Army Workshop on Lessons Learned from Software Upgrade Programs

William Anderson
John Bergey
Matthew Fisher
Caroline Graettinger
Wilfred J. Hansen
Ray Obenza
Dennis Smith
Halbert Stevens

November 2001

SPECIAL REPORT
CMU/SEI-2001-SR-021

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Army Workshop on Lessons Learned from Software Upgrade Programs

CMU/SEI-2001-SR-021

William Anderson
John Bergey
Matthew Fisher
Caroline Graettinger
Wilfred J. Hansen
Ray Obenza
Dennis Smith
Halbert Stevens

November 2001

This report was prepared for the Department of Army.

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

This work was created in the performance of Federal Government Contract Number F19628-00-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.

Table of Contents

List of Figures

Executive Summary

The Software Engineering Institute (SEI) conducted the "Software Upgrade Workshop for Legacy Systems" at the Redstone Arsenal from June 5–7, 2001. The workshop captured experiences from software upgrade efforts for the Abrams, Bradley, Patriot, Apache Longbow, and Multiple Launch Rocket programs. Workshop participants explored strategies that worked and those that failed, and the obstacles that were encountered. They addressed preaward planning, project management, systems and software, mandates, and deployment.

A high-level summary of the findings is presented below. The findings are summarized according to the topics of the four working groups:

Project Management

- Specify requirements clearly, using tool support when appropriate; and practice change control.
- Use historical data and tools for collecting data.
- Provide training on the use of cost estimation methods and tools.
- Consider the effect of potential upgrades when scheduling.
- Be sure to have experienced senior staff. Invest in worker skills and identify career paths.
- Involve all stakeholders throughout the project, particularly through the use of integrated product teams (IPTs).

Systems and Software

- Design the architecture with change in mind; plan for regular legacy system upgrades.
- Have the appropriate development and test environment resources and tools in place.
- When using commercial off-the-shelf (COTS) components for an extensive part of a system, allow ample time for integration, and require the contractor to integrate the components.
- Maintain the integrity of testing even if cuts must be made in a project.

Funding and Contracting

- When money is tight, don't sacrifice sound engineering practices.
- Designate a decision-making agency to resolve inter-project conflicts.

- Consider logistics, upgrades, and contingencies throughout the planning cycle.
- Address software problems right away; don't defer them to follow-on contracts.
- Strive for mature processes for both the government and the contractor, especially in the areas of requirements management and control.
- If a contractor has attained Capability Maturity Model® for Software (SW-CMM®) level 3, require the use of its metrics and data for managing the contract.
- Determine software ownership rights before the contract award.
- Mandates often have negative unanticipated consequences. Be very cautious about issuing mandates and involve all stakeholders in the decision-making process.

Deployment

- Be sure that the documentation is appropriate. It should contain enough information to adequately maintain a project, without including irrelevant details.
- Write training materials prior to the release and deploy training before fielding the new system.
- Avoid configuration proliferation.
- Plan for block upgrades.
- When possible, automate the installation of upgrades.
- Provide regular upgrade schedules.

® Capability Maturity Model, Capability Maturity Modeling, and CMM are registered in the U.S. Patent and Trademark Office.

Abstract

The Software Engineering Institute (SEI) conducted the “Software Upgrade Workshop for Legacy Systems” at the Redstone Arsenal June 5–7, 2001. The workshop captured experiences from software upgrade efforts for the Abrams, Bradley, Patriot, Apache Longbow, and Multiple Launch Rocket programs. Workshop participants explored strategies that worked and those that failed, and the obstacles that were encountered. They addressed pre-award planning, project management, software and systems, mandates, and deployment. Their effort resulted in a set of recommendations and guidelines to help organizations improve the process of upgrading legacy systems.

1 Introduction

The SEI conducted a Software Upgrade Workshop for Legacy Systems at the Redstone Arsenal June 5–7, 2001. This workshop was sponsored by the Office of the Assistant Secretary of the Army for Acquisition, Logistics and Technology (ASA [ALT]). The workshop captured lessons learned from software upgrade efforts to the Abrams, Bradley, Patriot, Apache Longbow, and Multiple Launch Rocket programs. Workshop topics and questions were based on experiences with other software upgrade and acquisition efforts.

The workshop split into four working groups consisting of peers from each of the programs mentioned above. Participants discussed basic issues and problems, and examined topics across programs. The lessons learned from this effort can be useful for future software upgrade efforts. All discussions were non-attributable.

2 Workshop Approach

This section describes how the topics and questions for the workshop were developed. It lists the working group topics, and describes the steps taken in preparing for the workshop. It also outlines the format of the report.

2.1 Development of Topics and Questions to Guide Workshop

The topics and questions for guiding the workshop were developed from SEI experience with other organizations that have undergone software upgrade efforts. Members of the SEI team synthesized their experiences from three different perspectives: (1) software migration and system evolution, (2) software acquisition, and (3) software risk evaluation.

In the course of working with organizations on software migration and system evolution, the SEI has developed a framework for disciplined system evolution, a set of checklists to use when applying this framework, and guidelines for system migration [Bergey 97, 98, 99]. This material helps organizations identify the issues that should be addressed for a successful migration effort, including issues related to software, systems, the organization, the project, and the legacy and target systems. It helped the SEI to formulate some of the initial questions for the working groups and to interpret the findings.

To help organizations benchmark and improve the software acquisition process, the SEI developed its Software Acquisition Capability Maturity Model® (SA-CMM®) [Cooper 99]. The model follows the same architecture as the Capability Maturity Model for Software (SW-CMM), but emphasizes acquisition issues and the needs of those who are planning and managing software-acquisition efforts. SA-CMM describes the practices that are most critical for an acquisition organization to follow. The issues identified by the SA-CMM formed the basis for some of the acquisition topics and questions.

To successfully acquire or develop software, organizations must identify risks and implement strategies to mitigate them. The software risk evaluation (SRE) process helps organizations identify, analyze, and develop risk-mitigation strategies while the software-intensive system

® Capability Maturity Model, Capability Maturity Modeling, and CMM are registered in the U.S. Patent and Trademark Office.

is still in development [Williams 99]. The team used the SRE approach to help formulate workshop topics and questions.

After consulting with Army ASA (ALT) representatives and reviewing the topics and questions from these perspectives, the team developed four working groups:

1. Systems and Software
2. Funding and Contracting
3. Deployment
4. Project Management

These areas interact with the program manager as shown in Figure 1.

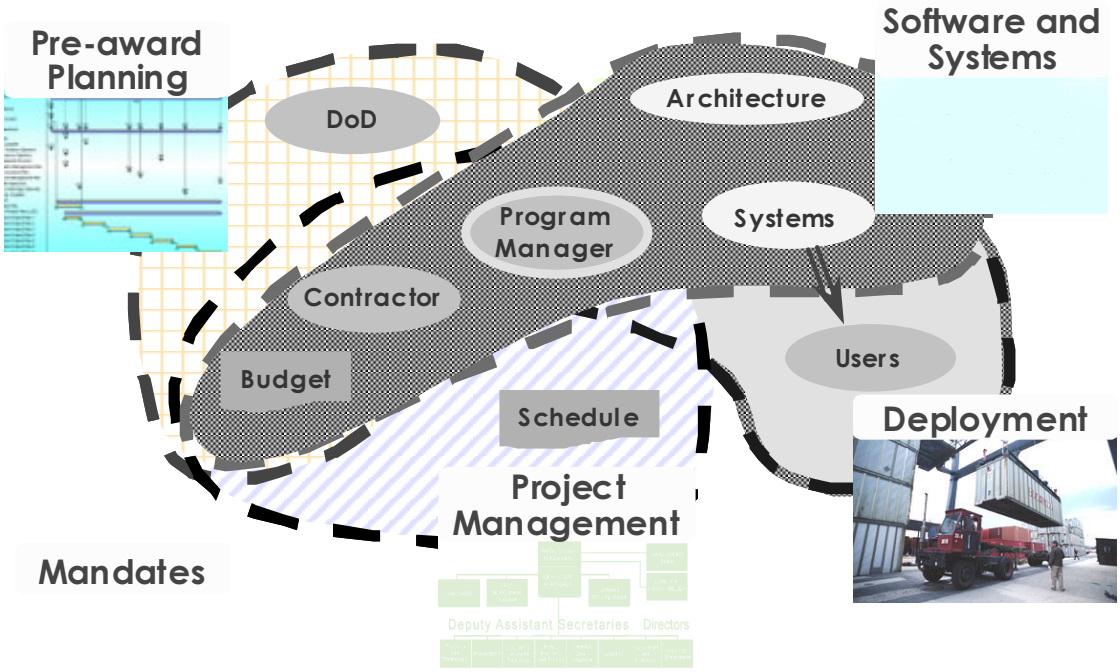

Figure 1: Four Work Groups (Plus One Topic: Mandates)

The team next developed starting points for each working group. These are outlined in Section 2.2. The team then analyzed source documents in detail to produce and refine a set of questions appropriate for generating discussion.

While the topics and questions for each working group were derived from experience, each group was encouraged to move beyond the questions as appropriate and spend time on topics that had the most relevance and importance. The questions thus provided a starting point for their efforts, rather than a rigid guide.

During their intense discussions, the working groups captured salient points on a visual display for review by all. This report reflects their findings and provides a rich set of lessons learned, recommendations, and guidelines based on their experiences.

2.2 Working Group Topics

The initial topics covered by each working group were:

1. Project Management

- requirements schedule
- project planning
- project management
- technical contract oversight
- process definition and maintenance

2. Systems and Software

- systems engineering
- software engineering
- technologies
- architectures
- legacy system
- target system
- interoperability

3. Funding and Contracting

- software contracting scope
- contractual planning
- government/contractor relationship
- contract performance
- decision making on cost and schedule

4. Deployment

- testing
- integration
- configuration management
- operational suitability
- migration planning
- training

2.3 Preparation for Workshop

All participants were provided with mailings that described the nature and focus of the workshop. A Web site[1] was developed containing extensive information on the workshop, includ-

[1] <http://www.sei.cmu.edu/cbs/workshops/ASA-ALT-Software-Upgrades-Workshop>

ing topics and starter questions. Currently, the Web site contains a record of the workshop, including the final briefings of each working group and this workshop report.

2.4 Format of Report

Following the introduction (Section 1) and workshop approach (Section 2), this report presents a summary of lessons learned (Section 3) and a set of conclusions (Section 4). The conclusions relate the lessons learned to the system-migration guidelines previously developed by the SEI. Appendices A through D contain a list of working group members, the major topics of the group's discussions, and any adjustments that group members made to their topics. Following these sections, the reports present findings by topic. The findings identify relevant issues and problems then examines

- approaches that programs have implemented to address the issues, and lessons learned. When appropriate, case studies are highlighted to illustrate challenges, actions taken, and results. Since the workshop promised non-attribution, these anecdotes, although reflecting a real situation, are phrased in general terms.
- recommendations showing how other programs or high-level management can augment or modify existing policies and practice

In some cases, a working group combined several topics under a new heading or added new topics that were relevant to the discussions. On occasion, two working groups considered similar topics from the differing perspectives. These discussions tended to be complementary and the findings are presented in the most appropriate report.

Appendix E follows the four working group reports. It discusses the findings in light of the enterprise framework for disciplined system evolution. Appendix F contains questions from the system-evolution checklists.

3 Summary of Lessons Learned

The discussions of the working groups were productive and prolific. In all, the four working groups developed more than 250 lessons learned. To summarize them, we identified common themes and then grouped lessons learned and recommendations under those themes. Each summary abstracts the findings of the entire workshop, and does not necessarily reflect the complete discussions of individual working groups. Appendices A through D include a record of the full discussions.

The findings represent thoughtful reflections by a group of people with hard-won experience. While the findings cover a broad range of topics, they are not designed to be a complete or systematic set of guidelines. If such an effort were launched, these findings would represent a very useful starting point.

Appendix E relates the findings to the "Enterprise Framework for Disciplined System Evolution" [Bergey 97]. This framework provides a comprehensive mechanism for understanding the findings in terms of their relationships.

3.1 Project Management

Project management lessons were classified under the categories of requirements, scheduling, performance management and skills, and communication.

Requirements

- Practice formal change control.
- Specify requirements clearly, using tool support when appropriate.
- Do not over-specify requirements.
- Enforce cutoff dates for requirements for each release.

Scheduling

- Use historical data and tools for collecting data.
- Provide training on the use of cost-estimation methods and tools.
- Collect, evaluate, and disseminate information on aids for estimating.

- Consider the effect of potential upgrades when setting schedules.

Personnel Management and Skills

- Be sure to have experienced senior staff. It is particularly important to have a domain expert as the PM.
- Invest in worker skills and identify career paths.
- Develop plans to retain skilled workers.
- Give program managers authority to improve work conditions and provide incentives to workers.
- Award managers based on performance.
- Provide more training in earned value, IPTs, and software-acquisition management.
- Train military officers to understand software issues and to raise issues about software decisions.

Communication

- Involve all stakeholders throughout the project, particularly through the use of IPTs.
- Clearly state the limitations of what will be delivered with the funding provided.

3.2 Systems and Software

Systems and Software issues include architecture, tool support, GFE and COTS, and testing.

Architecture

- Design with change in mind; plan for regular legacy-system upgrades.
- Analyze the implications of throughput and memory.
- Analyze potentials for reuse.

Tool Support

- Have the appropriate development and test-environment resources in place.
- When appropriate, select and adopt tools for

– requirements management
– risk management
– metrics management
– automated configuration testing and upgrades

GFE and COTS

- Allow ample time for integration.
- When using COTS extensively in a system, get the COTS vendor under contract.

