally, physically, and/or behaviorally related group of regularly interacting or interdependent elements; that group of elements forming a unified whole [CJCS 2001, CJCS 2006]

Subsystem
A system that is part of one or more larger system [Wikipedia 2010b]

System of Systems
A set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities [DoD 2010]

Capability
Ability to achieve a desired effect under specified standards and conditions through combinations of ways and means to perform a set of tasks [CJCS 2009]

Real Time (as in Real-Time System)
Hardware and software systems that are subject to a "real-time constraint"—i.e., operational deadlines from event to system response [Wikipedia 2010a]

Emergence (vs. Synergy)
The presence of novel (not just unanticipated) behaviors in a composition of multiple systems or system elements

Agility

Possessing both the qualities of flexibility (designed-in ability to change in anticipated ways) and adaptability (ability to be changed in unanticipated ways)

During the course of discussion, the group identified an additional term:

Deployment

Delivery of a capability to its intended end user

At the conclusion of the workshop, the definition of several terms proved elusive with respect to technology readiness (that is, the group was unable to agree upon definitions):

- readiness
- software (i.e., it is more than just "code")
- software maturity
- software readiness
- software technology

The group noted that, ironically, even the *Technology Readiness Assessment Deskbook* published by the DoD does not contain a definition of *technology* [DoD 2009].

3 Presentation Summaries

During the first day, two background presentations were given by experts in software development. This section summarizes each of those presentations.

3.1 Initial Thoughts—Cecilia Albert, SEI

Cecilia Albert, Chief Engineer for Army Programs at the Software Engineering Institute, presented some initial thoughts on software readiness.

3.1.1 Why is Software Readiness an Issue?

There are several reasons why software readiness has become an issue. Though software is often a small percentage of overall program budgets, it is often an overriding constraint to delivery of Army systems to the field. For embedded systems, software is the "nervous system" of the overall system, enabling much of the system's functionality. Yet, software still is not routinely considered early in program formulation. For example, software is a critical element of the radar systems of modern aircraft. Figure 1 illustrates a traditional view of software design evolution. Important software decisions are made at the very earliest stages of the program, but software typically is not considered until the point of software requirements development.

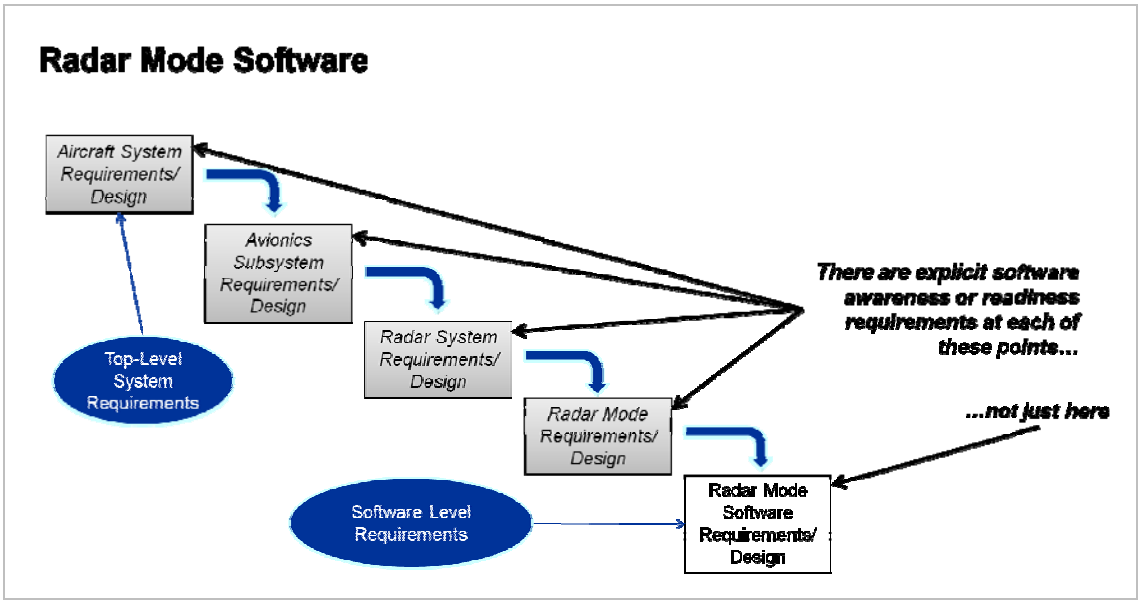

Figure 1: A Traditional View of Software Design Evolution

The recognition of the impact of systems of systems (SoS) on engineering, development, and acquisition also contributes to the need to think about software readiness. The advent of ubiquitous network interaction, both within and among systems, makes it infeasible to isolate software and defer dealing with its issues. Virtually no DoD system operates in isolation any more, and the situation is especially difficult when systems that previously had no external interactions must cooperate to achieve their respective objectives. The mechanism for achieving

such interoperation is invariably software—software that likely had not been envisioned as part of the baseline program.

In essence, the "ecosystem" of acquisition is changing, and PMOs must find ways to cope with the changes. They must find the information needed to address the software aspects of their programs that will result in systems that evolve as part of an SoS ecosystem. Unfortunately, current system engineering approaches fall short. Insufficient PMO knowledge and skills and inadequate levels of staff who are both system and software savvy tend to result in a force-fit of modern software approaches into older system engineering models (such as the venerated V-model).

Because software has become such an important part of system development, software readiness is a topic whose time has come.

3.1.2 Shortcomings of Current Approaches to Addressing Readiness for Software

Figure 2 depicts the acquisition life cycle [DoD 2008]. By law, a program must have achieved the equivalent of TRL 6 prior to Milestone B approval [DoD 2009]. However, the prominent mechanism for measuring readiness in DoD development programs—the TRLs—are considered problematic across a wide spectrum of software-intensive programs and program types.

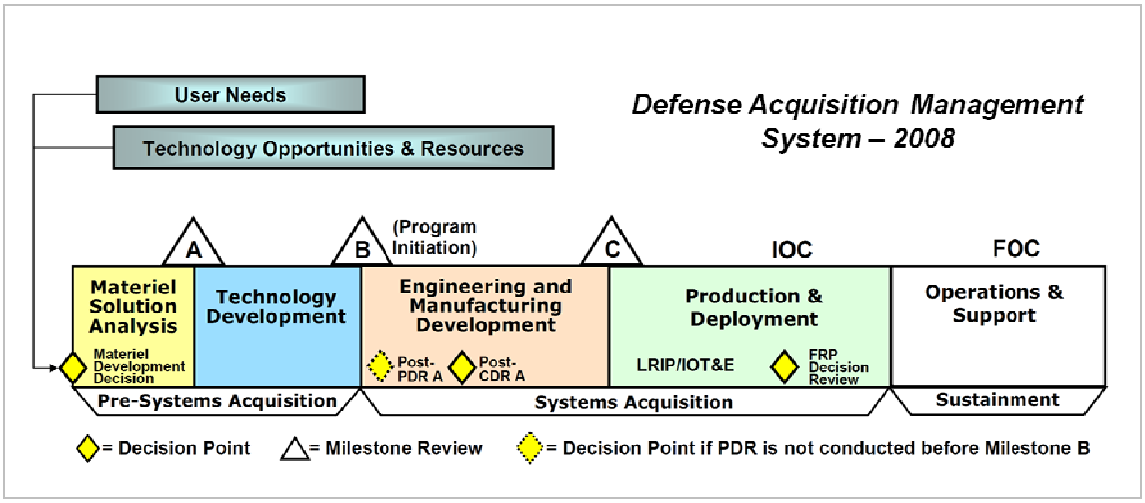

Figure 2: The Defense Acquisition Management Life Cycle

Having participated on a number of TRAs for systems involving software, the SEI has observed several shortcomings in the TRA process and its application [Bandor 2010]:

