Real-Time Systems Engineering: Lessons Learned from Independent Technical Assessments

Theodore F. Marz
Daniel Plakosh

June 2001

Dependable Systems Upgrade Program

Unlimited distribution subject to the copyright

TECHNICAL NOTE
CMU/SEI-2001-TN-004

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

Copyright 2001 by Carnegie Mellon University.

NO WARRANTY

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

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

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

External use. Requests for permission to reproduce this document or prepare derivative works of this document for external and commercial use should be addressed to the SEI Licensing Agent.

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 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Abstract

The Software Engineering Institute (SEI) has performed several Independent Technical Assessments (ITAs) on mission-critical/real-time systems for the Department of Defense and other agencies.

This paper contains observations, recurring themes, trends, and lessons learned about systems development as derived from real-time/mission-critical programs that have been reviewed over the last three years.

It is hoped that the observations contained in this paper will be of value to future program managers and help ensure their success.

1 Background

The Software Engineering Institute (SEI) has performed several Independent Technical Assessments (ITAs) over the last three years on Department of Defense and other government agency mission-critical and real-time software-intensive systems.

An ITA is an objective, technical evaluation of a specific software intensive system development or acquisition program that is conducted

- by an SEI team staffed with an appropriate mix of expertise[1]
- through a series of planned interviews with the program stakeholders
- with the goal of providing actionable recommendations to leverage the program's strengths and minimize/mitigate the risks

An ITA is similar to the activities performed by other federally funded research and development centers (FFRDCs) and the Office of the Undersecretary of Defense for Acquisition and Technology (OUSD(A&T)).[2] The ITA differs from these assessments in both the composition of the teams, the method of the assessment, and the method of reporting.

ITAs are typically initiated by the System Program Director, program executive officer, or another, higher level, acquisition official. The results of the assessment are briefed at the initiating office level; they are briefed at higher and/or lower levels only by request of the initiating agent.

This paper captures some observations, recurring themes, trends, and lessons learned from these SEI ITA assessment activities. The authors have attempted to abstract the observations and lessons presented here in a way that is not attributable to any given program. Instead, we attempt to present trends and themes that we have observed in several programs, with the hope that program managers can apply these lessons to their programs, and prevent similar situations from recurring.

This chapter outlines the ITA process, and how it differs from that of other program assessments. Subsequent chapters present findings on management practices (Chapter 2), technical development practices (Chapter 3), and infrastructure support issues (Chapter 4).

[1] External expertise is brought in through the use of SEI Visiting Scientists.

[2] OUSD(A&T) has founded the Tri-Service Assessment Initiative. MITRE, Aerospace, The Software Engineering Institute and other FFRDCs also participate in "Red Team" assessment activities.

1.1 ITA Process

The Software Engineering Institute has developed and refined a process for performing ITAs over the last three years, and has had a documented process for the last two years. We have harvested approaches from both the Software Risk Evaluation (SRE) process and other interview-based assessment frameworks.

The process can be outlined as follows:

- Customer Qualification and Contracting
  - Determine customer qualifications.
  - Establish contract.
- Kickoff
  - Create initial team.
  - Initialize information repository.
  - Plan/perform initial contact with program.
  - Review/revise team membership.
- Interview
  - Identify stakeholders.
  - Agree on description of problem and definition of success.
  - Determine milestones and deadlines.
  - Prepare for stakeholder interviews.
  - Perform stakeholder interviews.
  - Capture preliminary findings and recommendations.
- Briefing Preparation and Delivery
  - Review preliminary findings and recommendations.
  - Create draft briefing.
  - Review/refine draft briefing.
  - Perform peer review on briefing.
  - Finalize briefing.
  - Deliver briefing to customer.
  - Perform ITA process improvement activities.

1.2 Types of Programs Evaluated

Most of the programs evaluated have been U.S. Air Force and Navy programs. The programs have all been procurements of software intensive systems, mostly with the following application domain attributes:

- real-time vehicle electronic (avionics, shipboard computing, etc.)
- command, control, communications, and intelligence
- logistics support
- electronics test and evaluation
- satellite ground control

The ITA has usually been sponsored by the associated System Program Office or program executive office. In one case, the activity was sponsored by an informal IPT organization within the program, with the sponsorship and concurrence of program management.