- Require contractors to integrate COTS and GOTS products.

Testing

- Maintain the integrity of testing even if cuts need to be made in a project.
- Identify and preserve test assets.
- Provide automated test scripts when appropriate.
- Plan testing fully; be sure to align testing with all relevant projects.

3.3 Funding and Contracting

Funding and contracting issues include processes used to determine costs and schedules, role of software issues in acquisition, contract performance, software data rights, and mandates.

Decision Making on Cost and Schedule

- When money is tight, don't sacrifice sound engineering practices.
- Involve all parties in estimating schedule and effort.
- Ask external groups to develop estimates for interrelated, cross-PEO projects.
- Designate a decision-making agency to resolve inter-project conflicts.
- Consider logistics, upgrades, and contingencies throughout the planning cycle.
- Empower IPTs. Have senior management on the teams and give them decision-making authority.

Role of Software Issues in Acquisition

- Address software problems right away; don't defer them to follow-on contracts.
- Fund software from the beginning because software implements system performance.
- Insert a level-of-effort (LOE) funding line in development contracts for continuing software enhancements.

Contract Performance

- Strive for mature processes for both the government and the contractor, especially in the areas of requirements management and control.
- Extend the current CMM requirement for all ACAT 1 programs for all software contracts.
- If a contractor has attained CMM level 3, require the use of its metrics and data for managing the contract.
- Hold contractors to their processes.
- Use metrics in contract management and analyze them.

- Improve software capability maturity level of acquiring organizations.

Software Data Rights

- Develop policy and training about the acquisition and retention rights to software.
- Determine whether software ownership will be needed before the contract award.
- Do not let the contractor make default decisions on data rights and software ownership.

Mandates

- Mandates often have negative unanticipated consequences. Be very cautious about issuing mandates and involve all stakeholders in the decision-making process.
- Communicate clearly the purpose of any mandates and the mechanisms for complying with them.
- Specify desired goals, but do not mandate specific solutions, such as COTS, specific CPUs, specific programming languages, or software warranties.
- Identify an authority to resolve conflicts on mandates.
- If mandates are required, it is important to

– provide funding
– write cost-plus contracts
– adopt only widely used standards

3.4 Deployment

Deployment issues include documentation, training, configurations, and development of an upgrade mentality.

Documentation

- Require electronic documents and early access to the documentation.
- Be sure that the documentation is appropriate. It should contain enough information to adequately maintain a project, without including irrelevant details.

Training

- Train acquirers dealing with legacy systems in the areas of test planning and test discipline.
- Write training materials prior to the release and deploy training before fielding the new system.

Configurations

- Avoid configuration proliferation.
- Plan for block upgrades.

- Automate the installation of upgrades.
- Provide machine-readable identification for the hardware and software of every component.
- Use an installation tool that can choose the correct version of a required upgrade.

Development of an Upgrade Mentality

- Incorporate an upgrade mentality in the culture.
- Provide regular upgrade schedules.

4 Conclusion

4.1 Findings

These findings are the result of practical lessons learned from Army efforts. The findings are consistent with the broad migration guidelines previously developed by the SEI [Bergey 99], and provide important recommendations.

Below, we briefly discuss the workshop findings relative to the migration guidelines. Each guideline is listed, and relevant workshop recommendations are highlighted:

1. Develop a comprehensive strategy with achievable and measurable milestones for each reengineering project.

   Projects often try to address too many goals at once without intermediate milestones. The workshop stressed the importance of developing an overall strategy in bite-sized chunks, with a regular upgrade schedule and a clear plan of communication to all stakeholders.

2. When outside systems-engineering services are needed, carefully define and monitor their role.

   The working groups wrestled with how to define and monitor vendors. Current best-practice mechanisms include insisting on mature process use by both the government and contractor, and applying metrics to monitor relevant processes. Since software ownership often becomes a critical factor in follow-up systems, ownership rights should be resolved before the contract is awarded.

3. If new technology is used for a project, provide adequate training in both the technical content and the motivation for change.

   The workshop stressed the importance of investing in worker skills and identifying career paths. It recommended providing training in new methods and tools, and deploying new training prior to system releases.

4. Establish and maintain configuration-management control of the legacy system.

   The working group discussions attested to the importance of configuration management. Recommendations included avoiding configuration proliferation, planning for block upgrades, and automating the installation of upgrades.

5. There should be a carefully defined and documented process for eliciting and validating requirements.

Workshop recommendations included specifying requirements clearly, using tool support when appropriate, and to practicing and enforcing change-control processes.

6. Make software architecture a primary reengineering consideration.

   The Systems and Software working group identified the architecture as a key technical focus for the system. Recommendations included designing the architecture with change in mind, planning for regular technical upgrades, and carefully considering throughput requirements.

7. There should be a separate and distinct reengineering process.

   This guideline stresses that a reengineering or evolution effort should have a well-defined process with appropriate planning, scheduling, and monitoring. The workshop emphasized that organizations should use mature software processes to monitor contract performance, and should analyze historical data before making schedules and decisions. It also recommended establishing processes for requirements management, testing, and COTS integration, and implementing configuration-management and software-engineering tools.

8. Create a team-oriented reengineering plan—and follow it.

   The working groups emphasized the importance of involving all stakeholders in accomplishing project goals. In particular, they emphasized the role of IPTs in conflict resolution, consensus building, and communication.

9. Management must be committed for the long haul.

   While there was substantial discussion on the need for consistent, long-term vision, the practice of rotating top management makes this difficult to resolve. Specific recommendations included addressing software problems right away rather than deferring them to future releases, and considering future upgrades and releases throughout the planning cycle.

10. Management edicts should not override technical realities.

    Management edicts often specify a technical approach or schedule before a detailed technical analysis has been performed. The workshop highlighted the importance of understanding and managing requirements, and developing realistic plans and schedules based on available resources. In addition, the working groups discussed the negative consequences of mandates. Workshop participants urged caution in issuing mandates and recommended that mandates only be considered after affected stakeholders are involved and implications for the programs are analyzed.

4.2 Concluding Note

The workshop successfully elicited a set of practical and useful lessons learned from people who have experience acquiring, developing, deploying, and upgrading mission-critical systems. By applying the lessons learned to future programs, many similar issues and problems can be resolved quickly or, better yet, avoided in the first place. Specific lessons learned are

documented in Appendices A through D. Analyzing these lessons learned can help focus future efforts in terms of establishing high-priority issues.

References

[Bergey 97] Bergey, J; Northrop, L.; & Smith, D. An Enterprise Framework for Disciplined System Evolution (CMU/SEI-97-TR-007). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. URL: <http://www.sei.cmu.edu/publications/documents/97.reports/97tr007/97tr007abstract.html> (1997).

[Bergey 98] Bergey, John K. "System Evolution Checklists Based on an Enterprise Framework" (white paper). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, February 1998. URL: <http://www.sei.cmu.edu/reengineering/pubs/white-papers/Berg98> (1998).

[Bergey 99] Bergey, J.; Smith, D.; & Weiderman, N. DoD Legacy System Migration Guidelines (CMU/SEI-99-TN-013, ADA 370621). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. URL: <http://www.sei.cmu.edu/publications/documents/99.reports/99tn013/99tn013abstract.html> (1999).

[Cooper 99] Cooper, J.; Fisher, M.; & Sherer, S.W. (editors). *Software Acquisition Capability Maturity Model ® (SA-CMM®), Version 1.02* (CMU/SEI-96-TR-020, ADA320606). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. URL: <http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.020.html> (1996).

[Williams 99] Williams, R.; Pandelios, G.J.; & Behrens, S.G. *SRE Method Description (Version 2.0) & SRE Team Members Notebook (Version 2.0)* (CMU/SEI-99-TN-029). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, URL: http://www.sei.cmu.edu/publications/documents/99.reports/99tr029/99tr029abstract.html> (1999).

List of Acronyms

ACAT	Acquisition Capability
ASA (ALT)	Assistant Secretary of the Army for Acquisition, Logistics and Technology
AWR	Air Worthiness Release
CCB	Change Control Board
CPU	Central Processing Unit
C3	Command, Control, Communications
COTS	Commercial Off-the-Shelf
CMRT	Computer Resources Management Team
CTSF	Confirmed Test Site Facility
DCSCOPS	Deputy Chief of Staff for Operations
ECP	Engineering Change Proposal
GFE	Government Furnished Equipment
GICOD	Good Idea Cutoff Date
I/O	Input-Output
IPT	Integrated Product Team
IRAD	Independent Research and Development
IOTE/FOTE	Initial Operational Test and Evaluation
IV&V	Independent Validation and Verification
JTA	Joint Technical Architecture
LOC	Lines of Code
LOE	Level of Effort
LRU	Line Replaceable Unit
MPRNT	Material Process Review Team
MTTR	Mean-Time-to-Repair
OT/IOC	Operational Test/Introduction of Operational Capability
PCR	Problem Change Request
PDF	Portable Display Format

PDSS	Post-Deployment Software Support
PM	Program Manager
PCR	Problem Change Request
RAM	Random Access Memory
SBVT	Software Baseline Verification Test
SIL	System Integration Laboratory
SOW	Statement of Work
SPORT	Soldier Portable On-System Repair Tool
SRE	Software Risk Evaluation
STR	System Trouble Report
SW-CMM®	Capability Maturity Model® for Software
SEI	Software Engineering Institute
TEMP	Test and Evaluation Master Plan
TIM	Technical Interchange Meeting
TRADOC	Training and Doctrine Command
UML	Unified Modeling Language
VVID	Vehicle Verification Identification

Appendix A Project Management Working Group

This working group collected lessons learned and best practices from a project management focus. The scope included the topic areas of

- requirements management
- planning and estimation
- organizing & staffing
- risk management

Session participants included the following representatives from each of the legacy programs in the workshop:

- Ron Bokoch, PM Abrams
- Edmund Cheung, Bradley
- Jerita Crummie, MLRS
- Ed Fowler, Patriot
- Kevin Houser, PM Abrams
- Keith Robinson, Apache
- Abdul Siddiqui, PM Bradley
- Matt Fisher, SEI
- Hal Stevens, SEI

Requirements Management

Requirements Instability

For all the programs, there had been impositions of requirements changes, specifically related to battlefield integration and command and control from higher levels with directed deadlines and no additional funding or analyses of how these requirements changes would impact the individual programs. For some programs, these impositions included the integration of

non-stable products from other programs into existing systems, again with insufficient analysis performed on the impact of the product integration.

Approach and Lessons Learned

To help address the programmatic and technical issues, programs established working-level IPTs to coordinate schedules and facilitate the integration of products. These IPTs had representatives from the program offices, Training and Doctrine Command (TRADOC), and participating contractors. In addition, one program suggested that in order to meet the imposed schedules a good idea cutoff date (GICOD) be established. The GICOD would be a date after which no new requirements from external sources would be accepted for a specific system build. This approach enables the schedules to be met and additional requirements incorporated at a later planned time.

The use of IPTs allowed common issues among the participating programs to be discussed and allowed stakeholders to plan and manage requirements and builds. Most work of the IPT was accomplished at the working level. However, not all issues were resolved at this level.

The use of the GICOD in combination with IPT did not work for one program because dates proposed were not agreed to by all stakeholders and the products to be integrated were not sufficiently stable to comply with the dates. Stability here means that the product builds lacked needed functionality and performance. In addition, higher-level authority continued to add requirement changes even after the proposed dates. The timeframe for imposed integration was too constrained for the requirements to be implemented.

Recommendations

Use of IPTs to manage both programmatic and technical-integration requirements is still considered a good practice. However, these IPTs must involve all relevant stakeholders and be empowered to set schedules and enforce techniques such as the GICOD. The use of cross-PM IPTs was also recommended as well as including DCSOPS as members of appropriate IPTs.

The group also recommended that senior management be provided an oversight role to resolve issues that the IPT cannot resolve. This would assure that issues and risks are identified early and elevated to the correct levels for decisions and mitigation.

Imposed Technical Requirements

In all programs it was believed that insufficient technical analyses were conducted at higher levels of management to determine the technical feasibility of implementing imposed re-

quirements. This was especially true when these requirements involved application of commonality to multiple platforms.

Approach and Lessons Learned

In order to assure feasibility of the new requirements, one program employed prototyping of these requirements with systems involved. Programs also established IPT(s) to address interface requirements and feasibility. As part of this IPT effort, agreements were established among government agencies involved to specify responsibilities and to provide an infrastructure for resolving issues and conflicts.

The prototyping of new requirements with the existing legacy system was successful in demonstrating the technical feasibility of integrating new requirements. The technique also helped to reduce the cost of the effort and it mitigated risks associated with the integration. On the other hand, the IPT(s) did not work as well as expected because of competition among participants and lack of common goals. This can be partially attributed to the fact that the Government had not established a business case for involved contractors to work together.

Recommendations

It is recommended that analyses be performed to ensure technical feasibility of software integration with legacy systems before the imposition of new requirements. This would help to resolve technical issues up front, especially in the case of commonality where the same technical solutions could be used for all the programs. The use of prototyping in determining technical feasibility of integrating new requirements is considered an excellent way to implement the analyses.

If technical working groups (IPTs) are used to work out technical issues, these groups should be empowered to enforce decisions on the programs. If contractors are participating in these groups, then the government must establish partnerships (buy-in) between all the working group members. It was also recommended that an IV&V process be employed to provide unbiased assurance of technical feasibility.

Requirements Traceability

Tracing of requirements is a critical aspect of requirements management. Tracing requirements helps to ensure that the system design effectively and efficiently meets the users' needs. The trace of requirements provides rationale for prioritizing and justifying decisions on the design.

