The Impact of Software Support on System Total Ownership Cost

Naegle, Brad R.

Monterey, California. Naval Postgraduate School

NPS-FM-04-024

EXCERPT FROM THE

PROCEEDINGS

OF THE

FIRST ANNUAL ACQUISITION RESEARCH SYMPOSIUM

THE IMPACT OF SOFTWARE SUPPORT ON SYSTEM TOTAL OWNERSHIP COST

Published: 30 September 2004

by

Brad R. Naegle

2nd Annual Acquisition Research Symposium of the Naval Postgraduate School:

Charting a Course for Change: Acquisition Theory and Practice for a Transforming Defense

May 13, 2004

Prepared for: Naval Postgraduate School, Monterey, California 93943

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

The research presented at the symposium was supported by the Acquisition Chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request Defense Acquisition Research or to become a research sponsor, please contact:

NPS Acquisition Research Program
Attn: James B. Greene, RADM, USN, (Ret)
Acquisition Chair
Graduate School of Business and Public Policy
Naval Postgraduate School
555 Dyer Road, Room 332
Monterey, CA 93943-5103
Tel: (831) 656-2092
Fax: (831) 656-2253
E-mail: jbgreene@nps.edu

Copies of the Acquisition Sponsored Research Reports may be printed from our website www.acquisitionresearch.org

Conference Website:
www.researchsymposium.org

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Proceedings of the Annual Acquisition Research Program

The following article is taken as an excerpt from the proceedings of the annual Acquisition Research Program. This annual event showcases the research projects funded through the Acquisition Research Program at the Graduate School of Business and Public Policy at the Naval Postgraduate School. Featuring keynote speakers, plenary panels, multiple panel sessions, a student research poster show and social events, the Annual Acquisition Research Symposium offers a candid environment where high-ranking Department of Defense (DoD) officials, industry officials, accomplished faculty and military students are encouraged to collaborate on finding applicable solutions to the challenges facing acquisition policies and processes within the DoD today. By jointly and publicly questioning the norms of industry and academia, the resulting research benefits from myriad perspectives and collaborations which can identify better solutions and practices in acquisition, contract, financial, logistics and program management.

For further information regarding the Acquisition Research Program, electronic copies of additional research, or to learn more about becoming a sponsor, please visit our program website at:

www.acquistionresearch.org

For further information on or to register for the next Acquisition Research Symposium during the third week of May, please visit our conference website at:

www.researchsymposium.org

THIS PAGE INTENTIONALLY LEFT BLANK

The Impact of Software Support on System Total Ownership Cost

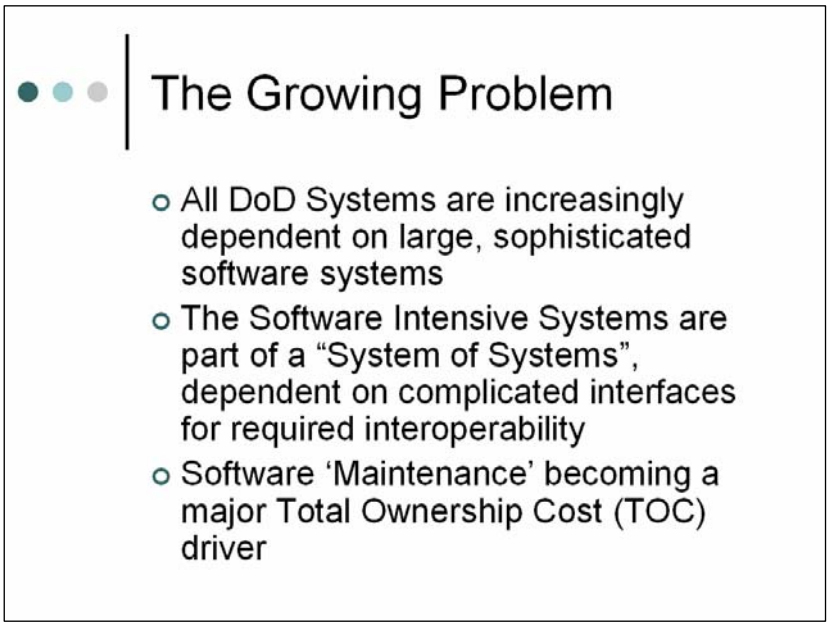
Presenter: Brad R. Naegle, Lecturer, Graduate School of Business & Public Policy, Naval Postgraduate School

Abstract

As a spin-off of the Total Ownership Cost (TOC) research that Mike Boudreau and I conducted, there was some interest in examining the TOC implications of software intensive systems and what the software component is adding to the TOC burden. I thought it would be interesting to get into this, it felt a lot like opening 'Pandora's Box.'