- **Technology choices: high visibility or "glamorous" vs. mundane.** Programs acquiring systems with highly visible or "glamorous" technologies may inadvertently overlook more mundane, but nonetheless high-impact technologies when identifying critical technology elements (CTEs).[2] Further, software that supports such advanced technologies is rarely

[2] According to the *TRA Deskbook*, "A technology element is "critical" if the system being acquired depends on this technology element to meet operational requirements (within acceptable cost and schedule limits) and if the technology element or its application is either new or novel or in an area that poses major technological risk during detailed design or demonstration" [DoD 2009].

considered, even though it could comprise many millions of undeveloped lines of code (and, therefore, a significant risk).

- **Lack of distinction between software types.** The TRA process does not appear to differentiate between newly developed software, reused software, and COTS software products when applying TRL definitions. The existing TRA descriptions for the measures of the software TRLs seem more appropriate to pre-existing software (e.g., COTS software) and could be difficult to apply in cases where a program has large amounts of software that is a mixture of new and reused code. The meaning of "technology readiness" has different implications for new code versus COTS software.[3] Assessing both types of software with the same readiness definitions is difficult. In addition, strict interpretation of software TRL 5 or higher would exclude all newly developed code that is created post-Milestone B. For software especially, this exclusion seems counter to the intent of assessing readiness.
- **Demonstrations in a "relevant" environment.** The software TRL 6 definition, "demonstration in a relevant environment," does not take into consideration who or which "team" performed the prior demonstrations. Historical trends show prior team integration experience with specific software technologies significantly contributes to reduced programmatic software risk. Yet, current software TRL definitions appear to discount prior use in similar and relevant environments almost as a "point solution." Experience also plays a role; there is potential for significant differences in "technology readiness" with an experienced integration team as opposed to a completely new integration team. In addition, care must be taken in determining the relevance of an environment. For example, technologies demonstrated in a commercial environment do not necessarily map to the anticipated use in a military environment.
- **TRA process inconsistencies with DoDI 5000.02 acquisition life cycle.** The software TRA process appears, in part, to be inconsistent when aligned with certain DoDI 5000.02 program life-cycle model events. The current TRA process presents a "chicken-or-egg" situation for software TRLs when newly developed code is involved. For instance, as mentioned earlier, TRL 6 is required prior to Milestone B approval. TRL 6 requires demonstration in a "relevant environment," which implies that some form of the software code exists within that environment to support a demonstration. Yet, a formal acquisition program does not exist until Milestone B, meaning contractors are not yet under contract and much of the software (particularly for embedded systems) is not yet developed at the point of the TRA.
- **Definition of a new software technology.** The TRL "threshold" for defining what constitutes software technology readiness seems vague enough that consistently applying it to new, reused, and COTS software technologies is subject to interpretation and inconsistency, which risks defeating the purpose of assessing readiness in the first place. One difficulty is that, particularly in cases of embedded software, the software product is viewed as an enabling technology in the eyes of the product engineer. For the software developer, however, that

[3] For instance, architecture of developed code is visible to the program, but often the architecture of COTS software is proprietary and not available to developers other than the COTS vendor. Further, COTS software is rarely a "technology element" for a larger system, where developed code often serves as the enabling technology for a particular hardware-based solution.

product is the result of using software technologies (such as languages and test environments) to create what is needed to support the engineering choices for the larger product. So software engineers tend to look at software technologies as the things that are used to develop the software, not as the software itself. This dichotomy of perspective consistently shows up in technology readiness assessments involving software.

- **Incomplete consideration of life-cycle maintenance/support.** There is not enough consideration in the TRA process of the life-cycle maintenance and support of technologies. COTS software changes and updates are largely driven by corporate market dynamics, not by the PMO. In programs with long development times, a chosen COTS software product may become obsolete and require replacement even before the initial system is fielded, potentially invalidating TRA findings from early in the program.
- **External influences on technology choices causing an implied CTE.** As more software programs are hosted by other organizations, technology choices or upgrades to those new environments (not directed or caused by the originating PMO) may cause an "implied" CTE. For example, say a host changes to a virtual server environment but the original software was designed to run in a more traditional N-tier environment that depends upon physical servers being deployed. The change affects the cost and schedule of the original program (cost savings from not having to buy and deploy additional physical servers), but it also causes a "new relevant environment" to be realized. According to the TRA guidelines, the server virtualization would be considered a CTE. A problem now arises: The original program has neither a virtual server requirement nor a performance requirement to which the new CTE can be traced (part of the TRA process).
- **Technologies begun in one increment and finished in a later increment.** Programs executing an incremental acquisition strategy may choose to initially implement a technology in one increment while completing the full implementation in a subsequent increment. The potential exists for a technology element or CTE to be missed as a result. A question arises: If a technology element or CTE is not missed, should it be evaluated as a partial fulfillment of the function, or later as a full implementation, or both? It also is possible for the technology's status to change between increments, thereby affecting programmatic decisions by the PMO. Such situations are not addressed in the current TRA guidance.

3.1.3 Summary

While technology readiness levels for software provide *some* useful information, they also have a history of problems. Many of these problems stem from the fact that software TRLs were derived from the original set of hardware TRLs. What is needed is a set of software readiness levels that have been derived from the important knowledge points during software development.

The question remains, what is the "real" information needed to understand the risk of moving forward past a major milestone when software is involved? If the answer truly is the current TRL type of information, then nothing more need be done. If the answer is something other than TRLs, then we need to define what that something is. If the answer is a hybrid of TRL and other information, then we need to figure out how to make that work.

The answer should be based on what helps Army program managers involved in complex acquisitions that include software move forward appropriately when approaching a major acquisition milestone.

3.2 Sustaining Software for Embedded Weapons—Larry Osiecki, Armament SEC

Larry Osiecki, Director of the Armament Software Engineering Center at Picatinny Arsenal, presented an overview of the challenges facing the Army in sustaining software for embedded weapon systems.

3.2.1 Challenges

In 2010, the Army conducted a systemic analysis of software-intensive acquisition programs. The study was a fact-based analysis of systemic issues impacting large-scale software-intensive acquisition programs in the DoD, and a benchmark for Army systems. The analysis team compiled a set of issues for Army acquisition leadership. Figure 3 shows the findings clustered by affinity groupings.