Approach and Lessons Learned

In some of the programs a requirements-management tool was utilized to trace requirements. The tool was used to centralize and share the requirement trace and to identify interdependencies among requirements for the contractor, program manager (PM) and users. In addition, the tool was used during periodic software-requirement reviews to verify requirements and the trace with all stakeholders.

The requirements-management tool provided more visibility into the requirements. Stakeholders were able to get a better understanding of the requirements and the system. Tracing the requirements and sharing the results with the stakeholders mitigated adverse schedule and cost impacts.

Recommendations

The group recommended that all programs use a requirements-management methodology or tool, and establish a requirements-tracing process. Such a process would include the sharing of results with all stakeholders of the system and would have a mitigation mechanism in place to accommodate requirement changes.

Planning and Estimation

Software Integration Planning

In order to be successful in obtaining approval of programs and funding, program managers often are forced to commit to success-oriented planning and schedules. Such plans do not allow flexibility to mitigate program risks. This is especially true when new requirements are imposed to effect integration across the battlefield with directed deadlines. Subsequently, these schedules are “forced” onto contractors who are developing or enhancing the system software and simultaneously trying to implement new requirements associated with the integration. This situation causes both government and contractors to sacrifice or compromise processes, it adds risks to the overall program, and it results in lower-quality deployed products, especially software. It also leads to a lack of synchronization between software release schedules.

Approach and Lessons Learned

To satisfy the success-oriented planned milestones or directed deadlines, sequential processes were often performed in parallel (e.g., contractor and government testing). This makes synchronization of events difficult. In some cases the times to effectively perform processes, such as testing, were reduced. In other cases, steps in established processes were modified significantly or completely eliminated. One program planned a “cleanup drop time” in the

schedule to accommodate schedule constraints and changes. On the other hand, there was one program that was allowed to slip its schedules.

Even with the approaches outlined above, the results were the following:

- degraded system/software product quality
- increased rework
- increased errors
- delayed cost until later (always costs more)
- burnout of people (they worked long hours and quit)
- cutting of corners (didn't do all of the things that should have been done)
- use of chicken wire and duct tape solutions

Recommendations

Programs should make process changes as difficult to make as schedule changes. Programs should plan for "cleanup drop time" for contractors and government similar to the way that companies such as Microsoft plan at least one third of development time as a cushion for contingencies.

Inflexible Schedule

Arbitrary target dates for fielding are sometimes established without sufficient analysis on the impact to program schedules or required effort to meet the date.

Approach and Lessons Learned

One approach to ameliorate schedule constraints was to reduce the scope of the functionality to be provided and still meet the fielding date. This leads to planned incremental upgrades to incorporate all the functionality at later dates. This approach did not succeed because it would not satisfy the schedule or provide all the functionality the user wanted.

A second approach is to implement a new technical solution to satisfy the functionality on the date imposed. This approach of taking an entirely new technical approach late in the system upgrade did allow the Initial Operational Test and Evaluation (IOTE/FOTE) schedule to be met.

Recommendations

Perform an “up front” analyses by all players to determine the feasibility of the technical solution(s) providing the necessary functionality and performance while satisfying the imposed schedules. These analyses would be provided to the decision makers who set the schedule(s) and would allow a more informed decision concerning the set dates. It was also recommended that an independent group conduct such analyses to provide an objective view and preclude any program bias.

Planning for Integration and Test

Most programs pointed out that there was inadequate planning for spares to support the system/software integration environment—including prototype hardware/production hardware and the actual platforms. In some cases there were requests to use systems (and associated resources) being tested that were not included in the test planning. In some cases these requests could not be honored.

Approach and Lessons Learned

Programs used the following techniques to resolve the issue of a lack of resources during testing:

- Develop a comprehensive plan to utilize system resources during all phases of the program, especially during testing. In this planning, programs anticipated unplanned requirements based on historical experience.
- Assign an asset manager to control and track assets.
- Schedule midnight shifts to satisfy requirements.
- Conduct continuous review and updating of plans.

The approaches noted above worked. However, it was felt that these should be improved upon and institutionalized. Even with these approaches, the eventual cost to the government increased, especially when early planning was not performed.

Recommendations

For the Integration and Test phase of programs it was recommended to

- develop comprehensive plans to include spares and test equipment for the integration environment
- anticipate unplanned requirements based on historical experience
- assign an asset manager to control and track assets
- conduct continuous review and updating of plans

Planning for Software Tools and Training (Life Cycle)

The use of and training for (life cycle) software tools and environments between the PM and the contractor helps in managing the overall program and facilitates the government's eventual use of the tools. The planning and implementation of this effort reduces overall program costs by minimizing duplication of training and coordination of schedules for tool usage.

Approach and Lessons Learned

Some projects coordinated software tool use during their programs. When the contractor and PM use the same software tools and environment, and share the same training for these tools, the program benefits through reduced cost.

Recommendations

Plan for the selection of software tools and environments and the coordination of their adoption and use. Plan for joint training for the contractor and PM on these tools and environments.

Estimating Software Project Cost

For most programs, available funding may not support the program office's estimates of costs of the software effort. The lack of availability of funding usually causes changes to the program plan. In addition interrelated programs develop their software cost estimates without sufficient coordination, resulting in conflicting estimates/cost for competing dollars. This is especially true for newly imposed interoperability requirements where unplanned test events for such interoperability were added to the program and were not envisioned during the original program-planning and cost-estimation process.

Approach and Lessons Learned

For most programs any shortfall in funds had to be" made up" from the reallocation of internal resources.

The primary lesson learned was that when programs are interrelated, especially for integration requirements, the integration cost estimates for each program must be developed, coordinated, and consolidated by all involved parties.

Recommendations

The group recommended that costs for interrelated programs, especially for integration efforts, be developed by an independent, umbrella organization. Such an organization would

help validate, manage, and approve all interrelated project-software cost estimates. It would also help interrelated programs to coordinate and synchronize estimates for system-of-system integration.

Organization and Staffing

Maintaining Adequate Staffing

There is a large amount of turnover of technical personnel in programs. This is true in the government as well as supporting and development contractors. In most of the programs this reflects not only the loss of technical knowledge but also the loss of corporate (e.g., program management) and domain-specific (e.g., avionics) expertise. In addition, there seems to be a lack of training for necessary skills, knowledge, and abilities for new hires, leading to an insufficient quantity of government personnel with appropriate skill, knowledge, and ability to manage the program and contractors.

Approach and Lessons Learned

Programs have tried to provide incentives to technical staff members to stay in the programs by improving the work environment, pay, and training. Another approach has been to employ matrix support from other government organizations and by contractors' augmentation of the staff (i.e., support contractors). However, as noted above, these support contractors are also suffering from the turnover and loss problems.

While these approaches have met with only marginal success, the lessons learned indicated that strong, proactive leadership was essential to retaining technical staff. Improving the work environment, providing workplace tools, and providing training has a positive impact in this area.

Recommendations

The recommendations from the programs were the following:

- Give program managers additional authority to provide incentives to employees by improving the work environment, providing workplace tools, and providing the necessary training.
- Identify paths for upward mobility within the programs.
- Identify transition employment opportunities for personnel assigned to legacy-system programs that are reaching the end of their life cycle.

Risk Management

Planning for Risk Management

Programs inherently have risks (both technical and programmatic) that should be planned for and monitored. Most programs were concerned that risk mitigation was not sufficiently rigorous to be effective.

Approach to Resolution and Lessons Learned

In most programs both the government and contractors track and manage risk through processes and supporting tools. Successful risk planning that implements a risk process should be accomplished “up front.” This ensures that periodic reviews and updates of risks are conducted in coordination with all stakeholders.

For some programs, risks and mitigation plans were identified early. This helped programs, including both government and contractor teams, to be better prepared to address risks. It enabled some of the more critical risks to be mitigated. However, in some cases mitigation options were eliminated when a risk-management process and techniques had not been planned or utilized.

Recommendations

Employ risk management to include continuous risk planning and mitigation.

Appendix B Systems and Software Working Group

This working group focused on systems and software, the aspect of the program that is concerned with the technical construction of the product.

The participants included the following:

- Ed Andres, Abrams
- Tim Carr, Apache
- Edgar Dalrymple, Bradley
- Ron Gormont, PEO AVN
- Fred Hansen, SEI
- Jim Linnehan, SA ALT
- Ed Naylor, Patriot
- John Parker, MLRS
- Tom Pollard, Bradley
- Dennis Smith, SEI
- Emory Steedley, Patriot
- Bob Topp, Apache

To initiate discussion, we considered systems and software through the following topics:

- systems engineering
- software engineering
- technologies
- architectures
- legacy systems
- target systems
- interoperability

These topics focused the discussion and stimulated consideration of additional areas. This section of the report is organized around the major categories that the discussion actually followed:

- legacy systems and obsolescence
- mandates
- interoperability
- government furnished equipment (GFE)
- commercial off-the-shelf (COTS) products
- architecture
- topics relevant to other working groups

Legacy Systems and Obsolescence

Since the theme of the overall workshop was software upgrades, many of this group's comments focused on legacy systems and their obsolescence.

A set of issues and problems was discussed, including

- logistics support for multiple versions
- scheduling upgrades
- what to do about obsolescent components
- how to deal with vanishing support for development tools

The negative effects of these problems include

- non-interoperability
- unit failures
- equipment downtime
- higher costs
- safety test is tough due to multiple configurations

Approaches and Lessons Learned

In discussing potential approaches to address the problems, example case studies were drawn from the experience of the programs. These included the following:

- Upgrades to one system specified the inclusion of a switch for a new component on an already-full panel. The developers were able to utilize an existing switch for old equipment that would not be used when the new component was installed. The software was

modified to check the hardware configuration in order to determine which component was present and to control it appropriately in response to the switch. This solution was possible because of a robust and modular architecture. However, the solution did by-pass standards.

- In order to try to field units with uniform configurations, one program required that every engineering-change proposal Engineering Change Request (ECP) have full funding for all changes. This funding was, in fact, provided. In practice, it was not possible to make all the changes simultaneously. In some cases, the fleet contained diverse variants for up to five years.
- Lack of team management and coordination can lead to more complex systems than necessary. In one case, the hardware developers "dumped" onto the software developers "a bunch of ill-designed boards." Instead of being in a compact array, the input-output (I/O) addresses were scattered across a space. So instead of a tidy iteration, the program needed a sequence of ad hoc code.
- A legacy system had organized its processing around a fixed cycle of 20 milliseconds. However, with additions to handle upgrades there was no longer time in the cycle to handle all the computation if there were five or more items under control at a time. To research the problem, the team studied the bus traffic, plotting its use by each task over a series of cycles. The team discovered that some tasks, for instance keyboard response, did not need processing at that cycle rate. It was possible to find a list of such tasks with lower requirements and process them less frequently. Thus, "decomputing" the frame times eliminated the need to upgrade the software.

These examples provided the following set of potential approaches for executing upgrades:

- Require the contractor to have a plan for obsolescence and a group to deal with obsolescence.
- Develop planned upgrade cycles to target obsolescence of hardware and software.
- Address tool support as a major effort.
- Program for block upgrades to tool support.
- Add electronically accessible identifiers to all hardware and software components. Use them to automate reconfiguration based on component presence.
- Plan for change and expansion. Budgets need unallocated funds, schedules need unallocated time, physical space should not all be filled, interactive computing tasks should be planned to complete in less than the desirable response time, and computer resources such as memory, computer power, expansion slots, and disk space should not be totally utilized.
- Load software from a CD-ROM having multiple software versions. Choose the version depending on what other components are present.
- Plan to rebuild every *k* years. A good value for *k* is three.
- Recognize "architectural decay." If fixing one problem causes three more, it may be time to re-architect.

- When planning an upgrade, do systematic reuse analysis to determine what components to replace. Reject an old software component if it does not meet modern coding techniques and standards. The steps to an analysis process are the following:

– Look at the code and identify problems.
– Use automated source-code analyzers.
– Identify potential solutions.
– Re-architect the software if necessary.
– Code and test.

- Plan for regular upgrades. This includes the following tasks:

– Target the obsolescence of hardware and software.
– Upgrade line replaceable units (LRUs).
– Exploit lessons learned.
– Manage the process with weekly integrated product team (IPT) meetings.

Recommendation

- Train acquirers to proactively address legacy systems.

Mandates

A "mandate" is a system design decision that comes not from the needs of the users, but from a higher authority. Mandates come in several forms.

- Standards mandates dictate that certain specified standards must be followed for some specific tasks.
- Government furnished equipment (GFE) mandates dictate the incorporation of some specific component, usually software, into a system. These are covered in the next section.
- Process mandates dictate certain sequences of actions and the generation of certain artifacts.

Mandates offer a range of problems to programs. Among them, loss of control—or at least perceived loss of control—can be salient to program managers. This loss magnifies other problems raised. For example:

- When mandates are handed down subsequent to the project's design, they impose significant rework, raising costs and challenging or destroying the schedule.
- Mandates must sometimes be met with short-term workarounds that must be undone later in order to extend the system.
- Some mandates prescribe a solution developed by or for the DoD in competition with commercial alternatives. In later years, however, there may not be funding for updates to the solution that were contemporaneously occurring with the alternatives.
- Mandates require resources for analysis of their applicability, regardless of whether they are, in fact, relevant for a program.