Introduction

The Growing Problem

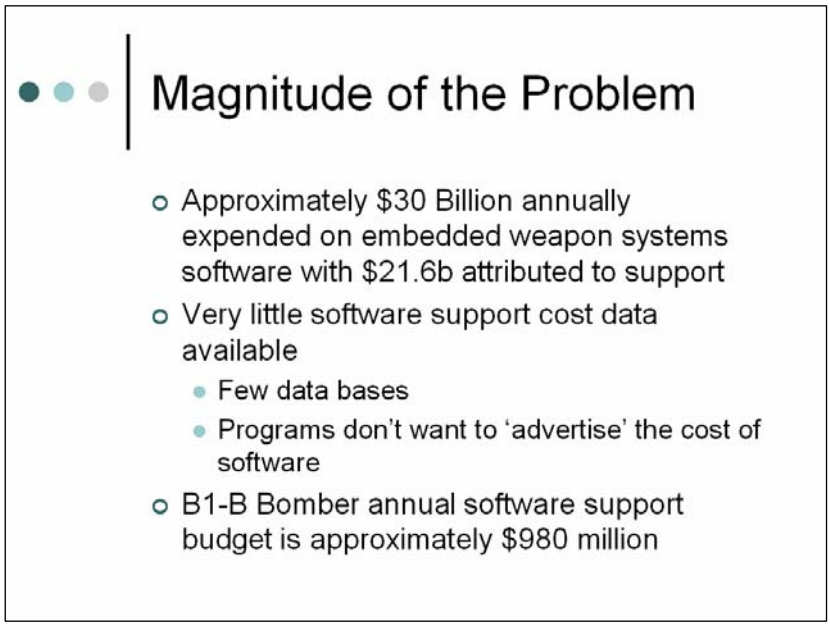


We are obviously significantly dependent on these software systems. Virtually everything we have is moving into a software intensive system. We've gone from the M-16 rifle to our new objective individual combat weapon, which has lines and lines of software code. We want to put these together in the system of systems that Dr. Gansler talked about in the keynote presentation at the Symposium.

These systems of systems are going to be an important concept as we talk about TOC and the software drivers linked into the difficult interfaces that are associated with making a

system of systems work effectively. Software maintenance is becoming an ever-increasing part of the TOC of our systems.

Magnitude of the Problem



How big is the problem? With the lack of databases that we discovered in the first research effort, we do not have a really good accounting of how much money is being spent on the software component of software intensive systems.

Some estimates indicate we spend about $30 Billion a year on embedded weapons system software. This is not the management information systems piece; this is literally the tactical systems portion. Of that, about $21.6 Billion is attributed to software maintenance and it's continuing to grow. Given what Dr. Gansler said, we spend about $80 Billion a year overall, a quarter of it being software maintenance at this time and growing.

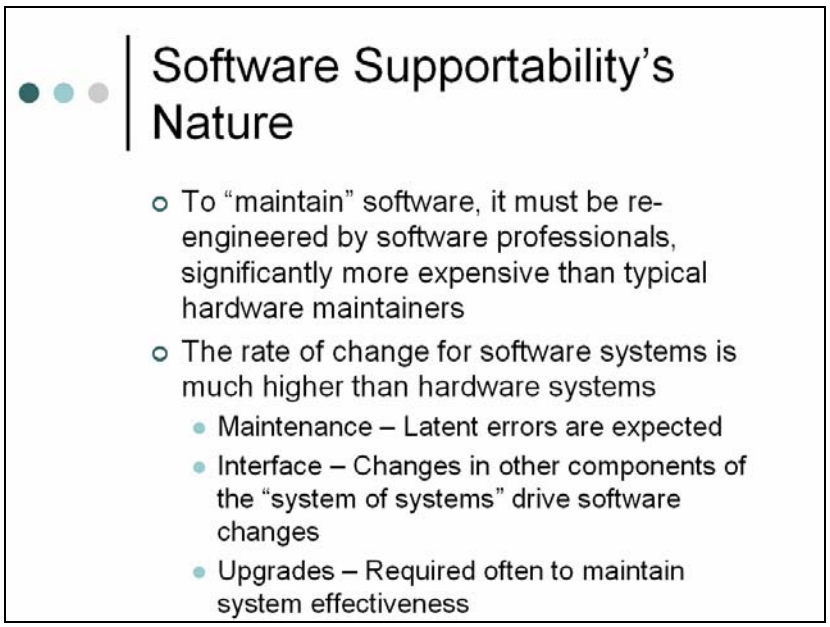
The cost data is hard to come by, with few data sources. I asked a number of program managers what it costs to support software. They are less than forthcoming with numbers, which might be attributed back to the program, as it is typically a large number.