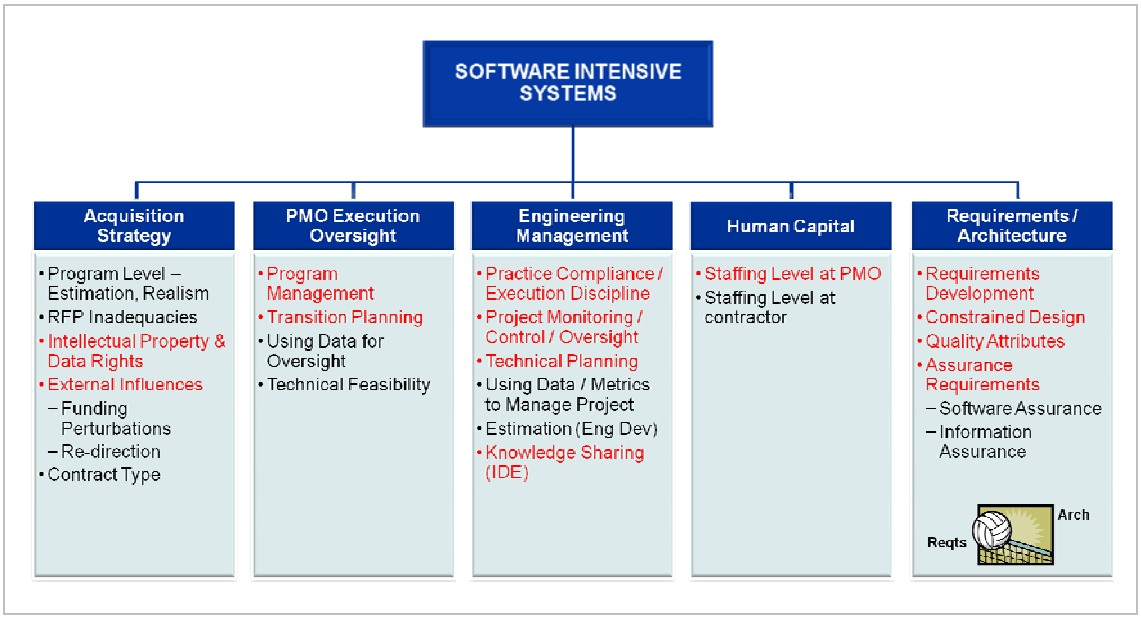

Figure 3: Challenges Facing Software-Intensive Systems Acquisition

Of the noted findings, several (denoted by red text in Figure 3) cause particular difficulty for the Army when weapon systems with embedded software enter the post-deployment phase of the life cycle.

- **Intellectual Property and Data Rights**
  - **Use of proprietary solutions and black box solutions.** Such closed solutions necessitate reverse engineering the software designs and architectures before any code changes can be made.
  - **Limited data rights and intellectual property.** Too often, the Army lacks sufficient rights (or sufficient understanding of the rights it has acquired) to be able to make the software changes needed.

- **Constrained Design/Quality Attributes**
  - **Narrow architectural focus that inhibits system evolution and inflexible designs.** Often, Army systems are developed with an inadequate focus on quality attributes, particularly modifiability. The result is systems that satisfy their original requirements but have very limited ability to expand to meet changing needs.
- **Knowledge Sharing**
  - **Poorly documented software engineering data packages.** One of the keys to effective maintenance is good system documentation—which is especially true for software, because software has no physical manifestation. However, many PMOs limit the amount of documentation produced for a system as a way to save on development costs, which then increases the difficulty and cost of performing software sustainment.
- **Execution Discipline/Assurance Requirements**
  - **Use of COTS.** COTS software is frequently used as a means to reduce development costs, but for embedded weapon software especially, it can complicate sustainment. COTS packages tend to be updated often, more frequently than the update cycle of most weapon systems. Weapon systems must be certified for safety and network security, and some for airworthiness, each time they are changed. Frequent updates are simply unaffordable. Further, COTS software can introduce security vulnerabilities that make some recertifications difficult.
- **Program Management**
  - **Lack of business case analysis for software sustainment.** Software sustainment plans rarely receive adequate attention during overall program planning. As a result, the question of which organization (an Army software center, the development contractor, or a third-party contractor) will perform maintenance remains open until the latter stages of the system development effort. The delay shortchanges sustainment planning and prevents sustainment considerations from influencing the software architecture and design up front.
- **External Influences**
  - **DoD and Army policies geared for information technology.** There is a tendency within the DoD and the Army to treat all software alike, as information technology (IT) assets. However, IT system software—from desktop applications to enterprise resource planning (ERP) systems and even some battlefield command-and-control systems—generally is not subject to the same performance requirements, stringent testing, or certification processes that apply to software for embedded weapon systems. The two types of software have very different upgrade and release timelines. Policies that ignore these differences make it difficult for software sustainment organizations to work effectively and efficiently.
- **Transition Planning**
  - **Inadequate transition planning/resource allocation.** Frequently, the transition to software sustainment is not adequately considered in the overall post-deployment plans for the system. The sustainment organization needs adequate time to train staff, acquire any special software tools, and develop any necessary facilities in order to perform the sustainment work.

- **Staffing Level at PMO**
  - **General lack of technical oversight exercised in acquisition.** Because of efforts to streamline acquisition offices in the past, it is now common for PMOs to have inadequate staff to provide robust technical oversight of development efforts.
- **Project Monitoring/Control/Oversight**
  - **Incomplete deliveries.** Too often, PMOs focus on the deliverable software for the system. As a result, deliveries include only that software; important and necessary elements such as tool suites, development environments, and supporting documentation are missing. The Army must then acquire those items after the fact and at additional expense. Sustainment organizations sometimes must reverse-engineer missing documentation.
- **Requirements Development/Technical Planning**
  - **Science and technology projects fielded to meet capability need.** A recurring problem in the DoD is the fielding of experimental or proof-of-concept systems because they meet a critical operational capability. These systems typically have been developed with no consideration for maintenance or evolution at all. Trying to force-fit such systems into a traditional acquisition model after the fact in order to complete their development has proven problematic time and again.

3.2.2 Combating the Challenges

To help combat some of the challenges in sustaining weapon system software, the Armament SEC has developed a customer-awareness program. The program helps project managers and PMO staffs think through some of their software issues, particularly with regard to future sustainment, early in the life cycle to facilitate better planning during development. Key elements of the program are helping a PMO to determine what makes the software being acquired on a program supportable throughout the life cycle and how to facilitate the materiel release process for the software.

The SEC helps customers with issues such as

- ensuring the possession of source code within the Armament SEC repository
- obtaining documentation that fully delineates the requirements
- ensuring the capture of the software architecture/design in documentation or in electronic form
- ensuring bi-directional traceability of requirements to software design, source code, test cases and test documentation
- determining the existence of and verifying the software engineering data package (i.e., the software production baseline), which is established through conduct of a software physical configuration audit
- ensuring the existence, availability, knowledge of use, and documentation of the software support environment, to include all taxpayer-developed emulators, simulators, and stimulators used in the software development and test

- ensuring software-build procedures exist, are documented, and are verified to produce a tested version of the software
- ensuring that, at a minimum, Government Purpose Use Rights are obtained for all delivered software, including the explicit identification of any data rights limitations and licensing agreements
- ensuring that configuration management procedures have been established for all software-related products
- providing continuity of operations (e.g., executable code, source code, and documents are secure in an alternate location for disaster-recovery purposes)

4 Discussion and Working Group Summaries

The group's discussion centered on the concept of readiness. What would software readiness connote? Participants agreed that the knowledge represented by a readiness indicator, at any level, should provide assurance that the software is mature enough to proceed to the next acquisition phase. Facets of such knowledge include an understanding that the software technology (e.g., programming languages, tools, and so on) is mature as well as an understanding that the software product (i.e., the code) is mature. There was discussion, but not agreement, that one should also be confident in the maturity of the software-development process.