Approaches and Lessons Learned

Case studies of mandates include

- documentation standards switch from DoD 2167 to DoD 4943. The question was raised as to why acceptable documentation should be rewritten.
- compliance with joint technical architecture (JTA) is mandated, and yet compliance is not well-defined. A large document is available with a number of potential solutions that sometimes conflict and that are subject to alternative interpretations.
- language mandate change. The contractor was funded to write a converter, convert the software, and write a compiler from the new language to a particular piece of hardware. When the hardware then became obsolescent, the tools were no longer available because the contractor could not afford to keep the tool team in place.

Approaches for dealing with mandates include

- request waivers
- use translators
- rewrite to avoid mandate
- wait to see if the mandate changes
- obey the mandate

The following outcomes were reported in response to the Ada mandate:

- One translation was completed, but was unsupportable, and was abandoned.
- Another translator is currently working and supportable.
- An Ada system that complied with the mandate now feels discrimination.
- Another system migrated away from Ada.
- Because of the size of the existing code, one group is staying with Ada.
- One group needs to change its processor, but must abandon Ada because there is no support for Ada on the new processor.

Lessons learned from dealing with mandates include the following:

- Revise plans to deal with mandates. Describe to both the mandating agency and to clients the effects of the mandate on outcomes. Will schedule slip? Will fewer units be delivered to the field? Will safety be compromised?
- Apply for a waiver. Waivers are usually the first resort of a program manager. However, mandates often address problems that must be solved on a level wider than the individual program. A waiver can be a short-term solution, but a potential long-term problem.
- Be prepared to determine and clarify the effects of a mandate, such as reduced number of fielded units or decreased safety.

- Request a meeting forum in order to deal with the problem that the mandate is to solve.

Recommendations

Going beyond steps that programs can take, the work group members offered the following recommendations to alleviate the problems posed by mandates:

- Evaluate mandates with a bottom-up consultative process to be sure the full effects on the field are understood.
- Request studies of the impact of mandates.
- Specifically obtain estimates of the schedule, cost, and other impacts of the mandate from representative programs.
- Recognize that some industry standards are more real than others. Predicate the adoption of standards upon the degree of adherence actually found. The most effective standards are those that are already in use before their adoption.
- Do not impose a particular processor or programming language. These are particularly subject to market forces and potential obsolescence.
- Use problem-focused forums instead of mandates.
- Have mandates interpreted by a government officer rather than leaving the interpretation to vendors.
- Mandates should be defined in such a way that the contractor can match them against other constraints. One good way to do this is to define the mandate "at the boundaries of the box." That is, do not specify how something is done, but specify the interface that must be provided.
- As a result of acquisition reform, contractors are told what to do and not how to do it. The participants recommend that management should extend the approach to programs by specifying the ends that programs must accomplish but not specific direction on how to achieve those ends.

Interoperability

The need for interoperability was a major concern. Lack of interoperability renders command-and-control systems useless for accomplishing mission requirements. Plans have been discussed to prescribe new interoperability mandates. However, program managers are reluctant to make the changes without conducting careful study of the cascading effects on the integrity of their systems. The issues can be summarized as follows:

- If some systems do not provide interoperability, then information will conflict.
- Interoperability is difficult for legacy systems.

Approaches and Lessons Learned

Example case study:

> One exercise was conducted using a single communication system shared between voice and data. This particular system does not incorporate an automated means to allocate the bandwidth of the system. As a result, voice communications overwhelmed the channel to the extent that computer traffic, especially the command to fire a weapon, was impeded. The results were blamed on the program instead of on the communication channel.

No approaches specific to interoperability were expressed. Instead, interoperability was treated as just another mandate. Frequently, waivers were justified.

Recommendations

The recommendations about mandates included the following:

- Get platforms involved in decision making.
- Leave implementation details to the platform.

Government Furnished Equipment (GFE)

GFE is often developed by a contractor under a government program and subsequently provided to other programs for integration as a subsystem or component within their systems.

GFE raised the following problems in both development and support of systems:

- Late arrival of GFE items may force schedule slippage.
- When the GFE does arrive, its behavior can conflict with its description. This results in complexity and frustration during system integration, as well as schedule slippage.
- Personal and corporate relations between a program's own contractor and the GFE contractor can deteriorate, exacerbating other problems.
- The architecture of the GFE may conflict with that chosen for the system. This is especially a problem if the mandate for the GFE arrives after development begins in earnest.
- The GFE vendor may have its own agenda for fostering the use of particular proprietary items.
- A program may inherit a high dependency on the GFE vendor, without fall-back options.

Approaches and Lessons Learned

A GFE case study was described as follows:

> In order to incorporate a COTS product into a host platform, a separate processor was added. A switch was also provided to swap use of the single display between the tank and GFE computers. The use of the desktop COTS system was judged by the program manager not to be "a nuclear hardened solution."

Lessons learned include the following:

- Make a separate contract with the GFE contractor for support of the program.
- Install a liaison person at the GFE contractor site.
- Install a separate computer to run GFE software.
- Allow ample time for integration of GFE into the system.
- Make the GFE vendor part of the development effort.
- Have a block upgrade program for GFE.

Recommendations

- Allow interface compliance as an alternative to incorporating the GFE itself. If a system behaves as though it contains the GFE, there is not a reason to favor it less than a system that actually does incorporate the GFE.
- Have a block upgrade program for GFE.

Commercial Off-the-Shelf (COTS) Products

COTS software promises a number of advantages to military systems. COTS products can provide benefits such as reduced costs through shared overhead and higher quality through wider usage. Participants cited several important technology advances made in the commercial sector: fiber optics, VMEbus, and graphical maps.

However, there are also strong caveats to take into account when using COTS. These include the following:

- One size does not fit all. Embedded systems have very different requirements than desktop systems do.
- COTS is not necessarily faster, better, cheaper.

COTS systems offer the following challenges:

- supportability
- compatibility
- control of performance
- security
- safety criticality
- intellectual property. Someone owns the code and may change the conditions for its use. There is license uncertainty, license-management overhead, opacity of proprietary code, and potential non-availability of a new version’s memory images.

- excessive (and often growing) resource consumption (memory, disk, central processing unit)
- robustness: frequent crashes, non-existent error recovery strategy, and poor error logging
- fragility of the hardware: sensitivity to vibration, heat, and shock; the impact on software
- longevity: new versions appear frequently; software companies can go out of business

Approaches and Lessons Learned

COTS case studies included the following:

- A COTS hardware piece costing under $100 was selected to solve a particular problem. Unfortunately, it needed certification for its military use, so the vendor was put under contract for the part. After two years, the cost of the part, still physically unchanged, had risen eightfold. The part is currently being replaced.
- Software running on a COTS operating system has been deployed in a weapon system. It suffers from
  - user interface deficiencies
  - insufficiently robust operating system. A software halt can kill a weapon system.
  - not being nuclear hardened
  - long power-up and power-down times
  - poor response to power outage (operators often “pull the breaker” to reset the system)

COTS lessons learned include the following:

- Eschew proprietary code or buy rights.
- Put the COTS vendor under a support contract. This approach offers reduced schedule risk, collaborative debugging, and more immediate resource availability.
- Perform a source/documentation/test case review
- Give the contractor the right to choose COTS and the responsibility to make it work.
- Use an independent evaluator to examine the candidate COTS software. At a minimum the independent evaluator should visit the vendor facility and review the source code, documentation, and test cases coverage.

Recommendation

- Do not over-mandate the use of COTS.

Architecture

Pay attention to the hardware and software architectures of embedded platforms.

A number of constraints affect the architecture task, including

- the need to address the primary quality attributes, such as availability, performance, security, safety, and maintainability
- the need to allow for evolution to support requirements changes and changes in other constraints
- environmental constraints, such as space, power, or heat
- standards and mandates
- preparation for software maintenance; especially system-construction tool support
- hardware obsolescence
- COTS software upgrades and obsolescence

Approaches and Lessons Learned

The discussion generated a set of lessons learned, including the following:

- Involve all stakeholders: hardware and software developers, testers, users, and management.
- In planning the architecture, design for change. Expect that things will change over time and that individual pieces of the developed system will have to be replaced.
- Incorporate interfaces between components to promote their independent replacement.
- Isolate groups of functionality.
- Analyze tradeoffs between competing requirements.
- Exploit open systems.
- Recognize the key role of throughput and memory.
- Recognize up front the differences between desktop and embedded systems.
- Recognize that architecture gets constrained quickly.
- Use simulation and prototyping. As in spiral development, choose a methodology to minimize risk.
- Plan to be able to change the programming language. Write code in a small, stylized subset of whatever language you do use.
- Plan for expandability by incorporating spare capacity.

Study the application of methods and tools. For example, unified modeling language (UML) with tool support has helped one program in defining architecture level requirements. It made digitization simple and allocated timing constraints with it. But it also automatically generated a large number of documents that required careful analysis and did not always map to the needs of the program.

- Place control of system design clearly with either the government or contractor.
- Define operational architecture and review it first, especially with users.

- Use a software layer to isolate from the hardware and provide a leverage point for future replacement software.

Topics Related to Other Working Groups

During the discussion, a number of issues were addressed that had relevance for other working groups. The main points of these discussions are recorded here. The issues and concerns were consistent with the discussions that took place in the other groups; the Systems and Software group added additional insights from its own perspective.

Funding and Contracting

Funding was discussed from the perspective of software and systems. Funding problems include the following:

- Uncertainty results from changes of federal government administration.
- Mandates will often impose additional costs on a project without offering funding to cover those costs.
- If there is a mandate for the use of a component, funding for sustainment of that component is sometimes withheld.
- Declining funding levels prevent retention of expertise.

Approaches and Lessons Learned

Lessons learned include the following:

- Recognize the difficulty of upgrades when arranging for funding. Revising software can mean relearning everything that the original developer understood when writing the original code.
- Always calculate the logistics costs. Include costs for new part numbers, new manuals, retraining, and multiple similar parts in inventory (e.g., space, increased error rate).
- Remember that a replacement component offered as an upgrade may well meet performance specs, but it may also create logistics problems.
- Big contractors can afford to retain tool expertise, but small contractors cannot. If you have software from small contractors, consider keeping the supplier under contract in order to support their continued expertise on essential tools.
- An in-house mentoring approach can be valuable.

Recommendations

- Provide resources as necessary to adhere to mandates.

Project Management

Project management problems include the following:

- There may be insufficient insight into the contractor's efforts.
- Optimism can lead to aggressive and unrealistic schedules.
- There is no consistent mechanism to resolve issues of conflicting mandates or inter-project issues.

Negative effects include

- schedule disruption
- integration difficulties
- failure to interoperate
- confusion

Approaches and Lessons Learned

The following two case studies were discussed:

- A replacement board had to be accommodated. All parties knew that it had been time consuming to integrate the original board. The engineers believed that the new board would be little better, but management overrode them in the belief that the prior experience would help. It didn't. The schedule target was missed. One way to look at this is as a failure to use prior data to hone present estimates for future work.
- One project arranged with a contractor to place a person from the program-management office at the contractor site. This individual became a valuable conduit for communications in both directions and across levels. He established a reputation for integrity and discretion and became a confidant of various contractor personnel.

Lessons include the following:

- Use data and metrics analysis. At first, when there is no historical data, this approach is painful. Later, as data accumulates, it becomes highly rewarding.
- Programs should always strive to defend themselves with real data.
- Exploit the fact that the DoD requires vendors to have attained CMM level 3. This is helpful because it requires vendors to have metrics. Demand to see those metrics and use them to manage the program. However, vendors may try to tailor their processes for your project to exclude metrics.
- In demanding metrics values and process documentation from major contractors, recognize that they, in turn, will be very demanding of the same materials from their subcontractors and suppliers.

- Recognize and discount the optimist factor. Programmatic people tend to be more optimistic than engineers, especially in proposing schedules. Decision makers are sometimes tempted to use the most optimistic, rather than the most realistic, estimate.
- Retain all schedule estimates and reward those managers who come closest, whether their approach was accepted or not.
- As with funding, recognize the difficulty of upgrades when doing scheduling. Revising software means relearning everything that the original developer understood when writing the original code.
- Utilize independent validation and verification (IV&V) support from either the DoD or the contractor.
- Send a liaison person to work at the contractor site.
- Demand that software managers respect memory and throughput requirements. Too often they don't see these as real and constraints are haphazardly exceeded.
- Hold inter-organizational meetings to resolve issues.

Recommendations

- Publish guidelines for reviewing metrics.
- Have clear criteria for which guidance and mandates to follow.
- Have an authority with the power to make decisions.
- Have readiness-driven schedules.

Requirements

While mandates arrive at the program manager's office from "above," the requirements are the project definition and arrive from the users "below." Traditionally, requirements are created as a thick document that changes frequently throughout the project.

Acquisition reform has changed many of the ground rules relating to requirements. It has eliminated many documents and has led to an emphasis on performance specifications. While this has streamlined part of the process, it has also led to the abandonment of many of the controls that program managers need to monitor a program effectively.

Problems in the requirements process include the following:

- Requirements can be difficult to determine and hard to gel.
- Requirement additions arrive frequently.
- Satisfaction of new requirements often translates into a need for higher data speeds.
- Requirements creep is exacerbated by email.

- Requirements traceability can be difficult with acquisition reform. It is not always possible to be sure that a system satisfies requirements or that a particular part of a system is necessary to satisfy the requirements.
- Requirements documents become very large and lead to very large design documents.
- Regression to earlier milestones may be required to meet interoperable software requirements.
- Requirements can be written with too many details.

The negative effects of these issues include the following:

- Late changes have schedule and resource implications.
- Late changes can lead to suboptimal solutions.
- Designs may need to be scrapped to accommodate late-arriving requirements.
- Large documents are very difficult to check.
- Excessive detail constrains the design and severely limits COTS options.