One of the pathologies I encounter is that we don't want to talk about the TOC of systems. The rational is that decision makers, Congressmen and others who can kill a program, are not seeing numbers presented in a way to illuminate TOC. No one wants to be the first to say that an M-1 tank doesn't cost $2 Million a copy; it actually costs $12 Million a copy if you look at it from the TOC perspective. Someone unfamiliar with the concept of evaluation would look at those numbers and eventually cancel the program.

I was able to locate information on the B1-B Bomber program; this is the old Reagan era Bomber. I happened to work with the software maintenance manager of that system who said

her budget was $980 million a year to support the software only on the B1-B Bomber. That gives some perspective on the number of dollars being put into software maintainability.

Software Supportability's Nature



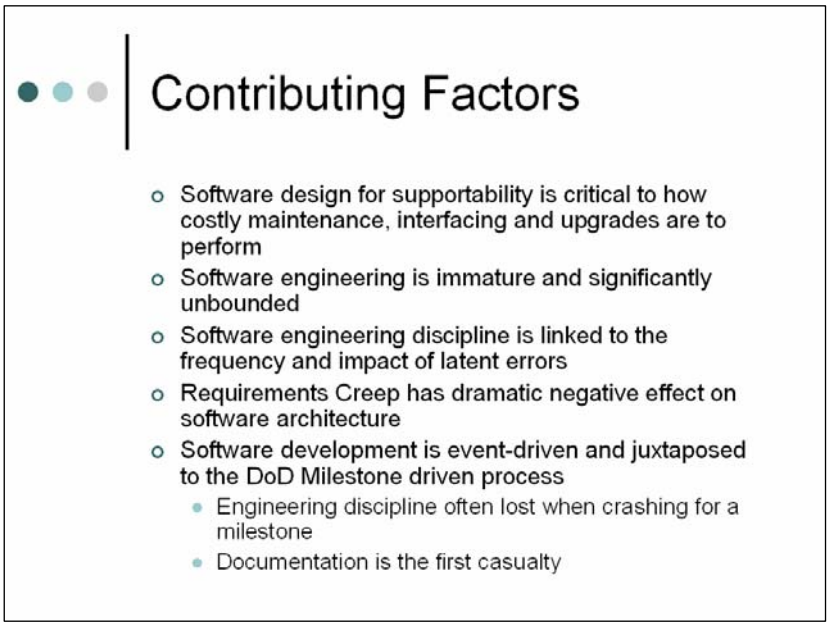
What is software maintenance? We often talk about it as if it's a supportability thing like hardware maintenance. Software maintenance is really software reengineering. Those responsible for software maintenance are software engineers or software professionals. To hire that group of people, the cost is much higher than for a typical hardware maintainer. Automatically, the cost-basis for hiring people to support our software are higher. It is also important to note, software systems are changed at a much higher rate than hardware systems.

As a point of reference, software is actually deployed with the knowledge that there are thousands of latent errors throughout and those errors will be identified in use. For example, when Microsoft released Windows XP, the very day of the release, 2.8 GB of patches needed to go on it. You have to expect the errors in these things. In fact, if Microsoft met their own goal for errors per 1000 lines of code; XP would have 8 million errors. That's what is expected in a software build, due to the complexity of it. Software is a different animal than what we have grown accustomed to in hardware deployment.

Interfaces between software systems and hardware within these systems of systems are critical to make the systems of systems run efficiently. When one change is made to one system within the system of systems, it requires interface changes to ripple across the rest of the systems that are involved. Sometimes the interfaces are seamless and go well and no interoperability problem occurs. More often than not, a single change in software function requires changes throughout the system of systems. This is a driving factor that continues to increase the maintainability rates for the software.