Further, participants said that a given readiness level for software should ensure that sufficient information is available to justify confidence that the software will be suitable for its intended use and, from a programmatic perspective, that there are adequate data to support confidence that the software will be on cost and on schedule.

Attendees were asked what might be the attributes of software that is ready for its intended use, which produced a list stating that the software must

- be safe
- have stable performance
- have known residual risk
- be maintainable
- be implemented in a way that reflects its intended architecture
- be secure (to the degree needed)
- meet functional and operational needs
- interoperate as defined
  - with partners
  - using defined standards/protocols
- be accepted by intended users
- be validated
- be verified
- be documented
- be supportable
- comply with software engineering best practices
- be on a hardware platform that is available and stable

Software product scalability and requirements stability also were mentioned as potential attributes, but the entire group did not agree that those should be added to the list.

All agreed that the various notions of readiness are important. However, because of time limits the workshop focused on the aspects of software readiness that indicate the maturity of the software product itself and how that maturity relates to *system* milestones.

The central activities of the workshop were the brainstorming sessions. The original plan was to divide participants into teams to brainstorm what should be known about software at the key acquisition life-cycle milestones (including post-deployment). Attendees instead elected to focus on defining a candidate set of "software readiness levels" (SRLs) as a single group.

4.1 Software Readiness Levels

During the course of discussion, one attendee remarked that Gartner, Inc.[4] had a list of technology maturity levels. From least mature to most mature, the list includes the following levels [Fenn 2009]:

- embryonic
- emerging
- adolescent
- early mainstream
- mature mainstream
- legacy
- obsolete

Attendees discussed the relative merits and disadvantages of the Gartner list. In particular, they noted that the latter levels (mature mainstream, legacy, and obsolete) connoted a system's decreasing ability to adapt to change. The last two levels are particularly valuable in that they can guide decision-makers about when to shift investment priorities (i.e., when to invest in further maintenance of an existing system versus when to acquire a replacement system).

Leveraging the Gartner list, the group brainstormed a list of potential software readiness levels that reflected increasing maturity through the software-development life cycle. Attendees then reflected on what might constitute appropriate completion criteria and evidence of completion. The results are shown in Table 3.

[4] Gartner is a leading information technology research and advisory company.

Table 3: Brainstormed Software Readiness Levels

Proposed Software Readiness Level	Completion Criteria	Artifacts that Could Provide Evidence of Completion
Concept	• Features defined • Operations concept defined that represents user expectations • Operational/stakeholder requirements are approved • Appropriate feasibility studies have been conducted (hardware, software, algorithm) • Mission/capability objectives have been defined for the system • Initial set of quality attributes has been defined • Interfaces are defined (as part of requirements?)	• Quality attribute scenarios • Stakeholder requirements document • White papers • Algorithms • Initial interface requirements • Software requirements specification (SRS) outline • Use cases • System definitions to show software context (allocation of system functions to hardware, software, humans and relationships among them)/functional architecture • Draft software development plan • Feature impact statement • Concept of operations document • Mission/capability statements
Architecture	• Appropriate architecture views (e.g., modular, runtime, deployment) have been defined • Selections of approved software-development tools have been made • Architecture evaluations have been conducted • Appropriate standards and protocols have been selected • Interfaces have been refined • Initial CSCIs have been defined • Architecture elements have been allocated • Software test strategy has been defined	• Architecture documented in multiple views • Architecture evaluation results • Interface control specifications • Software test and evaluation plan • System/subsystem design document (SSDD) • Criteria for evaluating prototype • DoD Information Assurance Certification and Accreditation Process (DIACAP) plan/selection of approved software tools
Design/Prototype	• Functioning prototype • Software design is defined • CSCIs are refined • Architectural allocations are fleshed out • Results of analyzing prototype(s) • Proof of “reproducibility” (via a functioning configuration management system) • Software test planning is complete	• Test plan/initial draft of test cases • Resource requirements (technical performance measures) • Software design document • Working version description document (VDD) • List of included COTS products • Requirements traceability matrix • Architecture-to-design traceability matrix • Updated SRS • Software WBS • Updated software architecture views • Database schema • Prototype evaluation report • Software development plan

Proposed Software Readiness Level	Completion Criteria	Artifacts that Could Provide Evidence of Completion
Developed	• Unit testing is completed • Software integration is completed • Interfaces have been verified • Software is documented and reviewed • Peer reviews are conducted and analyzed • Developed code conforms to documentation • Code is under appropriate configuration management • Quality attributes have been verified • Freeze criteria for code have been defined	• Unit and software integration test results • Functional Configuration Audit/Physical Configuration Audit (FCA/PCA) results • Peer review reports • Updated VDD • Software users manual • Software quality statement • Metrics map to software quality attributes (technical performance measures [TPM]) and analysis results • Criteria for software freeze • Software Formal Qualification Test (FQT) dry run report • Software test readiness review document • Deployment plan
Ready for Deployment	• Formal certifications are completed (information assurance [IA], interoperability, safety, authority to operate [ATO], etc.) • System test complete • User acceptance tests completed • Operational tests/assessments are complete • Maintenance strategy defined • Baseline is established • Post-deployment support infrastructure established (field support, tiered support, training, help desk…)	• Certification reports and issued certifications or waivers • Operational test report • VDD • FQT/Software Test Report (STR) • Software users manual • Software operators manual • Software tech manual • Software sustainment/ supportability/suitability plan • User acceptance memo • Deployment plan • Concurrence letter from PM • Validated data loading and installation scripts • Safety and IA checklist (how to use a simple key loader [SKL] to load an encryption key)

Upon reflection overnight, attendees were dissatisfied with the brainstormed readiness levels and associated artifacts. One attendee commented, "It feels like we reinvented software engineering," referring to the list of fairly standard completion criteria and artifacts shown in Table 3. Attendees acknowledged the need to address the goodness of things rather than the existence of things (i.e., artifacts must have achieved a certain level of quality and completeness to be considered acceptable). Further, artifacts are places to look for evidence; they do not necessarily represent evidence in and of themselves. Thus, there is a need for technical evaluation of artifacts and evidence; check-the-box exercises, where completed artifacts receive full credit and acceptance without regard for their sufficiency, will prove useless in determining the readiness of software.

There was considerable discussion about the appropriateness of using readiness levels for software. It was suggested that assigning arbitrary readiness levels might be an errant approach for software. Another suggestion was that *software readiness level* might be the wrong name, as the term "readiness level" carries a lot of baggage and may set incorrect expectations. In the end, attendees acknowledged the need for some way of measuring knowledge about software in

systems as a basis for investment decisions. Readiness levels can provide a means of communicating with people who do not understand software—it would be useful to state that a software change causes a program to move from SRL 5 to SRL 1. Unfortunately, current TRAs do not add sufficient value when making judgments about software.