1.3 General Finding Areas

In this paper, we have organized our observations and lessons into the areas of management (Chapter 2), technical (Chapter 3), and infrastructure (Chapter 4). We feel that this best aligns with the interests of the community, rather than the interests of an individual program.

In each of the finding areas we lead with one or more concise recommendations, offset from the main text of the section discussing the finding. Lessons that have strong affinity to each other are grouped together in a single offset. Lessons that have weak affinity are grouped in separate offsets.

2 Management Lessons

This section focuses on management issues, in particular lessons involving personnel, the relationships between acquisition and development organizations, and the technical tracking of programs.

2.1 Communications

Ensure that good communications exist between the program office and the contractors, between contractors cooperating on a development, and within individual contractor development teams. Evaluate this periodically. Any breakdown is a serious threat to the successful completion of a program.

One of the most critical factors to both the long- and short-term success of a program is having effective and efficient communications. This is true at all levels and phases of the program. By communications, we mean the creation of understanding, not just the transmission of information.

There are many possible sources of communications problems. There may be information quantity problems, where either too little or too much information is being passed between parties. There may be content problems, where information is being passed, but it is not the appropriate information, or the information is not correct. There are also nomenclature problems, where words mean different things to different people. Finally, there are interpretation problems, where different parties have different points of view, and come to a different understanding based on the same information.

Lack of good horizontal communications within the development team leads to errors due to incorrect assumptions. Lack of good vertical communications within the development team leads to inaccurate status estimation, dissatisfaction within the team, and an ineffective decision making process.

It is a bad sign when the communications between program partners is reduced to formal channels (contracts letters). This usually means that common understanding has been lost, and vital information either is not flowing, or is flowing at such reduced rates (and increased latency) to make it virtually worthless. For programs claiming to use integrated product teams (IPTs) or Integrated Product and Process Development (IPPD), this is a sure sign that the IPTs are not functioning.

2.2 Acquisition Reform Impacts

Use every opportunity to gain insight into a contractor's performance. IPT/IPPD participation is an excellent way to do this.

Ensure that all critical functional and interoperability requirements are well specified in the contract (statement of work, Statement of Objectives).

The acquisition reform initiatives have had profound impact on programs. While program offices have retained responsibility for controlling their budgets and ensuring that appropriate progress is made, their level of involvement has changed. The shift from having oversight to gaining "insight" into program activities has made this management task more difficult and generated much confusion. Contractor organizations executing programs have also been affected by the new arrangement. Although contractors have frequently protested, "leave us alone and let us do our job, we know how," many recent program failures have shown that contractors (as a group) don't know how. Many are used to being managed by direction, and are not trained to operate under the new rules of Acquisition Reform. This has caused a considerable amount of frustration in both the contractor organizations and the government program offices.

Much of the confusion appears to be caused by the fact that "insight" is not well defined (as compared to oversight), and there is neither an obvious way to gain insight, nor much guidance as to how to go about gaining it. Until the community works through some failures, learns from the mistakes, and transitions the new knowledge throughout the acquisition community, problems will continue to exist and frustration will continue.

The acquisition management problem is made both easier and harder by the associated "upscaling" of contracted requirements. In the past, program offices developed requirements at a fairly detailed level, and contracted with a statement of work (SOW). In many cases now, contracts are let with a collection of high-level operational requirements (Operational Requirements Document (ORD) or, perhaps, a Technical Requirements Document (TRD)), and a more general work statement, or Statement of Objectives (SOO). This has both positive and negative aspects. One advantage is that there is now a single organization—the contracted developer—that can reconcile cost and performance on a program. However, that organization is a commercial entity, whose interest is the maximization of profit. In the past, the program office managed the requirements tradeoffs between user organizations, requirements organizations, and the developer. While this was traditionally one of the most contentious phases of a program, it's execution promoted a balanced consideration of non-profit-driven factors. Under acquisition reform, the developers often have the responsibility to perform all requirements trades, placing them directly in conflict with all of their customers (program office, requirements organizations, and end users).