Along the same line, software must be upgraded continuously to maintain required levels of performance within the system. For example, the M-1 Abrams office tells me their goal is to reduce software drops or additions to the system to twice a year. Hardware systems do not change that frequently, it becomes much more difficult to maintain the integrity of our software systems.

Contributing Factors



There are some contributing factors to how the software is physically architected which have a huge impact on costs related to resolving issues, scalability, maintaining or other required alterations. Among these are:

Software engineering. With over 50 years of history, Software Engineering is still immature. We do not have a standardized language to build software. We still lack the skills and the skill sets that are required to build upon a standard body of knowledge like more mature engineering disciplines have overtime. Unfortunately, when a new software system is built specifically for the DOD, it can rarely be reused. The system is built from scratch. It's like implementing and maintaining a new technology every time we build a new software system.

Software is significantly unbounded. Software doesn't have the physical world as a concern. It is literally the logic processes that are involved with the coders and the people who are involved with the design of the software.

Engineering discipline is often linked to the frequency and impact of latent errors – the importance will be made clear later in this presentation.

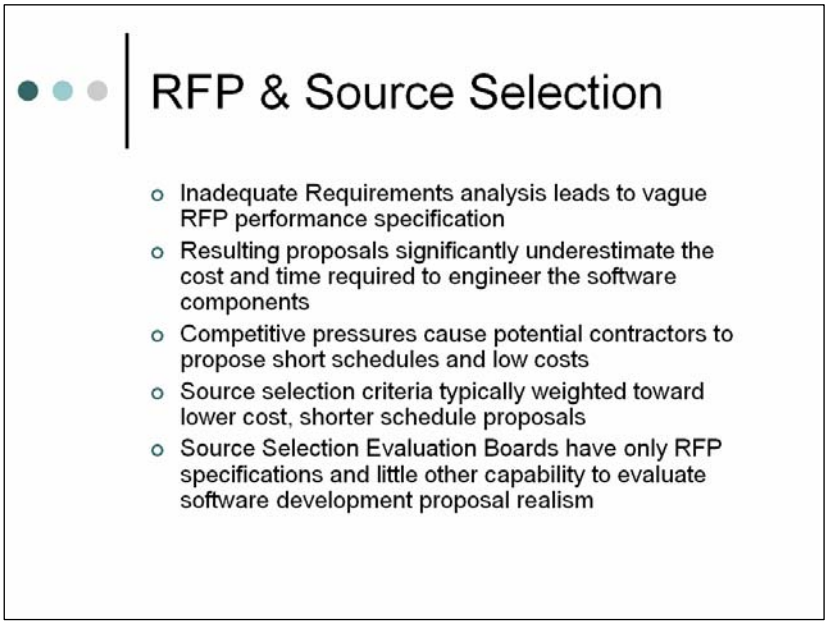
Requirements Creep has dramatic negative effect on software architecture. The negative effect is more dramatic than it is in hardware due to the complexities and the

interoperability pieces that go with the software. As we saw in John Dillard's presentation, we set up acquisition processes against the milestones. Those milestones are fixed in concrete because of the funding system that goes along with them.

It is well documented that software development is an event-driven process. Trying to put an event-driven process function within a milestone model creates significant issues, especially when imposed milestones are driven by oversight rather than clear software evaluation points. The first thing that typically happens is the engineering discipline is lost. The focus becomes milestone driven, rather than quality, losing engineering discipline and the ability to maintain the system.

The first casualty is documentation, which is critical for the supportability of the software. Processes are shortened, then "undisciplined coding to get functionality and move on," becomes the continual loop.