Attendees concluded that the workshop really was an attempt to fix the software-development process with respect to acquisition. Software engineering should be tied to system engineering in the acquisition life cycle. With that in mind, workshop facilitators kicked off the next activity.

4.2 Mapping SRLs to the Acquisition Life Cycle

The other central activity of the workshop was mapping the candidate software readiness levels to the acquisition life cycle. For this activity, attendees were divided into two teams based on their relative expertise:

- Team 1 focused on software embedded in a system. Such systems include modern tanks, helicopters, and similar systems.
- Team 2 focused on software as the system (i.e., where software is the principal system element, ignoring the computer hardware needed for the software to execute). Such systems include ERP systems and command-and-control systems.

Although the named readiness levels were useful in starting discussions, they very quickly drove waterfall thinking in both groups. This outcome suggests that numbered readiness levels may be the better approach because the abstraction away from concrete terms presents less of an enticement toward constrained thought.

4.2.1 Software Embedded in a System

Figure 4 shows how Team 1 mapped the brainstormed SRLs to the Defense Acquisition Management Life Cycle. As depicted, the mapping for embedded software was relatively straightforward. In general, Team 1 said a given SRL must be met in order to proceed further, with the exception that there may be some risk-reduction actions that justify approval to proceed even if an SRL has not been fully met. The mapping indicates the latest point at which a given SRL should be achieved; achieving readiness levels sooner is desirable. An SRL review (whether by TRA or another mechanism) should occur before the noted SRL assessment milestones. Team 1 felt it was important to ensure the software development remained synchronized to the system development. Hence, most SRLs are not mapped to the acquisition milestones but rather to key points in the system-development process.

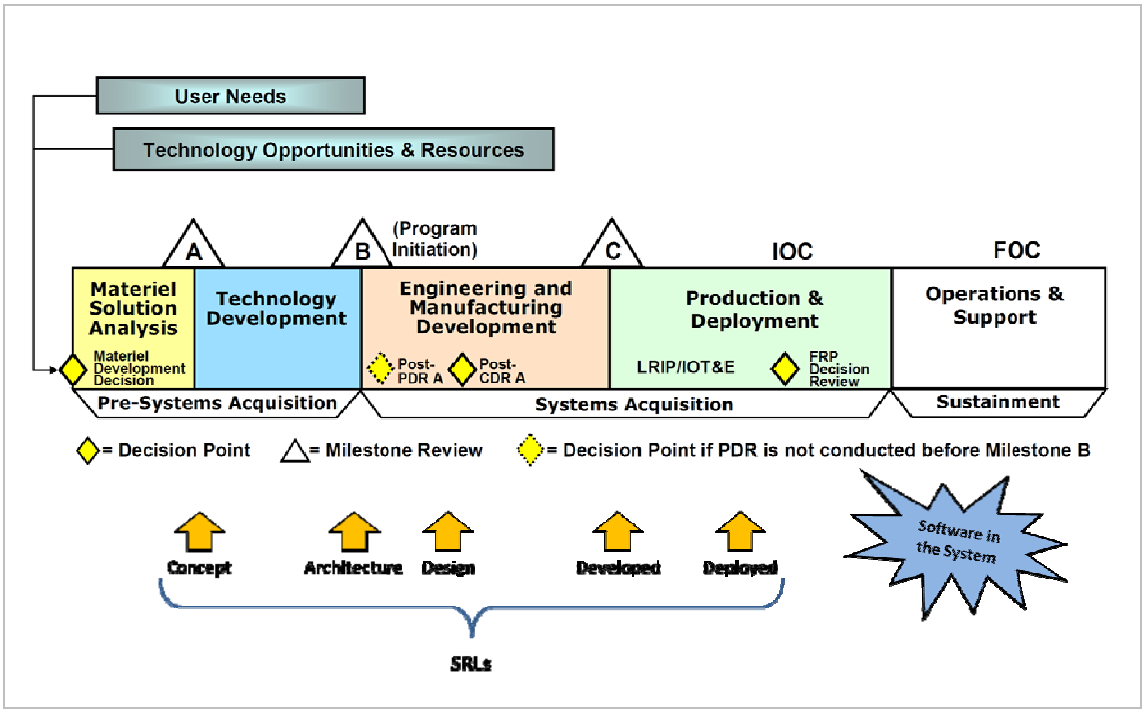

Figure 4: Mapping Readiness Levels to the Acquisition Life Cycle for Embedded Software

The *Concept Level* supports a Milestone A decision. There should be a firm idea of which system elements would be implemented in software. A list of hazards and vulnerabilities with respect to safety, security, and reliability should also be available.

The *Architecture Level*, meaning the software architecture is at a sufficiently mature state to be evaluated objectively, is tied to the system preliminary design review (PDR) or Milestone B, whichever comes first.

The *Design Level*, at which the software design should be largely complete, is tied to the system critical design review (CDR) (i.e., achievement of the SRL should be an entrance criterion for CDR).

The *Developed Level*, meaning that all software has been written and tested up to and including integration testing, is tied to a low-rate initial production (LRIP) decision. Team 1 highlighted the need to have achieved readiness in excess of the Design Level to pass Milestone C if the LRIP decision would fall after the milestone. A key component of passing Milestone C would be a plan for fully achieving the Developed Level by the point at which an LRIP decision would be made.

The *Deployed Level*, indicating that the software has received all required certifications (or waivers) and is ready to be installed on fielded systems, is tied to the materiel release decision.

Although post-deployment issues were not explicitly considered, Team 1 felt there would be a need to reevaluate the SRL in light of any post-deployment changes.

4.2.2 Software is the System

Mapping SRLs to the acquisition life cycle in cases where the software "is" the system was somewhat more complicated, as is evident from Figure 5. Figuring prominently in this domain is the notion of iterative development. Team 2 noted the need to plan incremental testing, start early, and do it frequently (in other words, be more agile). Interestingly, this group, too, for the most part chose to align SRLs to key points in the system-development process rather than to acquisition milestones.