In addition, since the contractor is not the end user of a system, there is an increased probability that some of the so-called "non-functional" requirements (reliability, dependability, usability) will be compromised inappropriately, as this area is one of the few open for trade.

Combine this transfer of requirements refinement responsibility with a program office performing inadequate "insight" and also suffering from a loss of organic engineering expertise, and you have a recipe for disaster.

2.3 Earned Value

Use EVM, focused on developmental areas of programs, to manage and track programs on a monthly (or more frequent) basis.

The Earned Value Management System (EVMS) is a tool for determining the performance of a development process. Utilization of an EVMS provides information regarding performance management, scheduling and cost [MDAPS 01].

Some form of EVM system has been required for DoD developers since March 1997, but the system is still not well understood by program managers and contractors, especially in complex or non-traditional developmental frameworks (e.g., multiple money sources, iterative development structures, mixed hardware/software developments, COTS/NDI environments, etc.) [Oberndorf 2000]. This is true even though EVMS is an evolutionary development over the previous cost- and schedule- monitoring tool, Cost Schedule Control System Criteria (CSCSC) [CSCSC 95], which was in existence for some 20 years.

In virtually all DoD programs assessed, the EVMS, while present, was not being used by the contractor management, and appeared to be only minimally used by the program offices. This is possibly because of its legacy: that of an accounting "bean counting" tool. This, combined with the latency of the data (typically at least two months lagged, due to DCMA and contractor review), and inappropriate roll-up reporting, limits its impact.

In most of the assessments performed, EVM data was examined and showed that the program in question was in trouble. In many cases, the EVM data refuted contractors' schedule recovery efforts. While contractors were going to operate substantially as usual, their own labor forecasts and deadlines were inconsistent with the calculated EVM System Performance Index (SPI), a measure of developmental efficiency.

This reluctance to adopt EVM is understandable when we consider some of its drawbacks. First, there are several ways for EVM data to be distorted [Morton 2000]. One way has to do with the fact that DoD programs have long lifecycles and several discrete operations. In many cases, EVM data is rolled up to the total program level, rather than providing granular-

ity at the level of significant development activity. This may provide a big picture perspective for the life of a program, but can significantly mask the state of other activities.

Another way that EVM data can be distorted occurs when the program has an incorrect or incomplete contract work breakdown structure (WBS). Lack of an accurate WBS means that earned value can be taken prematurely. In late phases of the program, a point will be reached where it would appear that 100% of the value has been earned, but work is proceeding and costs are increasing. This effect masks CPI and SPI variances in early lifecycle stages, generating surprises in late stages.

Both the manner in which EVM data is rolled up and the funding profile of a program can inhibit the detection of problems. Positive variances in one work line (or a long history of normal performance) can mask negative variances in the same, or another work line. In addition, a combination of developmental and maintenance funding can result in CPI and SPI artifacts, due to changes in the funding baseline.

Finally, some advanced development methods (significant COTS/NDI usage, spiral development methods) introduce their own challenges: it may be difficult to determine what "value" is and consequently difficult to determine when it has been earned.

2.4 Proposal Issues

- **Skeptically evaluate proposals with extraordinary cost savings claims.**
- **Work to acquire or retain enough organic software experience to be able to reconcile cost, schedule, and performance.**

Some of the programs assessed seemed doomed to failure from the beginning. One possible reason for this is that proposals, instead of being evaluated for "best value" were instead evaluated for lowest initial cost. In one case, a mixed hardware/software program, the hardware was procured separately from **and in advance of** the software. The contractor had an insufficient understanding of the actual requirements of the system and subcontracted the software. That subcontractor failed, leaving the prime contractor holding the bag for the creation of software that they did not know how to build. The end result was a program that was a minimum of a year late, at least 100% over budget, and didn't fulfill the requirements.