RFP & Source Selection

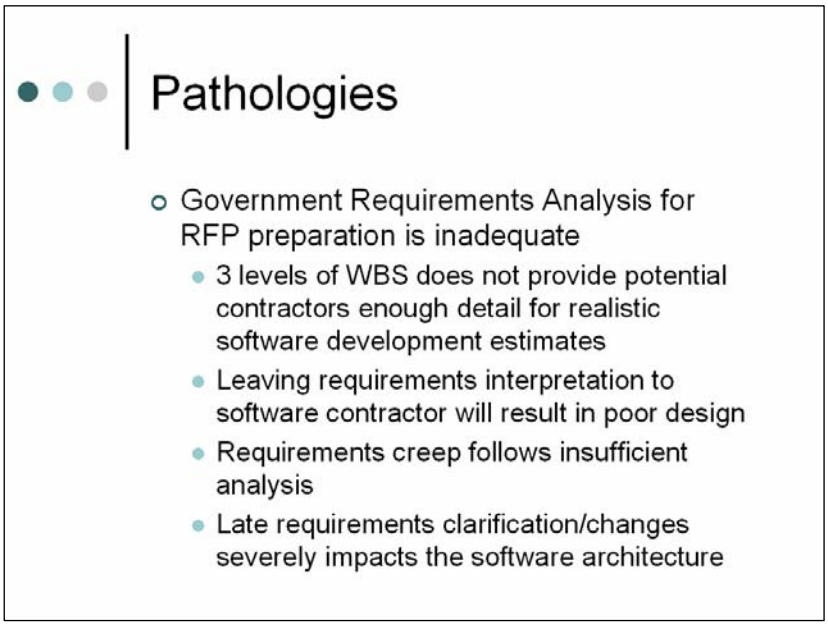


How do we go about doing the request for proposal and the source selection on our software intensive systems? The process is not significantly different than for hardware centric systems. With recent reforms toward performance-based specification, a lot of detail is left out. This is purposeful to garner innovation. Requirements analysis is weakened as the contractor is required to make sense of open-ended requirements and maintain cohesion within the system of systems.

Without clear requirement expectations, realistic estimates of time, effort, dollars and delivery schedule are nearly impossible. It also becomes much more difficult to compare contractors based on quantifiable selection factors like price and schedule. While the intent is quality innovation at a good price, the results are foggy requirements with unrealistic deliverables and schedule. Quality software innovation takes back seat to the selection process where evaluation boards only have the RFP type data to evaluate the software development

realism. The net effect; we still do not have an objective way of determining whether or not what is proposed and ultimately awarded will actually be anywhere near the reality of developing that software component of the system.

Pathologies

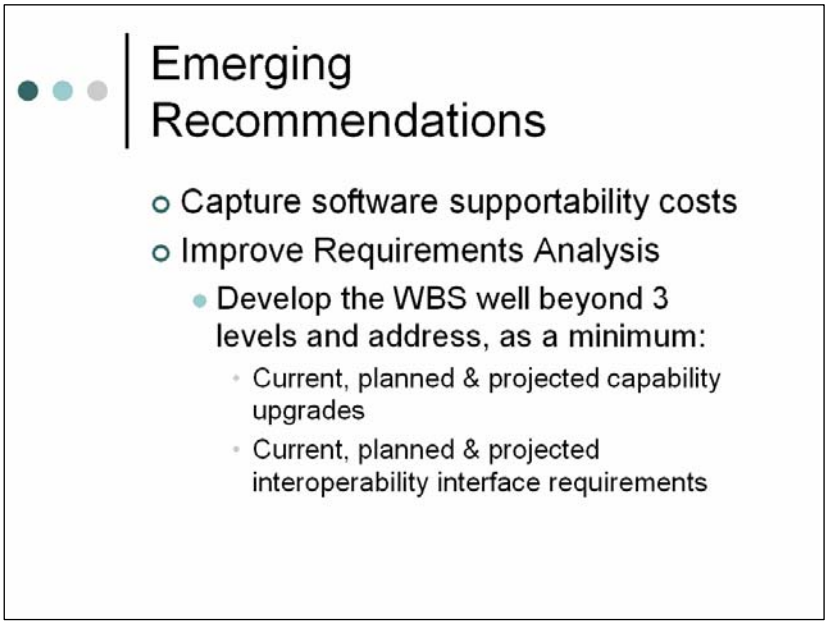


Here are some of the pathologies that go along with the software development piece. First, requirements are not broken into the level of detail required. Currently in the RFP process, level three is required of the work breakdown structure. This is one level below the major end item in the software architecture. This is not enough detail for the contractor to build as they would in a mature engineering environment such as with hardware.

Software requires a much more detailed approach to system requirements. If one leaves software system architecture to the interpretation of the software developer without clear requirements, poor design becomes standard. As noted previously, this introduces critical functional errors to the software system of systems as new software is built with top-line functionality only.

It is more costly to fix errors the later they are discovered in the software production cycle. Strong requirements, refined over time, develop stronger processes. Requirements creep is part of managing the software lifecycle; without a clear structure in place, late requirements clarification/changes will severely impact the software architecture and lengthen the time and costs associated with error corrections.