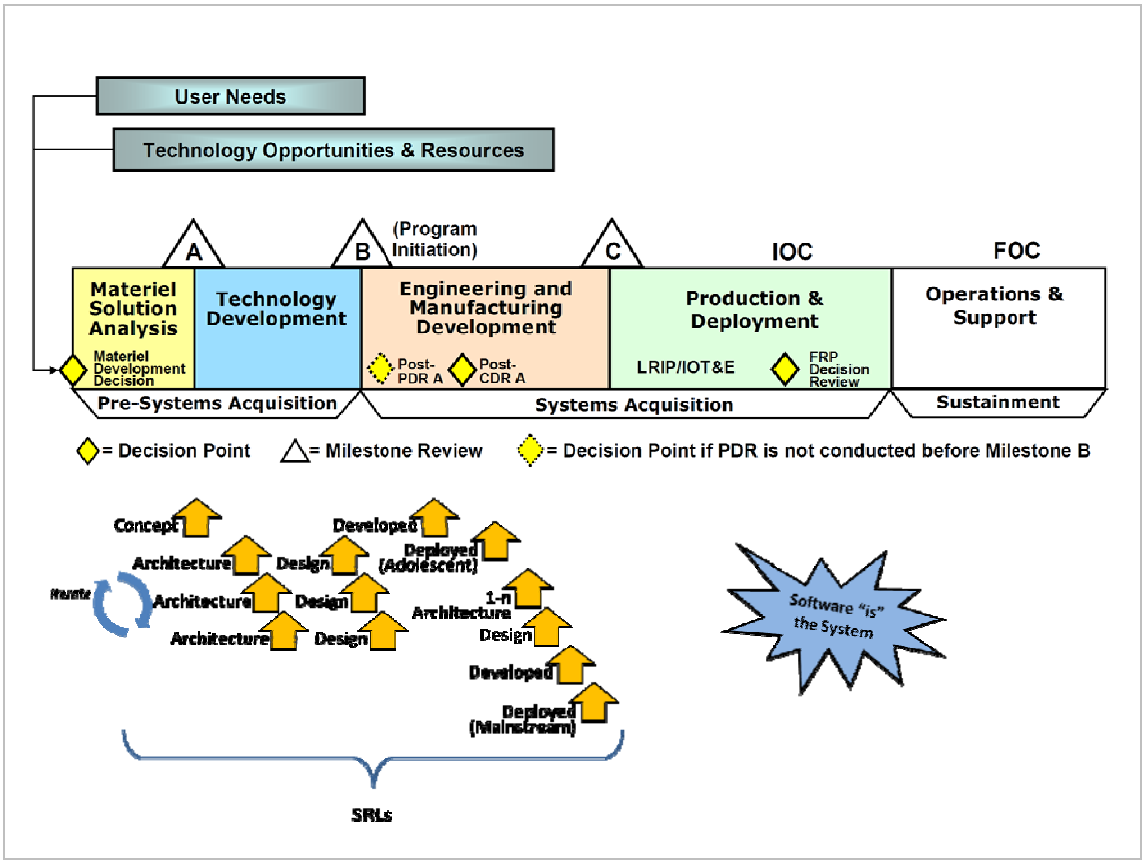

Figure 5: Mapping Readiness Levels to the Acquisition Life Cycle when Software is the System

Here again, the Concept Level should support a Milestone A decision. However, the concept of the software in this case is more concerned with identifying COTS/GOTS software packages for potential use and planning make/buy analyses.

The initial Architecture Level is achieved by the system requirements review (SRR). Such relatively early readiness is possible because, unlike an embedded system, the software system is not dependent on many non-software design decisions. Similarly, the initial Design Level can be achieved at Milestone B. The other significant difference from the embedded system mapping is that the architecture and design work iterate during the Technology Development phase of the life cycle between SRR and PDR. However, Team 2 noted that color-of-money issues and current acquisition schedules are too restrictive for software that may be ready to achieve the Develop Level prior to Milestone B.

As a consequence of the early iteration of architecture and design, the software system reaches the Developed Level of readiness early as well. As shown in Figure 5, this level is achieved by CDR. The Deployed Level of the initial (adolescent) system can then be achieved just after that time and prior to Milestone C.

Significantly, development of the software system iterates again at this point, as the system is matured to its fully operational state. The iteration results in another round of readiness level evaluations prior to the deployment of the "mainstream" system at Milestone C. The significance of this timing is that Milestone C is a *deployment decision* for the software system rather than a manufacturing decision, as would be the case for embedded systems.

4.2.3 Technology/Product/Process Issues

During the mapping exercise, a third team attempted to differentiate among technology/product/process issues. Based on Table 3, Team 3 noted that there seemed to be more product insight early on in the proposed SRLs, while there was more of a process focus at later stages. There was an acknowledged need to consider programmatic aspects of software development (such as cost estimations) but little agreement about how to do so.

Plenary discussion concluded that the proposed SRLs represented a good start but were by no means the final answer. There may be a need for more SRLs (i.e., levels that address post-deployment explicitly). Conversely, fewer SRLs might be possible if the proposed list were redefined. The group noted the need to reflect the DoD's currently defined TRL structure (see Appendix B), either by explicitly redefining existing categories (in collaboration with the DoD) or by mapping new SRLs to the TRLs. Finally, the group discussed the need to evaluate any set of SRLs against different acquisition models to ensure the levels made sense and added value across the span of different software-development projects.

4.3 System of Systems Considerations

As part of the original workshop agenda, the group was to discuss software-readiness implications in relation to systems of systems. Because of time constraints, the planned discussion was deferred to a sidebar after the conclusion of the workshop. A small subset of attendees participated.

Those who stayed for the discussion noted that, increasingly, each system is a component in one or more SoSs. Some of these SoSs are known in advance; planning for the necessary interactions can be achieved up front, often through interface agreements and specifications. However, some of these SoS are not known until "runtime." In other words, the external interactions with other SoS components are not known until the system is operating in its intended environment. In such cases, interactions must be governed by standard information exchange policies to which each member of the SoS adheres. For any given system, there is likely a combination of these types of interactions.

Fundamental to this new reality is that the communications model has changed. No longer can a system rely upon simple communication protocols. The tyranny of the network forces every system to deal with multiple layers of the open systems interconnection (OSI) model. At a

minimum, systems must identify the data and services they need as well as the data and services they provide.

In order to adequately protect SoSs, developers need to understand where to constrain these systems. However, much of the power of SoSs is in the unconstrained emergent properties they exhibit. Finding the right balance for that tension has proven difficult.

Another difficulty in SoS development has been the ability of different generations of technical people to collaborate effectively. The older generations, for whom personal computers were the technology standard, are effectively immigrants to the modern digital reality. Current generations—digital natives, so to speak—expect to select icons rather than write source code. A bridge is needed between the digital immigrants and the SoS-centric digital natives [Prensky 2001].

Current software-readiness notions seem to be especially challenged in an SoS environment, particularly when the constituent systems are on different development schedules. It is not clear that the ideas from this workshop would be any better.

5 Conclusion and Future Work

The importance of software to the capability delivered by most modern DoD systems has focused new light on software-development efforts even if they represent a relatively small part of overall system-development programs. As a result, the DoD has become keenly interested in determining the readiness of software with an eye toward reducing overall program risk. Unfortunately, current conceptions of software readiness were derived from hardware-readiness notions and have proven problematic in practice.

At the conclusion of the workshop, facilitators noted a lack of clear distinctions among three separate but related goals: (1) building product quality; (2) achieving predictable cost and schedule; and (3) developing confidence in the building blocks of a system. This outcome suggests that the answer to "readiness for what?" has not been clearly communicated across the DoD.

While this workshop offered some new perspectives on readiness levels for software, it is clear that the problem is not a simple one. Many questions must still be answered with regard to software readiness:

- What are the right sets of product/process completion criteria?
- How should SRLs be mapped to existing TRLs, if at all?
- How, if at all, should programmatic aspects be incorporated into assessments of readiness?
- How can the proposed SRLs be used with different acquisition scenarios (e.g., COTS-driven development or major weapon enhancement)?
- How, if at all, can SRLs work in the SoS environment?
- How should SRL concepts be socialized with the proponents of the current DoD TRL schema?

Of the 38 workshop attendees, 26 expressed interest in continuing discussions on this topic. Plans are underway for the ASSIP to continue investigations in FY11.