Other proposal-related problems occur when the program office either does not have a correct internal estimate of the time and cost parameters of a project, or is itself constrained by external forces to a particular schedule, cost, and requirement set. Submitted proposals that do not meet the specified schedule are considered non-compliant and removed from the competition. Contractors adjust their proposals to this reality. Suddenly, internal work estimates that are significantly longer than allowable are reduced, with no relaxation of requirements or increase in cost. We observed one program office accept one such offering, and get a product

that was significantly over budget and whose schedule came close to meeting the original (unacceptable) estimate.

It is the program office's responsibility to prevent these situations. Most program offices have had a significant reduction in staff, and more importantly, in engineering expertise. This lack of expertise increases the probability that there will be an undetected mismatch between cost, schedule, and requirements. Some of this is offset by the use of external engineering support contractors. The government needs to become a smarter buyer. This begins with proposal RFP/RFQ formulation and continues with proposal evaluation.

2.5 Incentives

Think about what kinds of behavior you are punishing and rewarding by your actions. This can range from how you structure the contract, through to how you conduct review and insight activities.

When one talks about incentives in a program management context, one usually means things like award fees. This is (in general) NOT what we are referring to here. Instead, we are being more general. We are asking, "What kind of behavior are we punishing, and what kind of behavior are we rewarding?" This punishment and reward may be performed via an award fee, but is often done, consciously or unconsciously, via program office interference into development organizations (significantly tighter review schedules, external reviews, increased management pressure, etc.).

Some examples of inappropriate incentives are

- awarding contracts to low bidders, who have "bought in" to the program, with no expectation that they can fulfill the original terms of the contract.
- disproportionate responses to announced schedule variances. That is, contractors will take virtually the same amount of impact or "punishment" (in terms of increased review/reporting activities) if they announce that they believe they will be six months late at six months to completion, or if they announce that they believe they will be six months late at one month to completion. Because of this, and the fact that probability is not reality, program offices get late notification of impending schedule slips.

It is in the program office's best interest to be aware of cost and schedule issues early. In contrast, it is generally in the contractor's best interest to notify the program office of these same issues as late as possible. This must change if we actually want to be able to manage programs based on risk, and take proactive measures.

The acquisition community has the right tools available to better give incentive to contractors. These tools include contractor performance assessment report (CPAR) ratings and award fees, as well as less formal and stringent methods. Documentation of the less stringent

methods is needed, and there needs to be better guidance regarding appropriate use of the tools that exist.

3 Technical Lessons

This section focuses on technical issues, in particular lessons involving the engineering of the system, rather than the management and support of the system.

3.1 Requirements Management

In many ways this, and the following topic, are actually management topics. However, they have enough of a technical aspect that they are listed in this section. Proper requirements management can make or break a program. And in the current environment, where the contractor is likely to have TSPR, the responsibility for performing requirements management falls on the contractor. This creates "social" tensions that the contractor has probably not had to deal with in the past, and is likely to create rancor between the other stakeholders of the program (program office, requirements organizations, and end users).

Since the contractor holds the requirements set, and does not hold either the budget or the schedule, this is the area that tends to yield first in the time/money/scope-of-work tradeoff. Some trades may be reasonable or acceptable, such as a reduction in the documentation end deliverables (although this is arguable). Other trades may be less acceptable, compromising either system capabilities or quality attributes (security, fault tolerance, modifiability, etc.) of the system. These trades may, in fact, lead to the development of an unacceptable system. Program offices and end-user organizations must therefore pay careful attention to the contractor's requirements management processes and activities.

3.2 Effort Estimation

- **Utilize most likely effort estimates in proposals and status reports.**
- **Find ways to promote the use of accurate effort estimation and productivity evaluation**
- **Lowest cost is not equivalent to best value. Question outliers.**

The ability to estimate quickly and accurately the scope of a project, and the consequent effort involved is critical to many aspects of a program. This must occur several times with different required levels of accuracy. The first occurrence happens within the program office, before the request for proposal (RFP) is let. In this case, a fairly rough order of magnitude estimate is required to determine if the requirements set, need date, and available budget are consistent. The RFP should not be released unless and until these factors are consistent.