Emerging Recommendations



Somehow, we have to get our hands around how the support costs of weapons systems are contributing to TOC, especially the software component. It is important that we capture where the money is being spent and attack issues as they relate to sustainability.

It is important that we improve the requirements analysis. Expecting to hand off a level-three work breakdown structure to a software intensive system and hoping to get a quality product is not realistic. At the very minimum, we need to tell contractors what is the current, planned and projected capability upgrades. Even though software is ever changing, it is important that we make a cut at requirements and upgrade expectations to enable contractors to build efficiently in the front end and construct the software architecture for flexibility to accommodate those changes and upgrades. This should also be applied for software interfaces.

We require higher safety and security requirements on intensive software systems, beyond what is readily available in most of the commercial markets.

Exception or fault handling: There are current software systems in the tactical world that lock up when a fault occurs. In a combat situation, this is deadly. A system needs to have a 'reject faults' capability, to move on and continue to function.

Recovery technique: For example, I spoke to a Navy commander who was involved with the STENNIS. A software glitch in the system caused the ship not to know where it was in the world. They didn't want to get too close to land masses or any other ships so they steamed around for about six hours rebooting the software.

Reliability: Our requirements for reliability in our weapon systems are thousands of times higher than what we expect from the software sitting on our desks and in our offices.

Redundant Capability: What do we need to make sure it does not go down under any circumstances?

Conclusion

The software component of our increasingly high-technology weapons systems provides the capabilities and lethality desired for our forces, but is potentially devastating to our ability to cost-effectively maintain their advantages.

The complexity of individual software-intensive systems is significantly compounded when they are combined in a "system of systems" architecture. The initial software architecture, driven by how requirements are translated into performance specifications, is critical in determining how much maintenance will be required and how much effort will be required in the necessary maintenance actions.

To gain more effective software design, significantly more effort is required in requirements analyses. Performance specifications must be much more developed than is typical in the current development model. Handing off performance specifications developed through just three levels of the Work Breakdown Structure (WBS) for software intensive systems is insufficient in a complex, system of systems environment dependent on seamless interfaces in an ever-changing architecture.

Significant development, incorporating all critical performance features, interface requirements, and known, planned and projected upgrades, changes and enhancements must be effectively transmitted to the developer for consideration in the software design and architecture.

Without these efforts, software supportability costs will continue to skyrocket as existing software will require expensive and time consuming re-engineering to accommodate interface and capability changes that were known or could have been derived from more thorough requirements analyses.

THIS PAGE INTENTIONALLY LEFT BLANK

2003 - 2006 Sponsored Acquisition Research Topics

Acquisition Management

- Software Requirements for OA
- Managing Services Supply Chain
- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Portfolio Optimization via KVA + RO
- MOSA Contracting Implications
- Strategy for Defense Acquisition Research
- Spiral Development
- BCA: Contractor vs. Organic Growth

Contract Management

- USAF IT Commodity Council
- Contractors in 21st Century Combat Zone
- Joint Contingency Contracting
- Navy Contract Writing Guide
- Commodity Sourcing Strategies
- Past Performance in Source Selection
- USMC Contingency Contracting
- Transforming DoD Contract Closeout
- Model for Optimizing Contingency Contracting Planning and Execution

Financial Management

- PPPs and Government Financing
- Energy Saving Contracts/DoD Mobile Assets
- Capital Budgeting for DoD
- Financing DoD Budget via PPPs
- ROI of Information Warfare Systems
- Acquisitions via leasing: MPS case
- Special Termination Liability in MDAPs

Logistics Management

- R-TOC Aegis Microwave Power Tubes

- Privatization-NOSL/NAWCI
- Army LOG MOD
- PBL (4)
- Contractors Supporting Military Operations
- RFID (4)
- Strategic Sourcing
- ASDS Product Support Analysis
- Analysis of LAV Depot Maintenance
- Diffusion/Variability on Vendor Performance Evaluation
- Optimizing CIWS Life Cycle Support (LCS)

Program Management

- Building Collaborative Capacity
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to Aegis and SSDS
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Terminating Your Own Program
- Collaborative IT Tools Leveraging Competence

A complete listing and electronic copies of published research within the Acquisition Research Program are available on our website: www.acquisitionresearch.org

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL
555 DYER ROAD, INGERSOLL HALL
MONTEREY, CALIFORNIA 93943

www.acquisitionresearch.org