Feedback

The SEI, through its Acquisition Support Program, is working to help improve the acquisition of software-reliant systems across the U.S. government. As part of its mission, the SEI is very interested in learning how other organizations are dealing with readiness levels and technology readiness assessments with regard to software. In addition, the SEI is pleased to discuss the information in this report in more detail.

Please send questions or comments about this report to Stephen Blanchette, Jr. at sblanche@sei.cmu.edu.

Appendix A: Acronyms and Abbreviations

The list below contains all acronyms and abbreviations, and their meanings as used in this report.

ARDEC
Armament Research, Development, and Engineering Center

AMRDEC
Aviation and Missile Research, Development, and Engineering Center

ASA(ALT)
Assistant Secretary of the Army (Acquisition, Logistics, and Technology)

ASSIP
Army Strategic Software Improvement Program

ATO
Authority to Operate

CDR
Critical Design Review

CECOM
Communications-Electronics Command

CERDEC
Communications-Electronics Research, Development and Engineering Center

CMU
Carnegie Mellon University

COTS
Commercial-Off-the-Shelf

CSCI
Computer Software Configuration Item

CTE
Critical Technology Element

CTSF
Central Technical Support Facility

DCMA
Defense Contract Management Agency

DIACAP
DoD Information Assurance Certification and Accreditation Process

DoD
Department of Defense

DoDI
Department of Defense Instruction

DT&E
Developmental Test and Evaluation

ERP
Enterprise Resource Planning

FCA
Functional Configuration Audit

FOC
Full Operational Capability

FQT
Formal Qualification Test

FRP
Full-Rate Production

GOTS
Government-Off-the-Shelf

IA
Information Assurance

IDE
Integrated Development Environment

IOC
Initial Operational Capability

IOT&E
Initial Operational Test and Evaluation

IT
Information Technology

LRIP
Low-Rate Initial Production

OSD
Office of the Secretary of Defense

OSI
Open Systems Interconnection

OT&E
Operational Test and Evaluation

PCA
Physical Configuration Audit

PDR
Preliminary Design Review

PEO
Program Executive Office

PM
Project/Program Manager

PMO
Program Management Office, Project Management Office

QESA
Quality Engineering and System Assurance

R&D
Research and Development

RFP
Request for Proposal

SEC
Software Engineering Center

SED
Software Engineering Directorate

SEI
Software Engineering Institute

Sim/Stim
Simulation/Stimulation

SKL
Simple Key Loader

SoS
System of Systems

SRL
Software Readiness Level

SRR
System Requirement Review

SRS
Software Requirements Specification

SSDD
System/Subsystem Design Document

SSTC
Systems and Software Technology Conference

STR
Software Test Report

TPM
Technical Performance Measures

TRA
Technology Readiness Assessment

TRL
Technology Readiness Level

VDD
Version Description Document

WBS
Work Breakdown Structure

Appendix B: DoD TRL Definitions

Table 4 shows the official DoD definitions of technology readiness levels for both hardware (left half of the table) and software (right half of the table) [DoD 2009].

Table 4: DoD Definitions of Technology Readiness Levels for Hardware and Software

Hardware TRL Definitions, Descriptions, and Supporting Information			Software TRL Definitions, Descriptions, and Supporting Information		
TRL Definition	**Description**	**Supporting Information**	**TRL Definition**	**Description**	**Supporting Information**
1 *Basic principles observed and reported.*	Lowest level of technology readiness. Scientific research begins to be translated into applied research and development (R&D). Examples might include paper studies of a technology's basic properties.	Published research that identifies the principles that underlie this technology. References to who, where, when.	**1** *Basic principles observed and reported.*	Lowest level of software technology readiness. A new software domain is being investigated by the basic research community. This level extends to the development of basic use, basic properties of software architecture, mathematical formulations, and general algorithms.	Basic research activities, research articles, peer-reviewed white papers, point papers, early lab model of basic concept may be useful for substantiating the TRL.
2 *Technology concept and/or application formulated.*	Invention begins. Once basic principles are observed, practical applications can be invented. Applications are speculative, and there may be no proof or detailed analysis to support the assumptions. Examples are limited to analytic studies.	Publications or other references that outline the application being considered and that provide analysis to support the concept.	**2** *Technology concept and/or application formulated.*	Once basic principles are observed, practical applications can be invented. Applications are speculative, and there may be no proof or detailed analysis to support the assumptions. Examples are limited to analytic studies using synthetic data.	Applied research activities, analytic studies, small code units, and papers comparing competing technologies.
3 *Analytical and experimental critical function and/or characteristic proof of concept.*	Active R&D is initiated. This includes analytical studies and laboratory studies to physically validate the analytical predictions of separate elements of the technology. Examples include components that are not yet integrated or representative.	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for critical subsystems. References to who, where, and when these tests and comparisons were performed.	**3** *Analytical and experimental critical function and/or characteristic proof of concept.*	Active R&D is initiated. The level at which scientific feasibility is demonstrated through analytical and laboratory studies. This level extends to the development of limited functionality environments to validate critical properties and analytical predictions using non-integrated software components and partially representative data.	Algorithms run on a surrogate processor in a laboratory environment, instrumented components operating in a laboratory environment, laboratory results showing validation of critical properties.

Hardware TRL Definitions, Descriptions, and Supporting Information			Software TRL Definitions, Descriptions, and Supporting Information		
TRL Definition	**Description**	**Supporting Information**	**TRL Definition**	**Description**	**Supporting Information**
4 *Component and/or breadboard validation in a laboratory environment.*	Basic technological components are integrated to establish that they will work together. This is relatively "low fidelity" compared with the eventual system. Examples include integration of "ad hoc" hardware in the laboratory.	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	**4** *Module and/or subsystem validation in a laboratory environment (i.e., software prototype development environment).*	Basic software components are integrated to establish that they will work together. They are relatively primitive with regard to efficiency and robustness compared with the eventual system. Architecture development initiated to include interoperability, reliability, maintainability, extensibility, scalability, and security issues. Emulation with current/legacy elements as appropriate. Prototypes developed to demonstrate different aspects of eventual system.	Advanced technology development, stand-alone prototype solving a synthetic full-scale problem, or standalone prototype processing fully representative data sets.
5 *Component and/or breadboard validation in a relevant environment.*	Fidelity of breadboard technology increases significantly. The basic technological components are integrated with reasonably realistic supporting elements so they can be tested in a simulated environment. Examples include "high-fidelity" laboratory integration of components.	Results from testing a laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the "relevant environment" differ from the expected operational environment? How do the test results compare with expectations? What problems, if any, were encountered? Was the breadboard system refined to more nearly match the expected system goals?	**5** *Module and/or subsystem validation in a relevant environment.*	Level at which software technology is ready to start integration with existing systems. The prototype implementations conform to target environment/interfaces. Experiments with realistic problems. Simulated interfaces to existing systems. System software architecture established. Algorithms run on a processor(s) with characteristics expected in the operational environment.	System architecture diagram around technology element with critical performance requirements defined. Processor selection analysis, Simulation/Stimulation (Sim/Stim) Laboratory buildup plan. Software placed under configuration management. Commercial-of-the-shelf/government-off-the-shelf (COTS/GOTS) components in the system software architecture are identified.