The contractors perform the second effort estimation. This estimate is still fairly rough, and should be a not-to-exceed type figure. This is the estimate that is used in the proposal for the cost and schedule data.

The program office then needs to review the proposals and perform an independent effort estimate based on the approach documented in the proposal. If this estimate cannot be reconciled with the cost and schedule data of the proposal, then the proposal should be considered inconsistent. Either it should be rejected or clarification should be requested.

The contractor then needs to progressively refine the work breakdowns, with associated resource loaded schedules. These are the most detailed effort estimates.

While modeling support exists for performing cost (and associated effort) estimates (e.g., SEERSim, COCOMO-II, etc.), these models are sensitive to "tuning" parameters. It is difficult to tune these models without good local historical data. This makes them valuable to contractor organizations developing cost proposals, but makes their use by program offices problematic. They can, however, be used to perform some gross reasonability checks on existing effort-loaded schedules. The creation of the initial effort estimate ("we need to develop *n* thousand lines of code") is still, however, an art, not a science.

3.3 The Confusion of "Real Fast" and Real Time

With the recent, significant increase in computational capacity, there has been an overloading of the term "real time," which has created confusion. With many business operations systems beginning to call themselves "real time," when in fact it would be more accurate to call them "online" (as opposed to "batch") systems, the unfortunate trend is that anything that is responsive and is operating with recent data is being called a real-time system. Nothing could be farther from the truth.

A real-time system is universally accepted in the engineering field to be one in which time is a factor in determining the correctness of the result. Usually, this means that some deadline exists which, if the system exceeds this time, it can be considered to have failed.

An example of this confusion comes from another definition of real time, this time from WebOPedia:
Occurring immediately. The term is used to describe a number of different computer features. For example, real-time operating systems are systems that respond to input immediately. They are used for such tasks as navigation, in which the computer must react to a steady flow of new information without interruption. Most general-purpose operating systems are not real time because they can take a few seconds, or even minutes, to react.

We submit that this is not a good definition of real time. It would be a better definition of a responsive, online system. Nowhere in this definition is any sense that time plays a role on

either the correctness of the result, or the success or failure of a system. Instead, it attempts to define by example, which is often imprecise.

A key to a real-time system is that it is predictable. Several real-time avionics systems have recently been designed which have questionable predictability. They have been constructed either as data flow systems (which can be real time, if the message rate can be determined/limited and the processing stages of the data flow are known and bounded), or as a consolidated system of formerly physically partitioned subsystems, combined with new functionality, and running in a shared computational system without any thought to schedulability. For example, one program had a safety critical timeline that could not be verified because no analysis had been performed.

A "real fast" system cannot fail (produce incorrect results) if it exceeds its "deadline." At worst, systems of this type can yield less valuable results (because the results are aged), or generate significant user dissatisfaction.

3.4 The Challenge of Reuse

Reuse in a mission-critical, real-time, or embedded system project presents many challenges, both technical and cultural. The engineering of these systems is complex, and dependability is an almost universally assumed attribute. Some of the challenges of reuse are

- There may be limited knowledge and experience with the reused item in the development team.
- The reused item contains assumptions as to its operating environment. These assumptions may not be explicitly stated, and must be verified for proper operation.
- There may be architectural conflicts between the reused item and the remainder of the system.
- There are cost and effort estimation implications in the use of reuse products.

There are some cultural challenges that can occur in reuse environments. For projects that are consolidations of sets of existing products, it is easy to take the position of "this aspect of the system has been done before, so it is not difficult." This fails to account for all of the technical challenge areas above, and can lead to serious program failure.

Many of these challenges also exist with the use of COTS components.

3.5 COTS Component Selection

- **Define a set of evaluation/selection criteria and then use it.**
- **Keep up with what is happening in the market; you may have to change or upgrade products.**

Many programs have gotten into trouble by committing to COTS products without performing either an adequate market survey or a comprehensive product evaluation. Problems with COTS products, either components or support tools, can plague a program for its entire lifecycle. Some simple factors to consider when performing a COTS product selection for a program are

- vendor reputation
- vendor stability
- maturity of all product components
- suitability of product
- interoperability/compatibility with other products
- product cost (lifecycle, per developer seat and runtime/royalty costs, yearly licensing, etc.)
- product license issues
- competition and history in this product area
- product migration/evolution plans (scalability and evolution)

In addition, once a component has been selected, the program team should keep aware as to the vendor's intention and future direction of the selected product(s). This should be done for reasons involving diminishing manufacturing sources (DMS), as well as for impacts on future product suitability and end-system lifecycle cost impacts.

3.6 Reliability and Fault Tolerance

Handle requirements that have architectural consequences as systems engineering issues—up front.

Reliability and fault tolerance are two quality attributes that are typically mishandled in programs. In many cases, program teams attempt to implement these attributes in a "back-end" fashion, late in the program by attempting to test-in quality/reliability. Program teams also attempt to add fault identification/fault isolation and redundancy management at late stages. This is not cost effective and is frequently unsuccessful.

W.E. Deming demonstrated decades ago that it is not effective to test in quality; it needs to be designed in. In other words, this means that quality (reliability) is a systems engineering level attribute, which must be satisfied via a combination of system architecture and implementation of defined processes. Similarly, fault tolerance is a systems engineering level activity that must be handled architecturally.

This being the case, software engineers have to participate in the systems engineering process. In this process, all of the quality attributes (reliability, fault tolerance, performance, modifiability) must be considered together, with appropriate weighting. The attributes must be traded off against each other in a considered fashion, to come up with an overall systems architecture that can support both the needed functionality and the needed quality. This is a difficult process, and may prove to be the cornerstone of engineering software-intensive systems.

4 Chapter 4 - Infrastructure

Infrastructure support for programs is an interesting interaction of management and technical concerns. It is a technical implementation, which requires strong management support to succeed. The tradeoff between overhead cost drivers and return on investment of enabling technologies is a difficult one, which is often done poorly or unconsciously. This section addresses some infrastructure issues that are enabling technologies, and that are sometimes indicators of the success or failure of a program. Note that these issues are of more concern in large distributed development activities.

4.1 Data Communications

In distributed development activities, get high quality, secure, broadband communications between sites. It is an enabler, not a cost.

In a distributed development activity, distributed collaboration tools are used to avoid travel to the various development sites. The underlying enabling technology behind these distributed collaboration tools is the data communications they are hosted upon. Having an appropriate communications infrastructure environment appears to be a success differentiator in distributed development activities. An example which presents this follows:

> Two avionics systems development programs are going on at the same time, involving some of the same organizations. One activity (Activity 1) has its prime software development ongoing at a single site, with systems engineering, program management and support activities occurring at remote sites. The other development activity (Activity 2) is distributed between many contractors and many sites across the country.
>
> Activity 1 has gone to no particular effort to establish a data communications environment between its engineering activity sites. The interaction between top program management and local program management is poor. The interaction between the systems engineering/design groups and the implementation/test activities is also poor. Data generated at one site is analyzed at a second site, with inadequate turn-around time and little or nor opportunity for clarification. There is little or no feedback on consequences of design decisions until implementation is complete and the subsystem is in test. The program is significantly over cost and schedule, has personnel retention problems, and seems unable to forecast its completion to within even an order of magnitude.

Activity 2 has taken care to establish high speed, high quality, and secure data communications between its development sites. Integration lab facilities are located across the country, with various levels of fidelity. Lab data is easily transmitted between integration facilities and analyst staff. Collaboration environments exist which allow staff at various locations to share documents and other data in a near-seamless fashion. This program is also over cost and behind schedule, but this is attributable to the scale and security requirements of the project.

There are many contrasting factors between these programs, but one critical one is Activity 2's adoption of high-speed data communications to help integrate their distributed staff.

In the not too distant past, data communications links were expensive and difficult to justify. With the Internet revolution and the increase in distributed development activities, bandwidth is easily available at reasonable cost. Virtual Private Network (VPN) technology helps to create normally secure environments. DoD link and packet level encryptors implement a similar capability with known characteristics. There is little justification in the current world for not having distributed developments interconnected and interoperable.

4.2 Common Development Environments

For distributed developments, use a common development environment at all sites.

With acquiring and retaining staff being an ever-increasing problem in today's market, increasing the productivity of your development staff is a significant concern. There are at least two ways to go about achieving this increase in productivity. Program management can 1) increase the individual productivity of development team members through the use of process, improved tools, and incentives; or 2) improve the portability of development staff so that resources can be easily migrated between different aspects or efforts of a development activity. The use of a common development environment (CDE) across a distributed development touches on item 1 and directly impacts on item 2.

Having a CDE supports the use of common processes among distributed development sites. It also supports staff mobility, both in performing a single work duty at multiple sites (they know that their expected tools are available at any site) and in performing multiple duties over time. It also cuts down on the re-training required when a staff member moves from one development activity to another within a program.

The downside of CDEs is that they are likely to be compromise environments. That is, they will be comprised of tools that are either general purpose, or represent a tradeoff of capabilities required by several different development activities, rather than an ideal point solution to the problems of an individual work group.

Because CDEs are compromise environments, put together via a negotiation between workgroups, they are especially prone to the next issue area: infrastructure currency and tool modernization.

4.3 Infrastructure Currency Issues

Since most development infrastructures consist of a collection of Commercial Off the Shelf (COTS) products, or other locally developed or non-developmental item (NDI) software components, they are subject to all of the normal problems of this class of software. Two of the most significant items are 1) receiving new required capabilities and 2) maintaining some level of product currency.

Tradeoffs exist in this area that have no simple solution. One tradeoff is that of stability vs. required capability. Other issues that arise are: vendor support; training and re-training staff; data migration efforts between versions or products; and legacy deliverable item configuration control. Every program team has to make this tradeoff on its own, to satisfy its program's unique capability and stability requirements.

One thing that can be stated unilaterally is that the ease of product migration and/or version upgrade should be one of the strongly weighted factors in any program's product evaluation and qualification efforts. Programs need to consider the effects of the COTS marketplace on software products similarly to how they treat end-of-life and diminishing manufacturing source (DMS) issues in other COTS hardware products. Failing to do this will adversely impact the lifecycle cost of a program, and may jeopardize the supportability of the software products.

Products or product families that conform to standards (IEEE, IETF, OMG, etc) tend to have less volatility, known (or knowable) upgrade paths, and a wider selection of equivalent (and often interoperable) alternatives.

5 Conclusions

All of the assessments summarized in this paper were on large scale, DoD (or related government agency) programs. All of the programs were in actual or perceived difficulty. Some of the recommendations were for substantial restructuring or cancellation of the effort.

With this in mind, we look to some of the root causes of the problems uncovered, and attempt to compare and contrast them to similar works in the non-defense world [Flowers 96]. In doing this, we find that there are more similarities than there are differences.

The most significant drivers to failure on these systems continue to be management and culture related, just as they are in commercial systems. Technological failings, while they exist, also have a strong management flavor, as they tend to cluster around failings in the systems engineering process. There are no technology "silver bullets," and anyone promoting any technology as a panacea should be viewed with suspicion. A recent Defense Science Board report states: "Too often, programs lacked well thought-out, disciplined program management and/or software development processes. … In general, the technical issues, although difficult at times, were not the determining factor. Disciplined execution was." [DSB2000].

There are numerous examples as to how this lack of disciplined execution manifests. Some deficiencies are related to human nature. Self-interest leads people to primarily consider their tenure on a job, cleaning up problems left for them by their predecessors and often not considering long-term consequences of short-term decisions. There is also a tendency to try to place blame on other organizations: customers and program offices cannot hold to a set of requirements; contractors don't live up to their obligations; vendor's products don't live up to their performance and capability claims. It is obviously someone else's fault. This is all a case of lack of discipline. We find that in programs in trouble, there are NO innocent parties. All stakeholders involved participated (at some level) in creating or abetting failure.

And failure, at least in software-intensive systems, is fairly common. Most DoD system procurements are inherently high-risk activities. Every modification or new system must be a significant advance over current state, while offering a reduction in lifecycle costs. There are few opportunities to learn from the mistakes of others. There is a pressure (based on Acquisition Reform and current DoD acquisition policy) to use COTS hardware and software, but there are few clear opportunities to utilize COTS components beyond the infrastructure level, because no other activity in the world involves putting "steel on target." The opportunities for technology reuse must therefore come from within the community, which has little or no cultural foundation for creating reusable software. Program offices typically have no money

to invest in components that are designed for reusability. Until this culture changes, there will be very limited use of COTS/NDI in weapon systems, and large potential long-term cost savings will not be realized.

References/Bibliography

[Baldwin 00] Baldwin, Kristen. "OSD Tri-Service Assessment Initiative Overview," *Proceedings of the Software Technology Conference 2000*. Salt Lake City, Utah, April 30-May 5, 2000

[Brandt 00] Brandt, Thomas. "F-22: Successful Software for the Millennium," *Proceedings of the Software Technology Conference 2000*. Salt Lake City, Utah, April 30-May 5, 2000.

[Cook 00] Cook, David. "Confusing Process vs. Product: Why We Have Problems Producing Quality Software," *Proceedings of the Software Technology Conference 2000*. Salt Lake City, Utah, April 30-May 5, 2000.

[CSCSC 95] DoD Implementation of the Cost Schedule Control System Criteria (CSCSC). Defense Systems Management College, Ft. Belvoir, VA: 1995. http://www.dsmc.dsm.mil/r/port/r/reg2848a.htm

[DSB 00] Report of the Defense Science Board Task Force on Defense Software, November 2000. http://dodsis.rome.ittssc.com/FinalDSB.pdf

[Flowers 96] Flowers, Stephen. *Software Failure: Management Failure*. New York, NY: Wiley & Sons, 1996.

[Lane 00] Lane, Jo Ann. "Distributed Software Development: Pulling Together the Right Technical Resources to Do the Job," *Proceedings of the Software Technology Conference 2000*. Salt Lake City, Utah, April 30-May 5, 2000.

[Lamothe 00] Lamothe, Steven. "Applying a Secure, Web-based Collaborative Environment to Software Projects: Strategies for Success," *Proceedings of the Twelfth Annual Software Technology Conference*. Salt Lake City, Utah, April 30-May 5, 2000.

[MDAPS 01] DoD 500 2-R. "Mandatory Procedures for Major Defense Acquisition Programs (MDAPS) and Major Automated Information Systems Acquisition Programs (MAIS)." January 2001.

http://www.acq.osd.mil/ar/doc/dodd5000-2-r-010401.pdf

[Morton 00] Morton, Barry. “Applying Earned Value to Software Development: Lessons Learned,” *Proceedings of the Software Technology Conference 2000*. Salt Lake City, Utah, April 30-May 5, 2000.

[Oberndorf 00] Oberndorf, Patricia; Sledge, Carol; & Staley, Mary Jo. “Earned Value for COTS Based Systems,” *Proceedings of the Software Technology Conference 2000*. Salt Lake City, Utah, April 30-May 5, 2000.

[STSC 00] Department of the Air Force, Software Technology Support Center. *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems; Command and Control Systems; Management Information Systems* (Version 3.0) May 2000.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

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

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

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Real-Time Systems: Engineering Lessons Learned from Independent Technical Assessments	F19628-00-C-0003

6. AUTHOR(S)

Theodore Marz, Daniel Plakosh

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-TN-004

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	

The Software Engineering Institute (SEI) has performed several Independent Technical Assessments (ITAs) on mission-critical/real-time systems for the Department of Defense and other agencies.

This paper contains observations, recurring themes, trends, and lessons learned about systems development as derived from real-time/mission-critical programs that have been reviewed over the last three years.

It is hoped that the observations contained in this paper will be of value to future program managers and help ensure their success.

14. SUBJECT TERMS	15. NUMBER OF PAGES
real time, independent technical assessment, ITA, acquisition reform, commercial off the shelf, COTS, reuse, earned value, incentives, reliability, fault tolerance.	28

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