Approaches and Lessons Learned

The following case study was presented:

> As a workaround for a late-breaking requirement, a GFE modular software package had to be modified to borrow RAM (random access memory) across the VMEbus. This is a slow access path and it hurt performance.

Lessons include the following:

- Use system segment design documents; they are voluminous, but well organized.
- Require change request documents.
- Have a formal change control board and make sure it performs.
- Demand that each change be accompanied with a full cost analysis.
- Discuss each change in a formal forum, such as the change control board.
- Allow changes to be made only with written approval.
- Approve change requests with signatures at commitment meetings.
- Use performance specifications. This form of specification requires a new mindset different from the detailed specification mode.
- When appropriate, use methods such as UML with tool support to describe requirements.

Deployment

Deployment problems include the following:

- Requirements to simultaneously field different interoperating systems can create problems of complexity.
- Delivered product is not documented or documentation is completed late. If documentation is completed early, it may be inaccurate and never get updated.
- Automatic generation of documentation creates large documents of questionable value.

Approaches and Lessons Learned

Lessons include the following:

- Require electronic submission of documents.
- Require early access to preliminary documents over the network. Review them early and often.
- Choose document formats carefully. Portable display format (PDF) is good, but should not be expected for early documents. Whatever format is chosen, be sure to have funding for enough people to be able to read it.
- Write requirements only to the level of detail that is needed.
- Require a software baseline verification test (SBVT) because it includes all documentation. One program ensures compliance by refusing to progress to air worthiness release (AWR) until the SBVT is accepted by the computer resources management team (CMRT). However, this approach requires discipline from an acquisition organization.

Individual Summary Recommendations

In the final session, each participant was asked to recommend one change that would be a top priority "if I were king." Many of these echoed other issues brought up during the discussions. While these comments do not necessarily reflect a group consensus, and in fact several of them contradict others, they are recorded here as being highly salient in the mind of at least one participant:

- Reinstate ADA.
- Get rid of mandates.
- Do not allow GFE.
- Carefully choose a time to declare that requirements are stable. Stop the changes and build the system. Defer all changes to later upgrade cycles.
- Exploit the fact that the DoD requires vendors to have attained CMM level 3. It is helpful because it requires vendors to have metrics.
- Have an authority with power to make decisions.
- Be sure that there is bottom-up consultation on mandates.
- Provide more inter-program meetings.

- Resolve issues clearly and in a timely manner.
- Require that all decisions be based on data and analysis.
- Provide mechanisms to resolve conflicting mandates.
- Government management should listen to its workforce as much as it listens to industry.
- Train military officers to question software decisions.

One participant listed the following five factors that are critical to the success of programs:

1. experienced senior staff. It is particularly important to have a domain expert as the PM.
2. mature processes in place. These especially include requirements management and control.
3. proper development and test environment resources
4. a propensity to plan for change, including strong risk-management processes
5. clear communications with the customer. It is essential to tell the customer clearly the limitations of what will be delivered with the funding provided.

Appendix C Funding and Contracting Working Group

The participants included the following:

- Nell Baites, MLRS government acquisition
- David Fogg, Patriot government acquisition
- Mike Patterson, Abrams contractor
- Neal Patterson, MLRS government
- Bob Pusicz, Apache contractor (Decilog)
- Caroline Graettinger, Software Engineering Institute
- John Bergey, Software Engineering Institute

The Funding and Contracting Working Group explored the following five topics:

- software contracting scope
- contractual planning
- decision making on cost and schedule
- contract performance
- government/contractor relationship
- For each topic, the group identified one or more issues, and their outcome, impact, and lessons learned. The group then made recommendations to address the issues.

Software Contracting Scope

System Versus Software Emphasis

Even though systems are software intensive, the current acquisition paradigm does not reflect the importance of software to the acquirer. In particular, an emphasis on the system at the expense of software is related to the following issues:

- Contractual arrangements purchase capability rather than software.

- Software is treated “in the small”—it is not possible to put the detail in the contract (e.g., statement of work (SOW) does not accurately describe what we want; we are required to cut out deliverables, etc.).
- Software requirements and costs are small compared to total platform costs, leading to software concerns not getting sufficient attention.
- Not enough up-front money is allocated to software.
- Software gets funded at the end when money and schedule are tight.

As a result, software is often treated as an implementation-specific detail. The Army does not obtain enough visibility and leverage to properly manage the software aspects of an acquisition.

Approach and Lessons Learned

There were two primary lessons learned. First, paying attention to the software from the beginning of a program is important because “software implements the system performance.” If you don’t do sufficient software planning up-front, Murphy’s Law will catch up with you in the end. Second, software problems are often not solved in a timely fashion; instead they get deferred to a follow-on contract. For example, requirements are often not definitive enough to allow project personnel to accurately quantify and describe the contractual tasks. As a result they tend to specify place-keeper tasks because it is not possible to specify level-of-effort contracts.

Recommendations

This problem can be addressed through education and training. The group recommends education and training in applying earned value, IPTs, team work, and software acquisition management. The latter would include how to use performance specifications to get what you want from software. Potential agents responsible for carrying this out include the ASA (ALT) or program managers.

COTS Introduces a New Set of Contractual Issues

The widespread use of COTS introduces a new set of contractual issues—some good and some bad. This is due to the relatively short COTS-technology upgrade cycle and the long system-development lead times. A project cannot get to operational test/introduction of operational capability (OT/IOC) without significant rework to address software obsolescence of COTS products that are part of the system.

Approach and Lessons Learned

Lessons learned suggest that projects need to be more proactive and plan block upgrades every three to five years to address

- obsolescence
- new functional requirements
- non-supported COTS

In addition, the experience of working group members suggests that an effective way of dealing with software obsolescence is to bring in contractors having special expertise in combating COTS software obsolescence.

Recommendations

The group recommends that there be a level-of-effort (LOE) funding line in development contracts for continuing electronic and software enhancements. Tasks to offset hardware and software obsolescence would be defined as needed. The group believes Program Managers should be responsible for funding such a line item in all programs involving use of COTS.

Contractual Planning

Mandates from Above Are Problematic

Mandates from above are almost universally viewed as a major problem. The classic example was the Ada mandate. But current mandates are viewed with the same concern. Current mandates include such items as

- joint technical architecture (JTA)
- performance specifications instead of detailed specifications
- warranties on software

The lone exception is the CMM level 3 requirement. Although it is viewed as being “painful,” the demonstrated benefits are worth it from the perspective of both the contractor and the program manager.

Approach and Lessons Learned

The working group identified a series of mandates that had an adverse effect on their projects. These include the following:

- There were differences between the performance requirements and the detailed specifications being mandated. This has benefits for hardware but not for engineering services or software. It results in
  - the lack of a recognized approach to ensure delivery of the software that fulfills mission needs
  - full control over the software development by the contractor
  - limited visibility and control by the government
  - an inability to achieve a graceful transition of software from one contractor to the other for post-deployment software support (PDSS)
- Mandating fixed-price contracts for large-scale development has not been effective.
- Warranties for software do not work, for the following reasons:
  - Software warranties are not cost effective. They apply to specific test plans and procedures but are not warranted under any other conditions.
  - Once the software is tested against the performance specification and accepted, all changes are costly.

Software warranties are viewed as being contentious and contributing to an adversarial government/contractor relationship. The group's experience suggests that money used to obtain a software warranty would be better spent by applying it to an engineering-services contract instead. Problems could then be fixed in accordance with the government's priorities and specific needs.

With regard to mandating fixed-price contracts, experience suggests that when a contract is cost plus and level of effort, the contractor will do whatever it takes to get the job done. The group believes the following contract mechanisms should be used:

- cost plus contract / earned value for new development
- cost plus contract / level of effort for maintenance

Recommendations

The working group recommends that all agencies do away with mandates for software warranties, and give the program managers the authority to decide if the mandates are applicable.

To address the systemic nature of the problem, the group believes that acquisition reform should be revisited to consider the full implications of adopting performance specifications.

Software Data Rights

In many situations the government should own software data rights in order to transition to post-deployment software support.

Approach and Lessons Learned

Specific lessons learned include the following:

- Don't let contractors put blanket proprietary labels on software deliverables because the acquirer needs data rights and this may preclude obtaining government rights.
- Don't let contractors incorporate independent research and development (IRAD) money or they will claim total ownership of software data rights.

Recommendations

Increase awareness about software data rights issues, provide guidance, and empower programs to enforce follow-through, especially where subcontracting is involved.

The government should obtain "government use" data rights to the software so that it has the means to perform and compete, if necessary, software life-cycle post-deployment support. Otherwise the development contractor will have a "lockout" on post-deployment software support.

Decision Making on Cost and Schedule

Cost and Schedule Estimation Are Problematic

The group discussed how to best estimate cost and schedule for software development and maintenance. This includes the following questions:

- How do we estimate cost per line of code (LOC)?
- How do we estimate cost per drop?
- Can there be a database or tool to provide hours/LOC for negotiation purposes or hours/LOC by ground-based systems or avionics or another domain?

Approach and Lessons Learned

The group's understanding is that there are no Army standards for cost estimation and this leads to arguments in planning contracting efforts. Some of the lessons learned include the following:

- Plan for quality but realize you will implement to schedule.
- Costing tools are available but there is a lack of community awareness.
- Make initial estimates using models such as COCOMO, Price-S, and SLIM. If the contractor's estimate falls within this range, it provides some independent verification that the estimate is realistic.

- Bring lessons learned back into the cost-estimation models.
- There should be greater effort dedicated to planning and estimating.
- Using cost and schedule data derived from historical data through working with a contractor is an effective workaround, but such data are not always available.

Recommendations

The government should provide training and make cost estimation tools and historical data widely available to support the cost and schedule decision-making process. A concern is that proprietary issues can limit capturing the appropriate data.

Contract Performance

How Do I Manage Contract Performance (When Schedule Considerations Always Predominate)

A problem cited by all the participants is that in day-to-day practice "schedule is everything." This makes it more difficult to properly manage contract performance.

Approach and Lessons Learned

Schedule, quality, and cost considerations (in that order) are understood to define acquisition priorities. Various approaches that have met with marginal success in this highly schedule-driven environment are captured in the following lessons learned:

- Manage by incremental builds and track by milestones if using level of effort (LOE).
- Use an integrated product team (IPT) approach or hold technical interchange meetings (TIMs) to track software progress.
- IPTs help but they should be given the power to make schedule and quality decisions within contract scope.
- Sharing of information at lower levels is key to success.
- Hold the contractor to its process by reviewing the contractor's internal quality audits.

Recommendations

The group had the following recommendations:

- Extend current CMM requirement for all acquisition capability (ACAT) 1 programs to all software contracts; hold contractors to their processes; and periodically reassess their capability.
- Improve the software acquisition capability maturity level of acquiring organizations.

Contractor/Government Relationship

No Significant Issues

Both the government and contractors rely heavily on the use of integrated product teams to manage the acquisition and perform their contractual tasks.

Approach and Lessons Learned

The single most important lesson learned is that effective use of IPTs mitigates many potential problems. The IPTs are effective because they provide

- communication, communication, communication
- regularly scheduled technical-interchange meetings
- longstanding relationships with contractors and subcontractors

The degree to which IPTs are embraced by the contractor/government community is evidenced by the fact that contractors involved in IPTs often establish subcontracts with one another to resolve problems and fulfill tasking (which overcomes the limitations of a stovepiped acquisition infrastructure). This is another good reason for using level-of-effort (LOE) and cost-plus contracts. Allocation of ample funds to cover IPT activities is also an enabling factor.

Recommendations

There were no recommendations. The participants place high value on the existing contractor/government relationships because they believe a good relationship is essential to their mutual success, especially in light of other factors (e.g., poor requirements management, schedule perturbations, funding constraints, etc.) that are not under their direct control.

Appendix D Deployment Working Group

Participants included the following:

- Bill Anderson, SEI
- James Cyr, MLRS
- Don DeHart, Patriot
- Jim Kelley, Apache
- Ron Lafond, Apache
- John Markovich, Abrams
- Ray Obenza, SEI
- Mark Willhoft, Bradley

The topics covered by the deployment group included the following:

- testing
- integration
- configuration management
- operational suitability
- migration planning
- training

The deployment group assigned a number to each lesson. These numbers were used later in ranking recommendations. The results of this exercise are reported in Figure 2.

Testing

Interdependencies in Testing (Lesson #2)

The ever-increasing interdependencies of subsystems in systems, and of systems in platforms, have made optimization of test programs difficult. This problem has led to redundant testing, unplanned resource allocation, missed milestones, and testing delays.

Often testing of one system requires involvement of another system or program. If the second program has not planned for this coordinated testing, the testing will be delayed. The problem cascades when the second program enters a testing phase that requires the involvement of the first program. Then the first program will often delay the testing of the second program.

Approach and Lessons Learned

When the test schedules of multiple programs are aligned, it enables the coordination of all related resources and schedules. Such an alignment also requires a contractual basis for cooperation between the contractors.

Recommendation

The group recommended that alignment of test schedules be considered early in the planning of a software upgrade program. The plan should include mechanisms to ensure participation by each of the contractors.

Compromising Testing When Cutting Back Schedules (Lesson #3)

Pressure on deployment schedules has created a tendency to compromise the development process. Because of its presence at the end of the life cycle, and because it is difficult to quantify its specific contributions, testing is often cut back when schedules are compressed or budgets are cut. As a result, there is inadequate testing to ensure that the initial modification kits function correctly.

For example, in some cases, a contractor may not be quite ready for testing. Because of schedule pressures, the government person is sometimes pressured to allow tasks that the contractor has not completed, with the contractor's assurance that everything will be fine upon release of the modification kit. This situation creates risk because the modification kits might not quite fit or fully function.

Approach and Lessons Learned

The grouping of modifications into block modification kits facilitates the testing of multiple modifications simultaneously. This approach has been used successfully in several cases.

Recommendation

The team recommended a more disciplined approach that will not compromise testing. A minimum testing standard could be established to ensure that adequate testing is performed.

Production-Configured Test Assets (Lesson #4)

The lack of production-configured assets can lead to a number of problems including testing delays, hampering of early problem identification, increased risk, slipping of program schedules, and cost overruns.

Approach and Lessons Learned

These problems can be averted when sufficient dedicated vehicles for each configuration are built into the schedule. When sufficient vehicles were available, testing delays were avoided and problems were identified early.

Recommendation

The team recommended that dedicated test assets, separate from lab development assets, be budgeted into programs.

Test and Fault Isolation (Lesson #6)

When testing a complex system, it is often time consuming to develop a configuration of the parts that are required for testing and fault isolation.

Approach and Lessons Learned

One program developed an automatic get list generation of all parts, tools, and documentation required to resolve a fault condition. This resulted in a significant decrease in mean time-to-repair (MTTR), thus improving the maintainability of the system.

Recommendation

Determine how to make this specific practice applicable to other programs. If it is widely applicable, disseminate the practice widely.

C3 Interoperability (Lesson #7)

The requirement for interoperability of C3 programs has resulted in a need for testing that spans multiple systems. The planned addition of aviation platforms gives added priority to this need.

Approach and Lessons Learned

Interoperability has been demonstrated across ground platforms. The mechanisms for achieving integration include integrated project teams (IPTs), system integration labs (SILs), and central technical support facility (CTSF).

Recommendation

Interoperability testing should be addressed at all levels of test plans, beginning with the high-level test & evaluation master plan (TEMP).

Full Inter-Platform Stress Testing (Lesson #8)

Sufficient time and resources are often not allocated for inter-platform stress testing for full interoperability. Often the current approach is to wait for a major exercise, such as a war game scenario, to discover the shortcomings. This is not the optimal environment for testing because the reason for such exercises is to use the equipment, rather than to test its operation. This leads to expensive, high stress, last minute, fire drill scenarios.

Approach and Lessons Learned

There is a need to have high level plans and to implement these plans for inter-platform interoperability. Plans to address performance are especially important.

Recommendation

The team recommended that a higher level TEMP be developed, resourced, and scheduled into program schedules. The TEMP should include inter-platform, interoperability testing. This recommendation is similar to Lesson #7, but specifically addresses performance (e.g., can the communication pipes handle the demand of all systems at the same time?).

Breadth and Depth of Software Testing (Lesson #9)

Issues of breadth and depth are critical for adequate software testing.

Approach and Lessons Learned

Several organizations reported that automated test scripts reduce test time, increase repeatability, and expand test coverage.

Recommendation

The practice of automated test scripts should be widely disseminated, and there should be efforts to standardize test scripting tools.

Test Development and Repeatability (Lesson #10)

Test development can be a labor-intensive task. If it is not automated, substantial repetition of effort will be required every time a software upgrade is made.

Approach and Lessons Learned

The automatic capture and generation of a reusable test script from a recording of operator actions while using the platform has proven to be effective in saving both time and resources and in making the test process more efficient.

Recommendation

Automatic capture and generation of tests scripts should be broadly practiced among the programs.

Integration

Scheduling Dependent Subsystems (Lesson #11)

The last-minute availability of subsystems for integration and testing has led to delays, reliance on patches, repeat testing, loss of coverage, and has delayed problem discovery to the field trial phase.

Approach and Lessons Learned

Greater discipline in following the testing and integration process has led to positive results. A particularly difficult issue is to find ways for ensuring that subcontractors will produce their systems on time.

Recommendation

The team recommended stronger discipline in the entire testing and integration process, including:

- more integration testing
- freezing hardware and software configurations prior to integration

- not permitting schedule compression to eliminate testing

Lack of Standardization on Tools for Subsystem Checkout (Lesson #13)

A lack of standardization of tools for subsystem check out has complicated integration. Unique maintenance tool sets are redundant and expensive, and they complicate training.

Approach and Lessons Learned

The automatic capture and generation of a reusable test script from a recording of operator actions while using the platform has proven to be very useful.

Recommendation

Every effort should be made to standardize these support tools, and to make them available and current.

Proprietary Subsystem Software (Lesson #15)

If the SOW has been written to contract for proprietary code, there may be issues associated with the integration of proprietary code on system platforms. These propriety issues raise issues of integration, testing, and problem resolution. The team made no recommendations.

Configuration Management

Resource Constraints Encourage Multiple Configurations (Lesson #5)

The flexible nature of software configurable systems can create a pull toward multiple configurations.

Approach and Lessons Learned

The indirect costs of configuration proliferation must be considered. Increased operating, support, and training costs have all been observed.

Recommendation

The team recommended limiting configurations and grouping changes into block modifications.

Configuration Flexibility (Lesson #12)

Multiple hardware and software configurations can make the complex task of configuration management more difficult.

Approach and Lessons Learned

An effective approach to improve configuration flexibility has been the use of vehicle verification identification (VVID), which is the capability to automatically self-test and sense hardware and software version compatibility.

The practice involves loading a CD ROM to upgrade the system. Once the upgrade has occurred, the software will not permit loading a previous version of any of the LRUs, even if desired by the operator.

This practice has shown dramatic improvement in MTTR and is considered a safety critical feature.

Recommendation

Disseminate the practice of VVID more broadly.

Performance Specification (Lesson #14)

The relaxation of performance specifications (COTS driven) has produced an accompanying relaxation of change disclosure requirements. This has resulted in integration problems due to unanticipated incompatibility between systems on a platform.

Recommendation

The team recommended a tightening of performance specifications to promote disclosure of changes that could impact the performance of other systems.

Maintaining a Single Software Baseline (Lesson #20)

Programs often develop multiple paths of software development that require a delta effort to reestablish the single baseline. Program funding and schedules often fail to take into account the extra effort to reestablish the baseline.

For example, a new configuration/upgrade can be applied to some of the equipment, then patches or changes would be needed and introduced to the next set of equipment, and so on.

As a result, it would be hypothetically possible that no single piece of equipment would be exactly like any other.

Approach and Lessons Learned

Programs that maintain a single baseline software configuration report success in keeping configuration management under control. A “Plug & Play” configuration capability is supportable under this model if the baseline includes a superset of the required components.

Recommendation

Maintain a single baseline software configuration.

Documentation Timeliness (Lesson #21)

The documentation change process is cumbersome and slow. This results in documentation that lags far behind software releases.

In addition schedule pressures for fielding new releases provides a pressure to push documentation updates to the back burner. For example, quick fixes are often needed during and after testing and integration, and this pressure pushes documentation aside.

Recommendation

The team recommended a line item to budget and schedule documentation updates with each fielding of an upgrade.

Training

Operator/Maintainer Trainer (Lesson #26)

Operator and maintainer training resources (hardware and software enabled training devices) cannot accurately reflect finalized systems until the systems are delivered. Just-in-time delivery of these systems seems to preclude development time for the training resources. This has led to unsatisfactory workarounds and to training procedures that are inaccurate.

Recommendation

No definitive recommendation was reached. Several possibilities for addressing the problem were proposed, including concurrent system and system trainer development from the same contractor.

Trouble Shooting of Training (Lesson #27)

It is difficult to train for trouble shooting operations when the training involves learning a technique for solving a problem rather than a specific solution.

Some of the potential benefits that could come from more effective training of trouble shooting techniques include retention of expertise into a database of knowledge, the ability to utilize less skilled labor, and more efficient use of limited resources.

Approach and Lessons Learned

One program reported a trial use of “learning systems” that would be used to maintain and fix equipment, recognize trends and common problems, provide guidance to maintainers to check the most likely cause, and so on.

Recommendation

Determine if the “learning systems” approach merits further investigation. Research the commercial world’s extensive capabilities in the body of knowledge associated with call center support systems.

Aligning Training With Equipment Fielding (Lesson #28)

Training is often required to prepare trainers at or before operational equipment is fielded. As a result, there are delays in fielding operational equipment, as well as errors in training due to making incorrect guesses about anticipated operational equipment.

(This complements testing lesson #2). (It also has parallels to Lesson #26 which addressed the symptom of training not quite available with the fielded system.)

Approach and Lessons Learned

Field the trainers at the same time as or before fielding the equipment.

Recommendation

None. It's a paradox.

Subsystem Iteration Without Early Identification of Training Requirements (Lesson #29)

Subsystems are permitted to iterate without early identification of training requirements. This causes last minute reactive scrambles without a clear understanding of training requirements. In addition the problem points to subcontractors who are behind schedule.

Recommendation

The team recommended uniformly enforcing the good idea cut-off date (GICOD) and problem change request (PCR) freezes.

Operational Suitability

Problem Change Requests (PCRs) (Lesson #16)

The coordination and processing of PCRs is a critical function that requires substantial effort to get appropriate user input, make decisions and monitor and track progress.

Approach and Lessons Learned

User representation in the following areas has been beneficial in ensuring operational suitability:

- change control board (CCB)
- setting of priorities
- post field technical reviews (PFTR)
- field problem review boards
- post-supportability reviews
- program field reports
- block phased updates

Recommendation

The support and adherence to a formal problem change request (PCR) or system trouble report (STR) process is mission critical. User involvement is critical for ensuring operational suitability.

User Requests and Feedback (Lesson #17)

Organizations are motivated to maintain high scores on the readiness report. However, this motivation also leads to a reluctance to report deficiencies and a difficulty in justifying funding for enhancements (e.g., if an organization is 99.7% ready, why does a system need enhancements?). This issue leads to poor communication, particularly on items that are not safety-of-flight critical.

Approach and Lessons Learned

Informal mechanisms have been effective, such as field service representatives' conversations with the troops. These informal mechanisms have provided a safe way for field units to share their problems and needs.

Recommendation

A more systematic, anonymous mechanism would provide a way of building on the success of the informal mechanisms and of increasing opportunities to get feedback.

Maintenance Data Recording (Lesson #18)

Similar to lesson #17, operational units are motivated to produce positive reports relative to maintenance data recording. As a result maintenance data recording may be inaccurate. Inaccurate data can impact the ability to justify funding for upgrades.

Recommendation

The team recommended a mechanism for accurate data reporting without negative consequence for the unit.

User Recommended Technology Recommendations (Lesson #19)

As new technology is introduced, the mission changes to leverage the new technology. This produces a user-driven change request stream that must be managed.

Approach and Lessons Learned

Several programs have implemented a user prioritized “plum” list that is worked from the top down after mutual agreement upon scope change and program impacts. The programs attempt to accommodate user requests as much as possible, but also establish mechanisms for establishing priorities. A key to their success has been obtaining user involvement in setting priorities and sticking to them.

Recommendation

The team recommended that user prioritized request mechanisms be adopted more systematically.

Migration Planning

Obsolescence and Conversions While Meeting Operational Readiness Requirements (Lesson #22)

Regular upgrades are a part of reality in today’s world. The obsolescence and conversion of software while continually meeting operational readiness requirements requires user participation in migration planning.

Approach and Lessons Learned

Acceptance of regular upgrade schedules needs to be institutionalized.

Recommendation

The team recommended proactive upgrade planning with strong user participation. It is important that upgrade planning becomes part of the operational units culture.

LRU Reprogramming Standardization (Lesson #23)

The lack of a standardized LRU reprogramming process has produced cumbersome, slow, and error-prone upgrades.

Approach and Lessons Learned

Soldier portable on-system repair tool (SPORT), while in an early stage, represents one promising approach. SPORT uses a single tool to upgrade each LRU. The use of such an approach requires that all LRUs need to able to talk to this single tool.

Recommendation

The team recommended a standardized LRU file structure to facilitate a single reprogramming tool on all LRUs.

Software Configuration Upgrades (Lesson #24)

Software configuration upgrades can lead to errors, such as the loading of the wrong software versions.

Approach and Lessons Learned

A CD-based upgrade erases previous software versions and disallows reinstall of previous versions. This is coupled with on-board systems configuration checking that prevents the loading of incorrect software versions.

Recommendation

The team recommended that the CD approach be analyzed in more detail in conjunction with the recommendation from Lesson 12.

Dynamic Nature of Software Configurations (Lesson #25)

Regular and frequent upgrades are difficult to manage and require compatibility and coordination to be addressed. The problem is aggravated by multiple versions of the soldier portable on-system repair tool (SPORT vs. Super Sport).

Recommendation

The team recommended effort to reduce the load/verify time associated with software upgrades.

Cross-Communication

Deputy Chief of Staff for Operations (DCSOPS) Not Included in Integration and Fielding (Lesson #1)

There is a communications problem because higher levels of authority are not usually included on many IPTs. As a result issues are not identified early enough or elevated to the appropriate authority in a timely manner.

Approach and Lessons Learned

The IPT concept is a vital innovation that can be used as a starting point for including broader participation. In some examples, a cross-PM IPT with subcontractors and system integration laboratory participation has been an enabler for early identification of issues and for more effective communication.

Recommendation

The team focused on the need to expand the scope of the IPTs to include higher levels of authority. DCSOPS should be included to help coordinate integration, fielding, and test working group (TWIG) IPTs.

Expected positive outcomes include early identification of issues and improved visibility to appropriate higher levels.

Prioritization/Rank Grouping

The team went through an exercise to rank the lessons learned based upon two criteria: the desirability of learning from the lesson and the likelihood that learning will occur. The lessons were then plotted to group them into quadrants. Highly desirable lessons that are likely to be adopted are the "low-hanging fruit" that may warrant immediate action. Lessons that are undesirable and unlikely are low risk and require less attention. The other two quadrants (desirable and unlikely, undesirable and likely) are the challenges. It should be noted that these last two quadrants are roughly equivalent because most lessons could be reworded in the opposite sense (i.e., not having IPTs is bad or having IPTs is good).

A data reduction exercise was used to summarize the team's overall judgment of lesson importance. The exercise resulted in three leaders for each of the following groupings: Success (Lessons, 6, 9, & 12) Challenge (Lessons 2, 19, & 28), and Disappointment (Lessons 8, 14, & 26).

Figure 2 depicts the rank groupings.

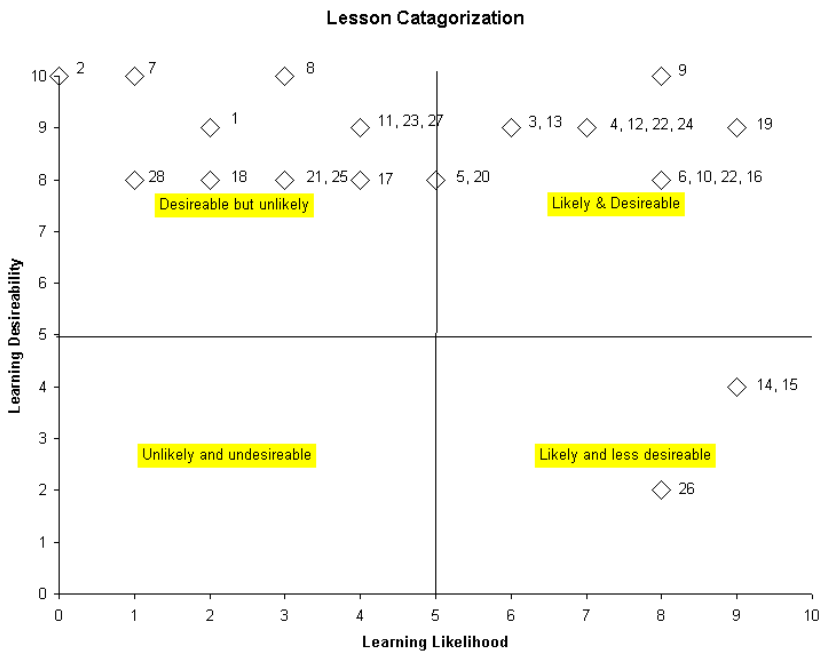

Figure 2: Rank Groupings of Lesson Importance

These results were ranked and grouped by likelihood and desirability as shown in Table 1.

Table 1: Likelihood and Desirability of Lessons Learned

	Lesson	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Likelihood	2	0	6	7	5	8	1	3	8	8	4	7	6	9	9
	Desirability	9	10	9	9	8	8	10	10	10	8	9	9	9	4	4
Votes for this as a ...	Success						5			4	1		5			
	Challenge		2					1				1				
	Non-succ.								4						5	1

	Lesson	16	17	18	19	20	21	22	23	24	25	26	27	28	29
	Likelihood	8	4	2	9	5	3	7	4	7	3	8	4	1	
	Desirability	8	8	8	9	8	8	9	9	9	8	2	9	8	not rated
Votes for this as a ...	Success														
	Challenge				4		1				1			5	
	Non-succ.											5			

Recommendations for Next Steps

Consensus was reached on many recommendations that should be easily actionable (the low-hanging fruit). The working group made the following overall recommendations:

- Additional work is required to quantify the benefits of the various recommendations to support their adoption or further development.
- Take on the challenges. There is much to be gained from the more difficult challenges.
- Implement higher level PM coordination, remove the stovepipes. So much is dependent upon close, timely communication. Develop contractual obligations to make the recommendations happen.

Appendix E Relationship of Workshop Findings to Enterprise Framework

The workshop findings can also be related to the enterprise framework for disciplined system evolution [Bergey, 97].

Reengineering project failures can be traced back to a small set of underlying problems. The enterprise framework offers an alternative to the lack of discipline that characterizes failures. It enables the following:

- evaluation of system evolution initiatives
- identification of global issues early in the planning cycle
- coordination of management and technical practices

The enterprise framework characterizes the global environment in which system evolution occurs and provides insight into the management and technical issues that must be addressed in evolving software-intensive systems. It helps managers to identify critical success factors and to develop a set of management and technical practices for planning, evaluating, and managing system evolution initiatives. The enterprise framework helps organizations to overcome the tendency to focus on a narrow set of technical issues without considering the broader systems engineering issues, the increased needs of the customer, the strategic goals and objectives of the organization, and the business operations of the enterprise.

As discussed in Section 3, the Enterprise Framework was one of the three perspectives that guided the organization of the workshop. The framework was initially developed for the development or maintenance manager with control over the system and its evolution. Because DoD is primarily an acquisition organization, the framework was supplemented with insights from the acquisition CMM and risk management. This appendix focuses on the role of the enterprise framework, considers how well it works from an acquisition perspective, and considers whether there are acquisitions that may be incorporated into the framework.

Overview of the Enterprise Framework

The enterprise framework has a systematic set of questions that help the program manager to identify the critical issues required for success in an evolution effort. At the highest level these questions include the following:

- How can we systematically sort out all the issues with which we are confronted?
- How do we plan the evolution of a large and complex system, including reengineering the system?
- What are the critical success factors of system evolution?
- How can we determine if we are on the right track?
- How do we evolve the system without adversely affecting operations?

The enterprise framework consists of seven elements that are building blocks for a successful system evolution effort. Each has a critical set of technical and management issues that are essential for developing a comprehensive plan of action.

The elements of the framework are

- organization
- project
- legacy system
- systems engineering
- software engineering
- technologies
- target system

Figure 3 is a high-level graphical representation of the enterprise framework. The arrows indicate how each of these elements uniquely contributes to a system evolution initiative.

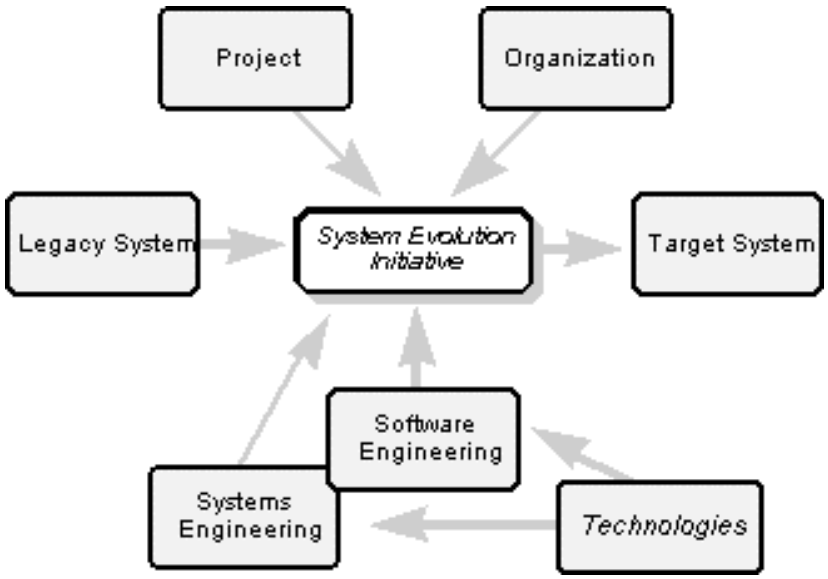

Figure 3: A Framework for the Disciplined Evolution of Legacy Systems

The framework breaks down each element into considerable detail. Appendix C has a set of checklists with approximately 200 questions associated with the framework to enable a manager to develop a comprehensive set of issues that need to be addressed and tracked.

For example, the following questions concern the legacy core system:

- Is there a current system configuration diagram? A system design document?
- Are the software architecture and software design well documented?
- Are the system interfaces and communication protocols documented?
- What are the dependencies on external interfaces?
- Is the functionality and operation of the system described adequately in user and system documentation?
- Have all the user interfaces been identified?
- Have the software applications and critical algorithms been identified? Have they been analyzed?
- Are the software interfaces and message and data formats documented?
- Have the performance characteristics of the system been assessed? Have benchmarks been run?
- Are the source code, library elements, and build scripts available? Are they current?
- Is there documentation on the logical and physical data dictionaries?
- Have dependencies on undocumented features been identified?
- Has the complexity and brittleness of the system been assessed?
- Has the integrity of the system been affected adversely by the maintenance legacy?

- How stable is the system's operation? Have the unresolved problem reports and change requests been reviewed for trend information?

Each of the other framework elements has a corresponding set of questions. Taken together, these questions provide a comprehensive checklist for a program manager who has control over the development and maintenance of a system.

While the framework focuses on developing a set of questions to guide a migration effort before it begins, the workshop captured a set of lessons learned after the completion of the system upgrades. The workshop offered an opportunity to test the use of the framework in practice and to determine the most critical practices found on an actual set of projects. It also suggested areas that are most important from an acquisition perspective, rather than a development perspective.

Not all issues identified by the framework would be expected to be equally relevant in a retrospective analysis of a completed project. In addition not all categories from the framework would be relevant for an acquisition organization.

Regrouping of Workshop Findings

To begin our analysis we group the high-level workshop findings in terms of the framework elements. (Because the workshop participants were from an acquisition environment, the software and systems elements are combined below.) This new grouping is listed below.

Organization

- Designate a decision-making agency to resolve inter-project disputes.
- When money is tight, don't sacrifice sound engineering practices.
- Determine whether software ownership will be needed before the contract award.
- Mandates often have unanticipated consequences. Be very cautious about issuing mandates, and involve all stakeholders in the decision-making process.

Project

- Specify requirements clearly, use tool support when appropriate, and practice change control.
- Use historical data and tools for collecting data.
- Provide training on the use of cost estimation and tools.
- Consider the impact of potential upgrades when making decisions about scheduling.
- Be sure to have experienced senior staff. Invest in worker skills and career paths.
- Involve all stakeholders throughout the project, particularly through the use of IPTs.

- Strive for mature processes for both the government and the contractor, especially in the areas of requirements management and control.
- Consider logistics, upgrades, and contingencies throughout the planning cycle.

Systems Engineering and Software Engineering

- Design the architecture with change in mind; plan for regular system upgrades.
- When using COTS for an extensive part of a system, allow ample time for integration and require the vendor to perform the integration.
- Address software problems right away. Don't defer them to follow-on contracts.
- If a contractor has attained CMM level 3, be sure to require the use of their metrics and data for managing the contract.

Legacy System

- Provide regular upgrade schedules.
- Avoid the confusion that results from configuration proliferation.
- Be sure that documentation is appropriate. It should contain the right amount of information to adequately maintain a project without containing irrelevant details.

Target System

- Plan for block upgrades.
- When possible, automate the installation of upgrades.
- Write training materials prior to the release and deploy the training before fielding the new system.

Technologies

- Be sure to maintain the integrity of testing even if cuts need to be made in a project.
- Have the appropriate development and test environment resources and tools in place.

Relevance of Framework

As might be expected, since the participants were from acquisition organizations, the workshop findings clustered in the organization and project elements.

Under organization, the framework focuses on such issues as goals, vision, scope of effort, decision-making, and the need for thorough technical analyses before committing to funding. The workshop findings are consistent with these concerns. Two workshop findings focus on specific areas that are highly relevant to DoD: software ownership and mandates. While these issues have hooks within the framework, future analyses of the framework from a DoD perspective may factor these issues more directly into the checklists.

Under project, the framework focuses on issues of plans, roles, responsibilities, management of work breakdown structures, and management of requirements. In addressing these issues, the workshop focused on ways to manage a contractor effectively in the light of acquisition reform. The lessons learned centered around historical data, communication, and mature processes on the part of both the contractor and the government.

The other framework elements relate to the system and its artifacts. The framework provides a detailed set of relevant issues to enable an understanding of the systems engineering decisions, the software engineering decisions, the legacy system, the target system, and the technologies. These issues are relevant for the contractor in planning and implementing upgrades. For example, detailed questions about the support environment, the operational environment, and the target and legacy core systems are often not explicitly addressed. In these areas the workshop participants, as may be expected, focused on planning, documentation, training, and testing.

The workshop provided real world evidence of the role of the framework in integrating a variety of lessons learned. The framework enables the findings to fit together to form a coherent picture of a recipe for project success. It provides a way of seeing the big picture and a means to abstract up from the approximately 250 individual lessons. The framework helps to make sense of the pieces of the puzzle. There is a need to more sharply distinguish between the differing issues and concerns of development and acquisition organizations and to provide insights for an acquisition specific framework focus. Both communities are highly relevant for the DoD since successful projects require teamwork between the DoD and the contractor.

Appendix F Questions from System Evolution Checklists

The questions from the system evolution checklists were used as a source for developing questions for the working groups. The complete set of system evolution checklist questions is listed below. The rationale behind the questions is detailed in Bergey [97, 98]. The questions are grouped according to six major categories: organization, project, legacy system, target system, system engineering, software engineering, and technology.

ORGANIZATION *CHECKLIST A*	*An initial set of questions that help probe the organization element includes the following:*
What are the enterprise goals?	
Has a common vision been developed and communicated?	
Have the key decision makers and stakeholders been identified?	
Are the goals of the organization aligned with the enterprise goals?	
Are there defined criteria for the successful accomplishment of goals? *Are these criteria measurable?*	
What is the corporate information technology strategy?	
What is the overall scope of the systems evolution effort?	
Is there an established procedure for performing business/mission needs analysis to determine how new customer needs can best be met?	
Are the roles and responsibilities of each of the organizational units involved in the systems evolution effort well defined?	
How will efforts be coordinated across organizational units and with external customers?	
Does the organization provide suitable infrastructure support to assist projects in contracting, quality assurance, and other key activities that may be beyond the scope of an individual project to perform?	
What is the review and approval process for new and revised work products?	
Is there a well-defined issue resolution process?	

ORGANIZATION *CHECKLIST B*	*There are things which organizations commonly tend to do, but <u>should avoid doing</u>. A representative checklist for intercepting bad practices includes the following questions:*
Have the benefits of evolving the legacy system been predetermined without first conducting a thorough analysis?	
Has the feasibility of evolving the system also been predetermined?	
Have all three project variables (capability, schedule, and cost) been determined by the organization prior to having the project develop a formal plan for evolving the system?	
Have some aspects of the solution space been predetermined before analyzing the system and involving the project team?	
Has sufficient time been allowed for a thorough systems engineering analysis before finalizing the project implementation plan?	
Is a complete project implementation plan required before developing a concept of operations for the target system and obtaining the agreement of customers and the user community?	
Is a new and unproved life-cycle process being mandated without soliciting project feedback and the agreement of project team leaders?	
Is training and other infrastructure support being provided for piloting the application of new processes, tools, and work products before attempting to institutionalize them?	

PROJECT CHECKLIST *— PLANNING RELATED —*	*A checklist for probing the project's planning-related practices includes the following questions:*
Is there a clear understanding of the organization's goals and a linkage between the organization's strategy and the project's strategy?	
Is there a comprehensive project plan? *Are all the deliverables specified?*	
Are roles and responsibilities defined clearly?	
Does the project plan define the migration strategy clearly? *Are the systems and software engineering teams fully supportive of the migration strategy?*	
How realistic is the project plan and work breakdown structure (WBS)?	
Does the WBS describe all the tasks for implementing the migration strategy?	

Does the plan include estimates of the resources and time required for each task?
Are there subsidiary plans covering risk management, configuration management, quality assurance, and software development? *Is ownership of each plan and project work product established clearly?* *Have the plans been suitably coordinated?*
What are the cost and schedule for completing the effort?
Is a network activity diagram included which identifies the intertask dependencies?
How will the project obtain and integrate the necessary interdisciplinary skills?
What kinds of infrastructure support do the systems and software engineering activities require from the project? *Are they included in the project plan?*
Has training been arranged for the system developers and software engineers?
Are all phases of the project's life cycle addressed adequately in the project plan?
How will progress be measured and reported?
Is there a process in place to ensure that the project plan is updated as changes occur?
Is there a chief systems engineer, or group, who is accountable for the systems engineering and software engineering effort?
Will a project team composed of key task leaders and interdisciplinary engineers be established to serve as a system design team? *If not, how will global systems engineering issues and specialty engineering requirements (e.g., security) be addressed and coordinated adequately?*
Do plans include training for customers and users of the system?

PROJECT CHECKLIST *— RISK RELATED —*	*A checklist for probing the project's risk-related practices includes the following questions:*
How will risks be managed and mitigated?	
Are a process and criteria in place for make/buy decisions?	
Has an effective contracting strategy been developed?	
Is the project adequately funded?	
Is there evidence of overly optimistic schedule compression?	

PROJECT CHECKLIST *— REQUIREMENTS RELATED —*	*A checklist for probing the project's requirements-related practices includes the following questions:*
Has a common concept of operations for the proposed system been developed and communicated?	
Does the project have a requirements change management process?	
How are the customer and user requirements prioritized?	

LEGACY CORE SYSTEM *CHECKLIST*	*The following questions form an initial checklist for probing the technical features and current state of the legacy system:*
Are the software architecture and software design well documented?	
Is there a current system configuration diagram? *A system design document?*	
Are the software architecture and software design well documented?	
Are the system interfaces and communication protocols documented?	
What are the dependencies on external interfaces?	
Is the functionality and operation of the system described adequately in user and system documentation?	
Have all the user interfaces been identified?	
Have the software applications and critical algorithms been identified? *Have they been analyzed?*	
Are the software interfaces and message and data formats documented?	
Have the performance characteristics of the system been assessed? *Have benchmarks been run?*	
Are the source code, library elements, and build scripts available? *Are they current?*	

Is there documentation on the logical and physical data dictionaries?
Have dependencies on undocumented features been identified?
Has the complexity and brittleness of the system been assessed?
Has the integrity of the system been affected adversely by the maintenance legacy?
How stable is the systems operation? *Have the unresolved problem reports and change requests been reviewed for trend information?*

OPERATIONAL ENVIRONMENT CHECKLIST	*The following questions form an initial checklist for defining the baseline for the legacy system's operational environment:*
Are all of the customers, customer sites, and user groups identified?	
Are all of the legacy system products and services on which the users depend identified?	
Is there a profile to accurately characterize the current system workload?	
Are all of the external artifacts, system files, and procedures on which the users depend identified?	
Are there operational usage scenarios to ensure that there is a common understanding of the system's capabilities and operation from a user's viewpoint?	
Is there an accurate, up-to-date network configuration diagram that specifies the subsystems and their interfaces?	
Are all of the external system interfaces identifiable and documented?	
Are the hardware and software interoperability dependencies with external sites identified and documented?	
Are the software communication protocols identified? *Are they documented?*	
Are the system's security provisions and features clearly understood by the project team?	
Are the logistic, support, and system administration operations (and roles and responsibilities) itemized? *Are they traceable to specific subsystems (and agents)?*	
Will the operation of the legacy system be sustained to allow adequate time for users to obtain training and fully make the transition to the proposed system?	

SUPPORT ENVIRONMENT CHECKLIST	*These checklist issues concern potential shortcomings of the support environments that could derail a systems evolution effort:*
What is the composition of the support environments? *What products and services do they provide?*	
To what extent is the development and maintenance environment consistent with the developer's original environment? *Does it include the tools used for requirements elicitation and validation, design, and testing?*	
To what extent are the tools in the support environments integrated? *Are there established procedures for their use?*	
Do the tools enforce or promote good programming practices? *Are there documented programming guidelines and practices?*	
Is a separate integration and test environment available to the maintainers apart from the operational system? *Does this environment accurately reflect the operation of the legacy system?*	
Are project management functions such as planning, estimating, costing, scheduling, progress reporting, and issue and problem resolution supported?	
Which functions are supported by automated tools? *Are there labor-intensive functions being manually performed that can be improved by adopting new tools or processes?*	
How is configuration management being performed on the hardware and software products undergoing development, reengineering, or maintenance? *Are the efforts coordinated?*	
Are software build processes well documented? *Do they produce repeatable results?*	
Does the integration and test environment provide an automated regression testing capability?	
How are new releases placed into operation?	
To what extent are proprietary or customized tools being used?	
Are the COTS tools up to date and still supported by the tool vendor?	
Is there a defined process for determining when COTS tools should be upgraded to the latest product release from the vendor?	
Are the support environments under configuration management and control?	

SYSTEMS AND SOFTWARE ENGINEERING CHECKLIST	*A useful checklist for carrying out the systems and software engineering activities (in conjunction with the target system element checklist) includes the following questions:*
Are mechanisms in place to ensure that software engineering tradeoffs and considerations are an integral part of the up-front systems engineering activities?	
Has an incremental development strategy been adopted?	
Has consideration been given to adopting an incremental implementation approach that is driven by the highest priority risks that have been identified to date?	
To what extent will prototyping be employed? *Have criteria been established?*	
Is there a defined process for performing system engineering tradeoff analyses and allocating system requirements to hardware and software?	
Is there a formal process for risk assessment and mitigation? *Is it performed regularly or is it a one-time activity?*	
Are appropriate systems and software engineering tools being used?	
What means are being employed to ensure requirements traceability?	
Is the systems engineering team responsible for the technical oversight of individual hardware and software product developments? *How will this oversight be accomplished?*	
How will the degree to which the legacy system software is salvageable and evolvable (from a technical and economical standpoint) be determined?	
Is there evidence to support that the prescribed systems and software engineering methodologies are effective?	
Is a process in place for evaluating candidate software architectures and assessing their quality attributes?	
What approach is planned to acquire an understanding of the design, functionality, usability, reliability, performance, and operation of the legacy system?	
What is the process for deciding to make changes to programming languages, operating systems, and related technologies?	
Are the training needs of the systems engineers and software engineers identified?	
Are programming guidelines established? *Are they followed?*	

Is there a change-management strategy for accommodating ongoing software changes to the legacy system that occur during the development of the target system?
Are the transition issues associated with operationally deploying the system being addressed?
How will changes to software interfaces with external systems be coordinated?
Is there a strategy in place for achieving upward software compatibility?
Are programs needed for converting existing data files and databases? *Will they automatically make the conversion or will user intervention be required?*
Are the training needs of the systems engineers and software engineers identified?

TECHNOLOGY CHECKLIST A	*A checklist for screening candidate technologies for potential project application might include the following questions:*
Does the technology have the potential to make a significant contribution to the enterprise goals and objectives? *Can it provide a competitive advantage?*	
Is the technology a prerequisite for the systems evolution effort?	
Is the technology sufficiently mature and stable?	
What tangible benefits can the technology provide? *Is it required for system compatibility?* *Is it a prerequisite for adopting other technologies?*	
Have pilot efforts or case studies confirmed the suitability of the technology for the specific application domain?	
Have the benefits of adopting the candidate technology been quantified?	
What is the potential impact of not adopting the technology?	

TECHNOLOGY CHECKLIST B	*Once a candidate technology has been determined to be generally suitable, a checklist covering the technology selection process should include answers to the following questions:*
Are the cost, schedule, and impact of applying the new technology acceptable?	
Is adequate training available? *Are key members of the project team already well versed in the technology?* *Can they act as mentors to other team members?*	
Have the pros and cons of alternative technologies been weighed carefully (preferably using a formal risk assessment process)?	
Has the impact of the new technology on existing customers and users been analyzed? *Do the customers and users have any strenuous objections?* *Unheeded cautions?*	
Is management aware of the technology adoption plans? *Are these plans consistent with the organization's strategic plan?* *Are there any reservations or cautions?*	
Is a suitable measurement program being adopted to quantify and evaluate the actual benefit of applying the technology?	
Is there a contingency plan in the event that any unforeseen technology "show-stoppers" arise?	

TARGET CORE SYSTEM CHECKLIST	*A checklist of issues to consider in making decisions about the desired features of the proposed target system include:*
Is there a prescribed means for eliciting and validating the target system requirements? *Has it been used before?* *Is there evidence of its effectiveness?*	
Is there a concept of operations to describe the proposed target system?	
Have operational scenarios been developed to describe how the proposed system will operate?	
Have the concept of operations and operational scenarios been validated with customers, users, and key systems personnel?	
Is the difference between the current "virtual requirements" of the legacy system and the new target system requirements well understood?	
Are there standards with which the target system must comply?	
What ground rules have been established for the use of COTS software?	
How robust is the current legacy system architecture? *Is it practical to evolve this architecture to meet the target system requirements?*	
Should the system be re-hosted on a new platform or operating system? *Is the use of a new programming language justified?*	
What process is used to determine the target system architecture requirements?	
What are the desired performance, availability, and security attributes?	
Can the target system be evolved incrementally over a period of time or is a major reengineering effort required to bring about the desired changes?	

OPERATIONAL ENVIRONMENT CHECKLIST	*An initial checklist of issues to consider for the operational environment that is envisioned for the target system includes the following questions:*
What changes are required in the operational environment to accommodate the new target system requirements?	
What is the projected impact of the proposed changes on current business operations? *How will these affect the customer and the organization?*	
Do the customer and user requirements include explicit changes to the operational environment? *How do these changes affect the target system (hardware and software)?*	
What is the projected impact of the proposed changes on performance and availability?	
What differences are there between the existing legacy system environment and the proposed target environment? *Are there incompatibilities that will need to be resolved?*	
Will support for some of the existing products and services be dropped? *What customers and users will be affected?*	
Which external interfaces need to be modified? *How will these modifications be coordinated with external systems and users?*	
What testing is needed to assure interoperability?	
What is the plan for "roll out" and "cut-over" to the new system?	
What parts of the target and legacy systems need to coexist during operational transition?	
In the event of a crisis, to what degree can support be rolled back to the legacy operational environment?	
Will the new target environment impose new operating procedures?	
Will operators or system administrators require training on the new operating environment?	
Have training needs been identified for customers and users of the system?	

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	November 2001	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Army Workshop on Lessons Learned from Software Upgrade Programs	F19628-00-C-0003

6. AUTHOR(S)

William Anderson, John Bergey, Matthew Fisher, Caroline Graettinger, Wilfred J. Hansen, Ray Obenza, Dennis Smith, Halbert Stevens

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2001-SR-021

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	

11. SUPPLEMENTARY NOTES

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

13. ABSTRACT (MAXIMUM 200 WORDS)

The Software Engineering Institute (SEI) conducted the "Software Upgrade Workshop for Legacy Systems" at the Redstone Arsenal June 5–7, 2001. The workshop captured experiences from software upgrade efforts for the Abrams, Bradley, Patriot, Apache Longbow, and Multiple Launch Rocket programs. Workshop participants explored strategies that worked and those that failed, and the obstacles that were encountered. They addressed pre-award planning, project management, software and systems, mandates, and deployment. Their effort resulted in a set of recommendations and guidelines to help organizations improve the process of upgrading legacy systems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software upgrades, project management, legacy systems, system upgrades	102

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102