Hardware TRL Definitions, Descriptions, and Supporting Information			Software TRL Definitions, Descriptions, and Supporting Information		
TRL Definition	**Description**	**Supporting Information**	**TRL Definition**	**Description**	**Supporting Information**
6 *System/subsystem model or prototype demonstration in a relevant environment.*	Representative model or prototype system, which is well beyond that of TRL 5, is tested in a relevant environment. Represents a major step up in a technology's demonstrated readiness. Examples include testing a prototype in a high-fidelity laboratory environment or in a simulated operational environment.	Results from laboratory testing of a prototype system that is near the desired configuration in terms of performance, weight, and volume. How did the test environment differ from the operational environment? Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?	**6** *Module and/or subsystem validation in a relevant end-to-end environment.*	Level at which the engineering feasibility of a software technology is demonstrated. This level extends to laboratory prototype implementations on full-scale realistic problems in which the software technology is partially integrated with existing hardware/software systems.	Results from laboratory testing of a prototype package that is near the desired configuration in terms of performance, including physical, logical, data, and security interfaces. Comparisons between tested environment and operational environment analytically understood. Analysis and test measurements quantifying contribution to system-wide requirements such as throughput, scalability, and reliability. Analysis of human-computer (user environment) begun.
7 *System prototype demonstration in an operational environment.*	Prototype near or at planned operational system. Represents a major step up from TRL 6 by requiring demonstration of an actual system prototype in an operational environment (e.g., in an aircraft, in a vehicle, or in space).	Results from testing a prototype system in an operational environment. Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?	**7** *System prototype demonstration in an operational high-fidelity environment.*	Level at which the program feasibility of a software technology is demonstrated. This level extends to operational environment prototype implementations, where critical technical risk functionality is available for demonstration and a test in which the software technology is well integrated with operational hardware/software systems.	Critical technological properties are measured against requirements in an operational environment.
8 *Actual system completed and qualified through test and demonstration.*	Technology has been proven to work in its final form and under expected conditions. In almost all cases, this TRL represents the end of true system development. Examples include developmental test and evaluation (DT&E) of the system in its intended weapon system to determine if it meets design specifications.	Results of testing the system in its final configuration under the expected range of environmental conditions in which it will be expected to operate. Assessment of whether it will meet its operational requirements. What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before finalizing the design?	**8** *Actual system completed and mission qualified through test and demonstration in an operational environment.*	Level at which a software technology is fully integrated with operational hardware and software systems. Software development documentation is complete. All functionality tested in simulated and operational scenarios.	Published documentation and product technology refresh build schedule. Software resource reserve measured and tracked.

Hardware TRL Definitions, Descriptions, and Supporting Information			Software TRL Definitions, Descriptions, and Supporting Information		
TRL Definition	**Description**	**Supporting Information**	**TRL Definition**	**Description**	**Supporting Information**
9 *Actual system proven through successful mission operations.*	Actual application of the technology in its final form and under mission conditions, such as those encountered in operational test and evaluation (OT&E). Examples include using the system under operational mission conditions.	OT&E reports.	**9** *Actual system proven through successful mission-proven operational capabilities.*	Level at which a software technology is readily repeatable and reusable. The software based on the technology is fully integrated with operational hardware/software systems. All software documentation verified. Successful operational experience. Sustaining software engineering support in place. Actual system.	Production configuration management reports. Technology integrated into a reuse "wizard."

References

URLs are valid as of the publication date of this document.

[Bandor 2010]
Bandor, Michael & Miller, Suzanne. "Status of Ongoing Work in Software TRAs/TRLs." *Proceedings of Systems and Software Technology Conference*, Hill AFB, 2010. www.sei.cmu.edu/library/abstracts/presentations/bandor-garcia-miller-sstc-apr-2010.cfm

[CJCS 2001]
Chairman of the Joint Chiefs of Staff. *Department of Defense Dictionary of Military and Associated Terms*, Joint Publication 1-02. Washington, DC: Chairman of the Joint Chiefs of Staff, 2001. www.dtic.mil/doctrine/new_pubs/jp1_02.pdf

[CJCS 2006]
Chairman of the Joint Chiefs of Staff. *Joint Operations*, Joint Publication 3-0. Washington, DC: Chairman of the Joint Chiefs of Staff, 2006. www.dtic.mil/doctrine/new_pubs/jp3_0.pdf

[CJCS 2009]
Chairman of the Joint Chiefs of Staff. Chairman of the Joint Chiefs of Staff Instruction, Joint Capabilities Integration and Development System CJCSI 3170.01G. Washington, DC: Chairman of the Joint Chiefs of Staff, 2009. www.dtic.mil/cjcs_directives/cdata/unlimit/3170_01.pdf

[DoD 2008]
Undersecretary of Defense for Acquisition, Technology, and Logistics. *Department of Defense Instruction 5000.02*. Washington, DC: Undersecretary of Defense for Acquisition, Technology, and Logistics, 2008. www.dtic.mil/whs/directives/corres/pdf/500002p.pdf

[DoD 2009]
Director, Research Directorate. *Technology Readiness Assessment (TRA) Deskbook*. Washington, DC: Office of the Director, Defense Research and Engineering, 2009. www.dod.mil/ddre/doc/DoD_TRA_July_2009_Read_Version.pdf

[DoD 2010]
Department of Defense. Ch. 4.1.4 "System of Systems Engineering." *Defense Acquisition Guidebook:* Washington, DC: Pentagon, 2010. https://acc.dau.mil/CommunityBrowser.aspx?id=332957&lang=en-US

[Fenn 2009]
Fenn, Jackie & Raskino, Mark. *Understanding Gartner's Hype Cycles*," Gartner, Inc., 2009. www.gartner.com/DisplayDocument?id=1069314

[Meade 2003]
Meade, Charles & Abbott, Meagan. *Assessing Federal Research and Development for Hazard Loss Reduction*. RAND, 2003. www.rand.org/pubs/monograph_reports/2005/MR1734.pdf

[Prensky 2001]
Prensky, Marc. *Digital Game-Based Learning*. McGraw-Hill Companies, 2001 (ISBN: 0071363440) www.marcprensky.com/dgbl/default.asp

[Wikipedia 2010a]
Wikipedia. "Real-time computing," *Wikipedia, The Free Encyclopedia.* Wikimedia Foundation, Inc. December 2010. www.wikipedia.org/wiki/Real-time_computing

[Wikipedia 2010b]
Wikipedia. "Subsystem," *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, Inc. December 2010. www.wikipedia.org/wiki/Subsystem

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Beyond Technology Readiness Levels for Software: U.S. Army Workshop Report	FA8721-05-C-0003

6. AUTHOR(S)

Stephen Blanchette, Jr., Cecilia Albert, & Suzanne Garcia-Miller

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TR-044

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2010-109

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Carnegie Mellon® Software Engineering Institute (SEI) facilitated an Army workshop, Beyond Technology Readiness Levels for Software, on August 10-11, 2010. The workshop, part of the ongoing Army Strategic Software Improvement Program (ASSIP), was an attempt to develop an Army perspective on the "right" software information to gather and analyze at significant program decision points (especially Milestones A, B, and C) to determine readiness to proceed to the next acquisition phase. This report synthesizes the workshop presentations, discussions, and outcomes.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Technology Readiness Levels; Software Readiness	55

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL