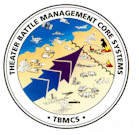
**RESEARCH REPORT**





**Systems Reengineering Process Improvement Plan, V1**

Authors: Ryan Patton

Confidentiality: Public

**Executive Summary**

In the 1990’s, Lockheed Martin worked towards building an all-encompassing command and control service for the Air Force called Theater Battle Management Control System. Although the Theater Battle Management Core System was eventually realized through an iterative, spiraled systems engineering approach, the first version fielded experienced many operational test errors that set the schedule back years and cost Lockheed Martin and the government millions of dollars. Heading into the first fielding of Theater Battle Management Core System, simple process improvements could have improved the overall performance of the system, saved Lockheed Martin and the government millions of dollars, maintained a better schedule, and achieved better technical progress. The content of this paper studies root causes and covers five acts by which LM could introduce software selection improvements: Define, Measure, Analyze, Improve, and Control.

**Table of Contents**

[1. Introduction 7](#_Toc39570192)

[1.1. Document Purpose/Problem Statement 7](#_Toc39570193)

[1.2. Document Scope 8](#_Toc39570194)

[1.3. Document Update 8](#_Toc39570195)

[1.4. Program Summary 9](#_Toc39570196)

[1.4.1. Program Objectives and Scope 9](#_Toc39570197)

[1.4.2. Project Summary and Deliverables 9](#_Toc39570198)

[1.4.3. Technical Description 10](#_Toc39570199)

[2. Measure 10](#_Toc39570200)

[2.1. Addressing the Problem 10](#_Toc39570201)

[2.2. Measurement Validation 11](#_Toc39570202)

[2.3. Baseline Data Collection 13](#_Toc39570203)

[2.4. Data Validation 14](#_Toc39570204)

[3. Analyze 14](#_Toc39570205)

[3.1. Process Models for Determining Potential Root Causes 14](#_Toc39570206)

[3.2. Methods for Evaluating Potential Root Causes 16](#_Toc39570207)

[3.3. Identification of Root Causes 18](#_Toc39570208)

[4. Improve 18](#_Toc39570209)

[4.1. Trade Studies 18](#_Toc39570210)

[4.1.1. Software Selections Trade Study 18](#_Toc39570211)

[4.2. Solution Feasibility 20](#_Toc39570212)

[4.3. Design of Experiments 21](#_Toc39570213)

[4.4. Challenges and Compensation for Implementation 22](#_Toc39570214)

[4.5. Implementation Plans 23](#_Toc39570215)

[5. Control 24](#_Toc39570216)

[5.1. Implementing and Monitoring Control Plan 24](#_Toc39570217)

[5.2. Measured Progress and Success 25](#_Toc39570218)

[5.3. Organizational Improvements for Success 26](#_Toc39570219)

[6. List of Acronyms 26](#_Toc39570220)

**List of Figures**

[Figure 3-1. Cause-and-Effects Diagram (Fishbone Diagram) 3-](#_Toc29846044)13

[Figure 4-1. Solaris 11 SPARC Performance 4-](#_Toc29846044)18

**List of Tables**

[Table 3-1. Comparative Analysis 3-](#_Toc29845658)14

[Table 3-2. Cause-and-Effect Relational Matrix 3-](#_Toc29845658)15

# Introduction

## Document Purpose/Problem Statement

Revisiting Lockheed Martin’s (LM) Systems Engineering (SE) process of the Theater Battle Management Core System (TBMCS), an integrated air command and control (C2) system that performs standardized, secure, automated air battle planning and execution management for Air Force (AF), multi-service, and allied commanders in theaters of operation worldwide, revealed many underlying issues surrounding its execution through every phase of the product lifecycle. A case study authored by folks from The MITRE Corporation and Lockheed Martin Integrated Systems and Solutions (LM-ISS) detailed five key systems engineering learning principles: requirements definition and management, system architecture, system/subsystem design, system integration, and validation and verification. The aim of this document is to revisit the actions taken by both the contractor and government leading to less-than-ideal SE practices and problems arising throughout the lifecycle which could improve general practices moving forward. Many problems contributed to schedule setbacks, drawing over-budget, and lack of technical progress. The intent of the following sections is to analyze the system as a whole to target one specific problem within the TBMCS, with the ultimate goal of enlisting definitive steps for improvement.

As mentioned, five SE learning principles contributed to progress impediment [2]:

1. The lack of definitive requirements set forth by both LM and the government’s military forces from the initial design phases and throughout the product lifecycle caused vague and lax building and testing standards, drove the project over-budget and over-schedule, and foreshadowed a lack of accountability by both parties. The lack of operational requirements set the program back a minimum of 4 years.
2. The system architecture needed further definition at lower levels to improve its system design and development details. Misalignment between LM and the government’s organizations caused frequent issues with software reuse with commercial software products. As a result of the layered architecture, migration to modern technologies was difficult and did not support significant system evolution from the legacy system.
3. The complexity of legacy applications, misunderstood maturity and complexity of commercial and third party software products, and lack of understanding of system employment by the user all hampered the system’s design and subsystems’ design.
4. The complexity of the system caused problems with system integration, stemming from lack of detail in the system architecture, mandates to use government-furnished equipment that was incompatible with commercial off-the-shelf products, the oversight needed for third party software product integration, and difficulty testing external system interfaces.
5. Validation and verification were difficult due to a lack of firm requirements baseline. No clear measures of success were set and often tests were run in parallel to meet schedule deadlines, ultimately affecting the inability to replicate the operational environment prior to acceptance test.

The complications arising from software collaboration between the government’s organizations and LM caused significant software integration delays of 6-18 months early in the product lifecycle with estimates between $5-$30 million for the amount the program overspent in software training in an effort that foreshadowed lax building and testing standards [3]. The lack of software training processes contributed to technical progress impediments, schedule delays, and budget overdraws.

Analysis of these SE principles will guide the reengineering process to practical systematic improvements that in retrospect could improve the schedule, budget, and technical progress. Significant attention will be given to the process for deciding on what the root cause(s) was so the process for improvement is driven by definitive metrics and follows a logical pattern. Relating back to the faults of the TBMCS, further details will break down what lessons could be learned for future contract work with the government’s military forces.

## Document Scope

The detailed analysis of the TBMCS encompasses all systems, subsystems, components, both hardware and software, third party products, both hardware and software, contractors, and government associations pertaining to TBMCS traceability. The process reengineering document fulfills documentation for any activities associated within the TBMCS lifecycle and any organizations associated with the TBMCS system. All activities, products, processes, tools, controls, integration technologies, proprietary information, intellectual property, security measures, key personnel, and safety guidelines associated with the TBMCS systems, subsystems, and components are retained within the scope of this document, even if not directly referred to. By doing so, the systems engineering approach from design through retirement can be controllably studied for imperfections in the SE process LM and the government’s military organizations used. Although third party applications may be studied in how they relate to the TBMCS system, the system’s studies must directly relate to the impact on the system under scrutiny.

The retrospective analysis inspected throughout the duration of the TBMCS program will capture LMs SE process for problem development and base recommendations on the unbiased findings.

## Document Update

By revisiting flaws in the SE process utilized by LM and the government, contrasting evidence may come to light on what impacted the cost, schedule, and technical progress of the system. At each instance, the Process Reengineering document may undergo revisions as seen fit to accurately reflect the discovery of new information. Periodical revisions capture the integrity of the Process Reengineering document to thoroughly analyze the TBMCS system.

The completion of the process reengineering draft will add new material on a weekly basis, following a timely order of the Table of Contents (TOC). Each week, the focus will largely revolve around the addition of new material but editing previously drafted sections of content may occur on an as-needed basis. Any revisions will be documented and undergo official review for entry on a bi-weekly bias where a new revised draft will be released.

## Program Summary

### Program Objectives and Scope

To monitor the reengineering process, objective thresholds and measurements are set. By analyzing the intricate flaws in one of the key systems engineering learning principles, a more well-rounded understanding can determine measures to be taken in the future to prevent reoccurrence. The TBMCS system will be studied as a whole before decomposition into the smaller, more manageable Problem Statement to show how the system-level problems interact with the problem studied herein. No problem exists within a bubble and the reengineering process intends to capture as much indirectly related factors as possible. The accountability of the Problem Statement’s analysis depends on the ability to detail the relationships between all interacting variables.

The process reengineering document intends to devote sufficient attention and time needed to ascertain the impact a number of issues arising throughout the lifecycle had on the ability for LM to deliver an on-budget, timely, and quality product to the government. While a significant burden lies on LM for the quality of the final product delivered, the government’s relationship will be examined to show the struggles of collaboration between LM and the government in this particular instance.

Limitations on the analysis mostly stem from the challenges to retrospectively analyze a system and on the limited documentation available. By studying the system post-deployment, problems under scrutiny may only seem relevant looking back and would go unnoticed in real-time. Documentation only covers certain perspective on what the issues around the TBMCS were and are by no means an exhaustive, all-encompassing list, inherently carrying bias based on who the authors were relative to identifying problems. In addition, the TBMCS system can mostly be studied from the limited manufacturing perspective, not the operational perspective due to the amount of classified information not being readily available.

Although the process reengineering document will be independently evaluating process improvements in the TBMCS system to apply to defense and commercial projects moving forward, the document will consult other documentation sources for gathering data and understanding the TMBCS system. The sources of information contained herein to assist with the reengineering process support the residing analysis and conclusions.

### Project Summary and Deliverables

Key milestones of the reengineering process will chronologically show the progress in developing a process improvement. Major phases following a Lean Six Sigma approach consist of the subsequent activities:

1. Define – April 6th, 2020
2. Measure – April 13th, 2020
3. Analyze – April 20th, 2020
4. Improve – April 27th, 2020
5. Control – May 4th, 2020
6. Organizational Learning and Leadership – May 12th, 2020

The deadlines represent analytical processes but do not encompass the scope of all activities needed to understand the TBMCS reengineering process. During the lifecycle study of the TBMCS, periodic assessments will dictate any schedule adjustments for sections requiring further analysis.

### Technical Description

Understanding the TBMCS system starts with allocating plenty of time for retrieving relevant information related to the TBMCS and consuming as much of that information as possible. While effective analysis does not necessarily require exhaustive documentation to be studied, the confidence in relaying effective recommended solutions relates back to how extensively the problem(s) could be studied. While the main focus of the process reengineering document will be on improvement measures related to communications between LM and the government, trade studies will reveal a logically following solution to the identified root cause(s). Problems affecting progression metrics that were intended to guide the TBMCS design process and further on will be explored to determine their cross-functional relationships. Without proper processes in place to examine potential systematic problems, the technical program measures could not be realized. Each subsystem and aspect of the system critical to LM for delivering its product on-time and on-budget will receive their own allocated trade studies. From the trade studies, the impact the root cause(s) had on each aspect of the system can be better understood.

While the intent of the process reengineering document is to analytically determine definitive measures for direct process improvement within the implementation of the TBMCS program, some sections will contain information pertinent to the technical details of the subsystems. The technical details provided for various aspects of the system will be instructively placed in context to relate back to the overarching general improvement concept. Technical details will aim to further comprehension of the system’s interactions where intuitive understand lacks. Technical details may include processes, tools, software, hardware, system-level design concepts, subsystem-level design concepts, SE concepts, and business development concepts.

# Measure

## Addressing the Problem

The lack of software guidance set forth by both LM and the government’s military forces from the initial design phases and throughout the product lifecycle caused vague and lax building and testing standards, drove the project over-budget and over-schedule, and foreshadowed a lack of accountability by both parties. The lack of software cohesion and training set the program back a minimum months if not years. Due to the government’s failure to suggest easily integrated software products and develop a more efficient software training plan the program spent significant amounts of time and effort improving the legacy system instead of adding in design to an improved current system.

If software programs were embedded into the initial negotiations with the representing government bodies and LM, the TBMCS would have achieved greater levels of traceability and accountability. The TBMCS did not need software programs convenient for one side of the negotiations, the program needed jointly agreed software programs, even if commercial, to assess system performance and leave little open to interpretation. The directionless software choices, set much later on in the product lifecycle, forced LM to accommodate what the government thought would be easily integrated software solutions. The deployment of the software nor the interaction among operation centers and the system never even reached fruition. Over 20 organizations were involved at some capacity in the TBMCS software selections and training in the year before TBMCS underwent its first operational test and evaluation, with a complicated architecture for developing the system’s software and network design in the TBMCS from the top down. This lead to a chaotic clash between the organizations. In addition, there was a lack of formal concept of employment, tactics, techniques, and procedures (TTPs) which were inconsistent and varied, and there was never an agreed on formalized measures of performance leaving vague stipulations for what kind of software selection would be needed for optimal performance. LM and the government failed to establish a distributed learning environment inductive to adequate funding, the best training materials, and delivery of quality products on time to all users.

From the start, the TBMCS was supposed to provide a common and shared operations and intelligent database with a common suite of tools to plan, manage, and execute the air battle plan (ABP), and was to include a common operational picture for shared situational awareness. Ultimately, the decision flow generating difficult-to-manage software expectations affected not only the system as whole but at a system-of-systems level. The difficulty in determining the impact of software selections was directly seen during operational testing and schedule delays. Due to the massive failure of the software selection and prohibitive training for the TBMCS system, the performance baseline was volatile up to system acceptance, long after operational tests and evaluation were performed.

## Measurement Validation

A number of measurements could be implemented and tracked to improve the problems arising from a lack of definitive requirements early on in the TBMCS program.

Since the software selections drive early development and design decisions, project performance measures may have included cost performance index (CPI), schedule performance index (SPI), defects per single line of code (SLOC), customer complaints, corrective action requests, inquiry response time, and defect containment.

The process of their performance could be guided by defects (deficiencies) per unit (DPU), defects (deficiencies) per million opportunities (DPMO), rolled throughput yield (RTY), sigma levels, and process capability indices (PCIs). This group of measurements would help determine the progress and consistent quality of the end-product. Since a legacy system is already in place, assuming the data is available, LM members could compare their relative progress and success with the legacy system to get a rough idea of their progress. Defects can be relative to the difficulty of the integrated technology and schedule pressures but overall the number of defects should be very similar to the legacy system, barring any major issues during development. For example, if 50 software bug defects were identified per million opportunities in the legacy system, if the TBMCS system was showing 500+ software bug defects per million opportunities this would signal major issues arising during the process flow. DPUs in the TMBCS would be a more difficult measure just because the units of the TBMCS would most likely only break down to theaters which have a higher variance than more simplistic sub-systems.

The communication of the software decisions and training could use a top-down flow or downward flow approach, a bottom-up flow or upward flow, and horizontal communication. In the top-down flow of information, upper management in the TBMCS would directly communicate to all the workers below them in the organizational chart. While the message may come as more inspiring and invoke a sense of connection, it could also risk coming off as condescending to middle-managers. In the downward flow approach, middle-management would take more control of the communications by receiving instructions to relay to those below them. This would risk the trustworthiness due to lack of transparency with upper management and those low on the organizational chart. The other approaches basically show how information would flow up to upper management but the key takeaway is transparency by upper management leading to the lower-level organizational chart members feeling free and trusted to voice their dissenting opinions without repercussions.

The TBMCS program lacked communication between LM and the government. The government would be paying for the features of the TBMCS and LM appeared to dance around demanding wants from the government and asking questions they would need to make the TBMCS a more definitive design early on. The government for the most part would take for granted software demands because they most likely wouldn’t know the direct technical impact introducing new software would have as LM could if they worked with them more regularly. The government would just name their functionalities vaguely and hope LM would produce the proper paper trails and technical progress to monitor progress on the feasibility of each feature. Since a legacy system was already in place, the government just stated what they would like improved over the legacy system without a firm grasp about what it would take to accomplish their desired features. This lead to additional confusion because the government could then demand a certain functionality they did not explicitly request but actually wanted be incorporated into their system. By going through an in-depth discussion of software functionality, LM could reveal features to the government that wouldn’t be obvious by just talking about top-level functionality of the system.

International Standards Organization (ISO) 9000 and 9001on quality management systems (QMS) and compliance and ISO 5807:1985 on information processing may help with process flow and mapping. Organizations distribute process information to their suppliers and customers usually in an effort to obligate their contract. By complying with ISO 9001, the written procedures and work instructions written by LM would help communicate necessary information to the government consistently. By capturing what is done in the process, why it is done, where it is done, when it is done, who does what, and how it is done the government retains the desired traceability for a multitude of program and process improvement factors. If a high-level supplies, inputs, processes, outputs, and customers (SIPOC) map was in place, the cause-effect relationship of requirements could be visualized by a fishbone diagram. After the cause-and-effect diagram with additions of cards (CEDAC) was brainstormed and analyzed, a list could be composed ranking the priorities of the enterprise. They could then be visualized in a Pareto chart where LMs resources could be put into issues critical to the government or issues that have more financial impact where they would be weighted.

A number of measurements could be tracked to show the program’s overall progress. LM seemed to lack basic measurements that would have foretold the struggles they would encounter down the road. Measurements stemming from software and training such as technical performance measures (TPMs) and measures of performance (MOPs) could have defined what specifically the government wanted and what benchmarks they needed to hit going into operational tests instead of just showing the TBMCS was an improvement over the legacy system. More detailed plans could have been generated showing what technical processes needed to be put in place to reach features requested by the government without schedule delays.

As with any program, LMs TBMCS program as operating under a number of constraints vying for resources. By amassing data with statistical significance, valid conclusions may be drawn. The population of the software selections may have included statistics such the binomial distribution to show the number of defects in the software as a result of the software cohesion. Quality and other divisions within TBMCS must decide what measurements are critical to determining how their product is coming along with the implementation of software early on.

## Baseline Data Collection

Some data collections methods LM could utilize with government are surveys, face-to-face interviews, focus groups, mystery shopping, customer feedback, automatic data capture, and manual data capture. From the methods, one-on-one methods retain a higher integrity and provide opportunities to clarify with respondents. Surveys would have a lower response rate and can mislead the end-results. Some pitfalls that the data would aim to avoid:

* Undefined units of measure
* Illegible handwriting
* Inadequate measurement system resolution/discrimination
* Precision error
* Emotional bias
* Guesswork or personal bias
* Multiple points of data entry
* Poor instructions or training causing erroneous data entry
* Ambiguous terminology
* Clerical or typographical errors

With the main issue stemming from hard-to-integrate software selections, the TBMCS program would have benefitted immensely by constant feedback from the customer, the government’s military forces. The vision of the government branches involved with the program did not align with LMs vision because of the disconnect between the two, leading to a lack of baseline target performance measurements. While there is little indication about the amount of automatic and manual sampling from the technical process aspects of the TBMCS, case studies on the program seem to point the finger at the confusion spreading due to what measurements would be critical to proving their product’s functionality.

Ideally, LM would get together with the government and make a definitive list of software selections for each sub-system and functionality, not mere demands by the government later on. From that point, technical performance measures (TPMs) and other measures stemming from the software could be set and agreed upon by LM and the government.

## Data Validation

Many of the technical measurements’ validation would derive from the software performance set forth. Many technical measures will have upper and lower specification limits with designated criteria such as “acceptable”, “unacceptable”, “exceptional”, etc. Some pitfalls LM would look out for in the measurement system analysis (MSA) are:

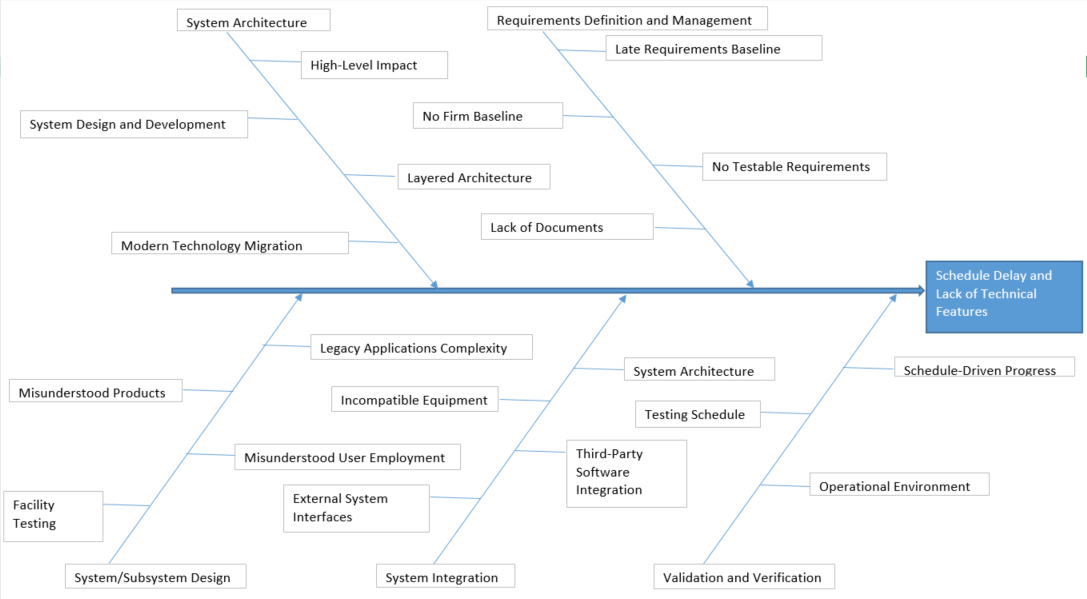
* Calibration
* Stability
* Repeatability
* Reproducibility
* Linearity
* Bias
* Accuracy
* Precision

LM could conduct a gage repeatability and reproducibility (GR&R) or similar study in hopes of figuring out what measurement variables are important with what parts over many trials. A GR&R would relate back to the avoidance of pitfalls listed above to understand variation in the production process.

# Analyze

## Process Models for Determining Potential Problem Causes

With the problem defined and data inputs captured in the measurement phase, the problem may now be understood at a level where an in-depth analysis could yield benefits for LM, the stakeholders, and the TBMCS. In determining what caused TMBCSs progress to be delayed by years, a variety of causes were investigated and analyzed by cause-and-effect diagrams. The cause-and-effect diagram shown in Figure 3-1 details a high-level breakdown of key factors leading to schedule delays and lack of technical features in the end-product.



*Figure 3-1. Cause-and-Effects Diagram (Fishbone Diagram).*

The Fishbone Diagram above lists off 19 potential causes contributing to the main process improvement problem, third-party software integration. The key causes are broken down into their 5 major divisions for cause contribution: system architecture, requirements definition and management, system/subsystem design, system integration, and validation and verification. In Table 2, each process input from the Fishbone Diagram is combined in a relational matrix with each of the 5 cause contribution divisions. By doing so, a wider system perspective can be observed showing how each process input affects the system as a whole. The process inputs can now be compared even in divisions where their impact may not be directly impacted.

The Comparative Analysis seen in Table 2 directs attention to problem diagnosis to refine the ranges or batches where defect(s) source(s) emerged. The difficulty in applying this to the TBMCS system lies in the lack of documentation supporting what explicitly would define a defect. The TBMCS did not necessarily produce an abundant defects throughout its lifecycle but failed to define what software upgrades would be needed to achieve the features desired by the government’s organizations.

*Table 3-1. Comparative Analysis*

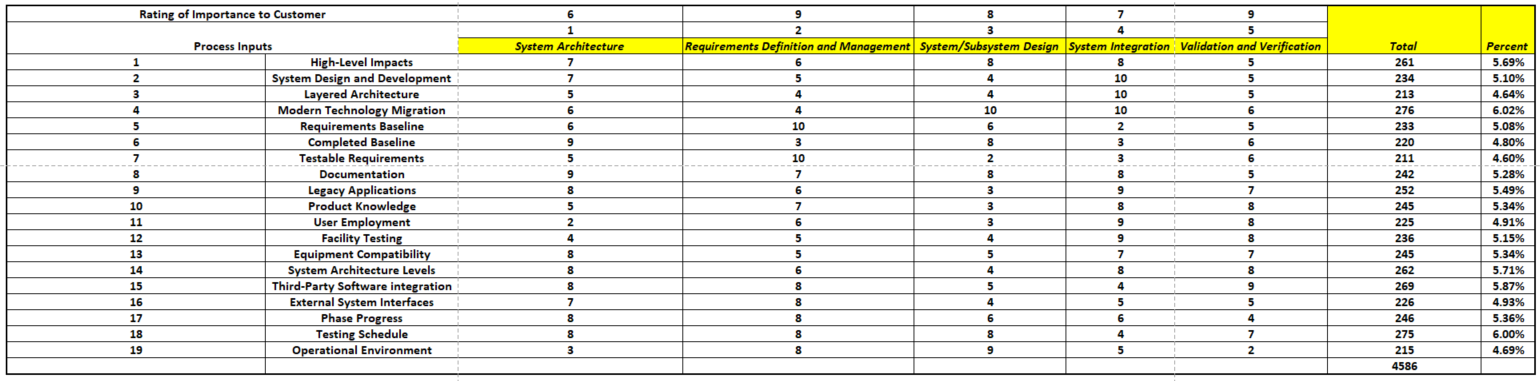
|  |  |  |  |
| --- | --- | --- | --- |
| Problem statement: | Schedule Delays and Overdrawing Budget | | |
|  | **Is** | **Is not** | **Differences and changes** |
| What | Software bugs and integration complexities | Legacy system baseline to improve upon | Mutual agreement on which software applications to utilize from beginning |
| Where | Mostly in the virtual software space with some hardware interfaces | Defects most noted in lack of functionality in testing and operations | Isolating software bugs before testing on hardware, allow for separate testing for each software integration |
| When | Defects at the design stage, mostly stemming from stringent measures on what would be considered defects | Defining what qualifies as defects more |
| Size | Defects scattered throughout whole system, ultimately affecting lofty operational testing requirements/goals | More consistent testing measures and procedures |

Breaking down Table 1, no single defect nor issue produced the problems brought on by the TBMCS. No indication was given that the output of code was specifically causing defects other than issues stemming from the integration of various software products which would not necessarily be defects but would cause delayed schedules. The Is/Is Not Comparative Analysis may not be the most applicable for this system because of its heavy reliance on software functionality. The analysis, however, does highlight how the TBMCS could not produce definitive results to even highlight definitive measures on what would qualify as a defect. The legacy system was supposed to work out all of the kinks in the software development and reduce defects. The main schedule delays happened long before operational testing, when the software was still being implemented and hadn’t yet been tested.

## Methods for Evaluating Potential Root Causes

Many methods and statistics exist to aid in find the problem’s root cause. The sample size of the data would need to be evaluated to determine the significance under evaluation. By testing the means, the data would receive validation in its normality to validate the null hypothesis. A t-test and Z-test could then make inferences about the population mean, with the population variance unknown. The mean, variance, confidence intervals, and proportions would validate the hypothesis generated.

The approach would start with a Cause-and-Effect Relational Matrix shown in Table 2 to narrow down specific root causes showing a high likelihood of the input being a root cause. The matrix is by no means a comprehensive final decision on the root cause given its imperfection but should guide the selection of a root cause.

 *Table 3-2. Cause-and-Effect Relational Matrix*

The results from the Relational Matrix above show each process input impacting the system at a percentage between 4.60% and 6.02%. The variety of the process inputs span a medley of issues within the TBMCS system making it difficult to determine a root cause for improvement. Surprisingly, testable requirements received a low percentage tally because its impact was not widely felt across the whole system, specifically system/subsystem design, and system integration. Modern technology migration’s percentage score of 6.02% scored high due to two high scores in the same areas testable requirements lacked, system/subsystem design and system integration with all three other divisions receiving mediocre ratings around 5. The Cause-and-Effect Relational Matrix demonstrates a plethora of factors contributed to problems in the TBMCS ultimate output.

While the ratings of the importance to the customer may be debated without direct input from the customer, analysis of what the customer wanted from their customer can guide the process for determining the ratings. The government wanted improvements over their legacy system and was not concerned as much with lapses in providing some technical documentation. Requirements definition and management and validation and verification were rated the highest because they both affected the functionality of the end product the most. System/subsystem design received an 8 out of 10 rating because the government wanted their system/subsystems to contain certain features but showed little desire to get involved with the technical design and means for achieving features they desired. System integration received a 7 out of 10 rating because the design would be based off a fully-integrated legacy system so they took for granted the ease with which the new system may be integrated with all of the inputs. The lowest rating, system architecture, received a 6 out of 10 because the government’s involved organizations did not need a complete new architecture to merely upgrade features.

## Identification of Root Causes

From the Cause-and-Effect Relational Matrix, the root cause would appear to be Modern Technology Migration from the System Architecture. The software requested by the government used was antiquated and did not align with the legacy system’s software. Modern Technology Migration by no means was the obvious favorite but the software-intensive nature of the project proves a viable breeding ground for software upgrade problems.

# Improve

## Trade Studies

The selection of software between LM and the government organizations could have been streamlined along with the training. Their selections of software and training methods may have shown strategic improvements with the conduction of proper trade studies before implementation, rather than the government suggesting their choice software. TBMCS incorporated 76 applications, 64 point-to-point external system interfaces, and 413 segments involving over 5,000,000 lines of software with two commercial relational databases highlighting the difficulties for integrating changes in such a software-intensive environment [5]. Programming languages included a wide mix such as Ada, Java 2 Platform Enterprise Edition, C, C++, Perl, Unix Shell Scripts, Standard Query Language (SQL), Ant, Batch Code, Visual Basic, and AWK. Multiple compilers were also used such as Sun C++ compiler, Forte Developer 7 C++ 5.4, Sun C compiler, Forte Developer 7 C 5.4, KL Group XRT Table v2.2.1, Field v1.1.0, Gear widgets, v2.0.1, etc. Operating systems included Solaris 10, Windows 10, Windows Server 2012 (later), and middleware consisting of Oracle 9i/12c, Oracle WebLogic Server 12c, Systematic IRIS, and Java Runtime Environment 1.6 with compatibility to Java 1.6 [4]. Third-party applications demanded high levels of resources due to integration difficulty stemming from fitting into the Defense Information Infrastructure Common Operating Environment (DII COE) or because its Commercial Off-the-Shelf (COTS) infrastructure was more current than that of the TBMCS. This ultimately led to LM reducing applications in functionality or replacing them with other products to achieve integration and operational capabilities. A trade study conducted on the following topic may have saved LM and the government millions of dollars: Software selections, specifically the upgrade to Solaris 8 from 2.5.1. The move was delayed by dependencies on Common Operating Environment (COE) products and by the cost of upgrading COTS products to match new baselines [1].

### Software Selections Trade Study

The TBMCS was able to accomplish monumental software accomplishments including writing software to test access and gain data from Enemy Order of Battle (EOB) services, writing connection software for testing track services, integrating track services into data feeds with TBMCS Common Operational Picture (COP) views, and tests written for TBMCS Track, EOB, and Airspace Service. Strategical advantages gained by this system, however, were offset because of the intimate knowledge needed to operate the system. The system used an outdated version of Java with limited Java Doc and the services were rendered useless without a fundamental understanding of the data stored in Java Doc. In addition, LM was not able to simulate and/or exercise its 64 external interfaces until system tests. Configuration control of the interfaces were constantly going through changes because no formal agreements were in place between LM and the government. With a constantly shifting baseline, interfaces to the applications lacked stability and the applications and interfaces were impossible to lock down. Once TBMCS went into field operations it had to be inoperable with other systems. Many changes were made last minute prior to operational testing, leading to many regressional tests, increased costs, and schedule slips.

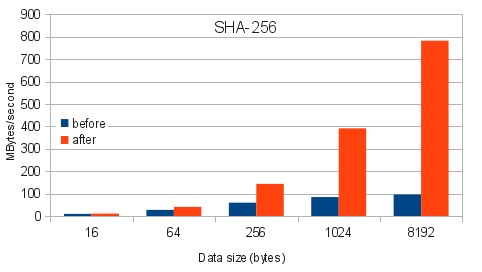
After the early growing pains of the software and other related issues, TBMCS went on to perform capably in Operation Enduring Freedom and Iraqi Freedom and went through four subsequent releases over three years since the release of V1.0.1. Clearly, LM demonstrated its process improvements and systems engineering processes learnt in their attempts to field V1.0.1 could work successfully and repeatedly in a software-intensive environment. In fact, throughout the 1980s and 1990s, large Department of Defense (DoD) projects and commercial acquisitions, not only TBMCS, were frequently overrunning costs and schedule deadlines as the industry played catch-up to larger geographical and cultural program distributions. Three areas were identified for key concept areas representing necessary process and systems management supports: life cycle support, risk management, and system and program management. The early software growing pains could be attributed to TBMCS life cycle support. In the early stages of TBMCS, Solaris 2.5.1 was used and because of multiple conflicting demands, was not timely upgraded to Solaris 8, three releases newer.

In comparison to its newer release, Solaris 8, Solaris 2.5.1 had several flaws. Solaris 2.5.1 included support for the PowerPC platform, however, the port was cancelled before the release of Solaris 2.6 and was never brought back. PowerPC never gathered the lack of application software needed for proper support into the future and the chip was essentially ignored by its customers, including Sun. The buggy launch of PowerPC 620 solidified the instability of PowerPC. While Solaris 2.5.1 merely supported PowerPC, it was utilized by LM and may have affected embedded operations. The general PowerPC platform came to be seen as a hardware-only compromise to run many operating systems one at a time upon a single unifying vendor-neutral hardware platform, the perfect microcosm to represent LMs struggles for a simultaneously flowing architecture [6].

The support dates of Solaris 2.5.1 did not mesh with the future support needed for TBMCS. Although Solaris 8 and 2.5.1 would be both licensed in traditional form before the software became open source, support for 2.5.1 ended in September, 2005 while Solaris 8 received support up until March 2012. The TBMCS failed its first operational test in March, 1999 and suspended its second operational test in January, 2000. Based off these deadlines, LM was on a schedule crunch using Solaris 2.5.1 adding additional schedule and technical progress pressure. Based off releases prior to 2.5.1, the average release support for each version was around 8 years. In January, 2000 after suspending the second operational test, LM knew it most likely had around 3-4 more years of support for 2.5.1 before they would be forced to upgrade for security measures alone, if not technical upgrades [7].

The version upgrades between 2.5.1 and 8 showed significant improvements in Sun’s software that would have benefited LMs integration efforts. Version 5.6 included Kerberos 5, PAM, TrueType fonts, WebNFS, large file support, enhanced prcofs, and dropped SPARCserver 600MP series support. Version 7 saw the release of 64-bit UltraSPARC, added native support for file system meta-data logging, and dropped MCA support on x86 platform. Version 8 included multipath Input/Output (I/O), Solstice DiskSuite, Integrated Pathogen Modeling Program (IPMP), first support for Internet Protocol Version 6 (IPv6) and Internet Protocol Security (IPsec), Modular Debugger, Role-Based Access Control, and removed sun4c support. From the above features, the relevant ones to the TBMCS between 2.5.1 and 8 are the additions of large file support, dropping the SPARCserver 600MP series support to replace with 64-bit UltraSPARC, meta-data logging, Modular Debugger, and Role-Based Access Control. The upgrade to 64-bit UltraSPARC, large file support, and Modular Debugger would improve runtimes between the lagging TBMCS operating systems. Meta-data logging and role-based access control could demonstrate security improvements and version control in the system where collaborative efforts were already causing disorganized updates.

Figure 4-1 presented below shows the performance of Solaris 11 SPARC server, showing the difference between bit allocations.



*Figure 4-1. Solaris 11 SPARC Performance.*

Although Figure 4-1 represents Solaris 11 SPARC performance, careful examination shows the benefit between a 64-bit server and 16-bit server. The difference would be more extreme between the 600 SPARCserver 600 Multilink Point-to-Point Protocol (MP) series and the 64-bit UltraSPARC, possibly recording a 2-3 times improvement between the two. While Sun’s enterprise software was only part of the TBMCS process and would by no means be solely responsible for the slow processing times in the TBMCS, upgrading to Solaris 8 form 2.5.1 would lessen the cause of the Solaris software as a limiting performance factor.

## Solution Feasibility

As mentioned, the move to upgrade to Solaris 8 from Solaris 2.5.1 was delayed by dependencies on COE products and by the cost of upgrading COTS products to match new baselines.

Although moving to one infrastructure and converging applications onto one system is no easy feat and admirable as a long-term goal, removing dependencies on COE products may have freed up Solaris 8 integration efforts. Many of the COE product dependencies were loosely correlated with flexible operational test goals, not requirements. Given the struggles TBMCS encountered fielding V1.0.1, LM could have focused on improving performance metrics of the system as a whole so operational tests could far exceed lofty goals. With the large number of integration nightmares encountered by LM using different software products, LM could have at least demonstrated the excellence and start-of-the-art technology that would be integrated even if operational tests failed. By delaying the upgrade to Solaris 8 and subsequent versions, LM was adding future risk to integration. By introducing Solaris 8 as early as possible in the product life cycle and removing questionable dependencies, LM could check another box off for making the system as future-proof as possible heading into failed operational tests to at least impress government organizations in that capacity. Heading into operational tests, LM was ideally aware of the difficulty they would encounter with integrating different server stacks and separated functions. Upgrading to Solaris 8 from 2.5.1 would help the TBMCS with delivering standardized solutions, transition its systems onto a common framework for the coming years, and utilize embedded subsystems bringing new capabilities forward.

TBMCS baselines constantly evolved up until 2004 when a relative level of stability was finally reached. The cost factor of upgrading COTS may be valid, but relying on the baselines set forth for the TBMCS does not make sense in retrospect. Many of the baselines proposed between LM and the government’s organizations related to legacy system improvements. The upgrade to Solaris 8 would demonstrate LMs efforts to deliver a technologically advanced improved system capable of staying relevant in the AF and other military operations for decades to come. Integrating new technologies into a system is a constant trade-off between expensive up-front costs and long-term use. By upgrading to Solaris 8 at least one year prior to its first operational test in 1999, LM could save money long-term by allowing for earlier adjustment to the added capabilities in Solaris 8. The engineers and technicians need an adjustment period and growth time to understand the new functionalities for any new piece of software or hardware added so delaying the upgrade to Solaris 8 was driving hidden costs further back in the product life cycle to be revealed. While LM may argue they were at the whim of the government’s military bodies on a portion of their software selections, the government did not dictate the particular versions of software they wanted LM to utilize. LM had free reign to prove their system capabilities by upgrading the software. Although upgrading to Solaris 8 represents a small, what was most likely thought insignificant upgrade at the time, a small process adjustment as such could have saved LM significant budget money by decreasing the amount of hours burned adjusting to the upgrade and its embedded workings. Given the amount LM spent fixing early issues in TBMBCS, the upgrade to Solaris 8 from 2.5.1 may have played its part in reducing the amount of bugs and integration struggles encountered while boosting overall system performance.

## Design of Experiments

The initial TBMCS contract conception in 1995 anticipated a $40 million dollar budget divided into progressive increments, over a five-year period. By 1997, LM had already burned through $17 million in reproduction costs alone, foreshadowing spending difficulties in the three years to come through early stage operational tests. Between the hardware and software training infrastructure for servers alone, LM spent nearly $1 million up front. The most expensive software, Plateau Enterprise v3.1 only amounted to $172,750.00 with most software programs costing less than $2,000, Sun products following into this category. Given the overall amount spent on the initial infrastructure, an upgrade to Solaris 8 would cause minimal impact in the year 2000. A common theme between trainees using the system at later dates was that facilitators lacked operational knowledge of the system. By implementing Solaris 8 at its earliest release data, by the time TBMCS would be used by trainees may have at least gained a better grasp of its functionalities in that respect. The delay of implementing Solaris 8 early would play its part in the mass confusion created by incremental software upgrades with products LM personnel were not necessarily familiar with.

Soon after Solaris 8’s release in 2000, LM should have upgraded with budget factors aside. Other software selections and upgrades relied on systems engineering foundations between LM and government. The decision to upgrade to Solaris 8 represented a definitive measure LM could take to produce a better end-product.

## Challenges and Compensation for Implementation

The upgrade to Solaris 8 and later upgrades to Solaris would undoubtedly cause challenges. Costs of product integration were a constant concern with TBMCS, especially given the integration struggles and shifting baselines during a critical testing period. The timing of the release of Solaris 8 would take place during a period where LM was feeling the pressure to deliver a functioning system after suspending operational tests. The government was not aware of the technical sustainability measures needing to be took to improve over the legacy system so at the time, LM would need to assume full responsibility for making sustainability decisions such as upgrading to Solaris 8. LM would feel the pressure from the lack of communication with the government as it tied into COTS decisions. The layered architecture further drove misinterpretation of what the desired software processes/practices would be as it was at such a high level that it resulted in DII COE segmentation processes. By the year 2000 when upgrading to Solaris 8 would be a feasible option, the DII COE was still evolving and the inter-process communication Application Program Interfaces (APIs) was not well defined. LM particularly references that COTS products did not integrate well with an automated database replication scheme and Solaris 8 would not ease this burden. Solaris 8 would be operating under a different set of requirements than some of the system’s other third party applications that would not scale well or be compatible with current versions of the COTS software product baseline, Netscape browser. LM already struggled to standardize its infrastructure from base to base, as well as among different services. Upgrading to Solaris 8 would further this divide if some bases were upgrading to Solaris 8 and some were still operating with Solaris 2.5.1 for various reasons. Overall, LM would benefit from taking its time to evaluate the difficulties they would encounter and produce a fully-functional system even if it meant disappointing its stakeholders with significant product delays.

## Implementation Plans

LM would produce a better TBMCS, earlier on, by upgrading to Solaris 8 from 2.5.1 across all bases in the year 2000 upon its initial release. The amount of coordination required would be significant across all the bases and subsystems.

An earlier upgrade to Solaris 8 would require efforts by both LM and the government. LM would ideally recruit software engineers and systems engineers to evaluate the different functions to be expected out of Solaris 8 before its release. Sun had already released Solaris 7 in November, 1998 giving the engineers a baseline to compare with their current 2.5.1 version. If not incrementally upgrading to Solaris 7 before making the jump to Solaris 8, select engineers could have played around with the upgraded Solaris 7 functionalities to figure out any key differences. From this, they could be more informed about what to expect out of Solaris 8. After this initial research, training sessions or paper training material could be distributed to all engineers so they could familiarize themselves with the upgrade to be expected. After initial training, architects and engineers could use a proactive approach to collaborate and anticipate any integration struggles to come, instead of their evidentiary reactive approach. Any dependencies resulting from the upgrade could be identified in the COE and may even result in earlier mitigations. LM personnel could visit each base to ensure the military personnel would be prepared for the upgrade and to compare hardware/software compatibility for the upgrade. The stakeholders would be required to update the government bodies on their progress to updating throughout the life cycle. The architecture and system requirements/baselines/goals would be updated to reflect this software change along with system performance expected during a vital period for operational tests.

Given the software estimates from the initial infrastructure, a maximum budget would be $200,000 for the software alone. Falling in tune with earlier infrastructure numbers, a total maximum budget somewhere between $500,000 and $1,000,000 would be reasonable to roughly equal the initial infrastructure setup. While this number may seem absurdly high relative to the benefit from upgrading to Solaris 8, given the amount of money LM would spend failing operational tests and configuration management, any minor improvements given from software modifications would be justified within this price range.

Pre-release efforts would start in 1998 upon release of Solaris 7 so by the time Solaris 8 is released in 2000, the upgrade process could be implemented in 6 months to a year, with another 1 year buffer during testing to find and fix unanticipated problems as a result of the upgrade. Audits around 6 months to 1 year after the release of Solaris 8 by LM would ensure compliance with the upgrade system-wide.

At the time, stakeholders did not expect to be involved with problems such as software upgrades. In hindsight, stakeholders could maintain better contact with LM to understand the reasonable budget increases and schedule delays to be expected from the upgrade as well as what they would be getting from the upgrade. Tying into requirements, stakeholders could explicitly state what they hoped to achieve performance and specifications wise at a system and sub-systems level during this time period so they could better understand the reasoning why LM would be upgrading to Solaris 8 during a testing period.

# Control

## Implementing and Monitoring Control Plan

With a foundation for upgrading to Solaris 8 from Solaris 2.5.1 in place, LM would need control measures in place for implementing and monitoring the upgrade.

Solaris 8 should signal an overall minor improvement in the system’s health and performance over 2.5.1. Although this upgrade would constitute such a small part in the software architecture, the amount of bugs in the system should decrease as a result. With simultaneous evolving baselines and general instability in the software, directly correlating bugs produced after implementing Solaris 8 may be difficult, however, forensic analysis of bug causes could reveal the root software problem(s) product(s). The most effective, direct way to measure the enhancements gained by upgrading to Solaris 8 would be analyzing number of bugs and their sources as well as system health/performance.

If Solaris 2.5.1 was ever a limiting factor in system health/performance, measuring effects of upgrading to Solaris 8 could be as simple as measuring how often Solaris 8 was the limiting performance factor and by what deficit compared to Solaris 2.5.1.

Many technical measures could be assessed at a system level, however, demonstrating the measures directly tie in may be difficult. To counteract this, engineers could be surveyed and encouraged to report any issues/bugs they find with the upgraded Solaris 8 software. Engineers working with the software everyday may find issues with the software that support the measures and could give leads on which measures may need further analysis to evaluate the effectiveness of the upgrade. For example, if a software engineer noticed that the distribution of central processing units (CPUs) across the hardware is not the same as before the upgrade, this could prompt further investigation to determine if Solaris is causing issues. The engineers would be more in tune with the correlations in a systems-of-systems context to determine how Solaris 8 affects TBMCS even if there is not measures in place to support their hunches at the time.

From base to base, audits by configuration management personnel could determine the progress of each base’s upgrade to Solaris 8. During this time period, reporting by each base of their experience with the upgrade process to Solaris 8 would be extremely important for better assessment of the upgrade’s impact. If one base is reporting performance issues during the upgrade process but no other base has, the problem may not lie with the software upgrade as much as the software/hardware configuration at that particular base. If multiple bases are reporting slower performance times and common hardware/software integration issues during the upgrade process, this may signal a systematic process issue with the upgrade.

If during the upgrade to Solaris 8 major issues arise that can be directly attributed to the software, the immediate action is to isolate the problem(s) with the software as quick as possible and cut it off from the rest of the system. If the upgrade is affecting every base’s performance, the bases who haven’t upgraded their software to Solaris 8 will continue with Solaris 2.5.1 until the problem(s) is found and fixed. If one base runs into a major problem that is significantly affecting testing as a direct result of Solaris 8, that base will be isolated from the rest of the system until the problem(s) can be found and fixed. Depending on the indicated problem, shutting down the system temporarily may be necessitated. If a bug in the new software upgrade allows for unauthorized personnel to enter into TBMCS this would warrant such measures. Condition indicators and performance indicators should reveal similar to slightly improved metrics after the upgrade. Any measures not supporting this will warrant further investigation.

## Measured Progress and Success

During the implementation of Solaris 8 and months to years afterwards, bugs, performance, and user satisfaction should stabilize to better metrics. While Solaris may not be directly responsible for defects, it contributes to the software environment.

During implementation, the process control may be monitored by checklists and maintenance guidelines. Checklists could address testing each function used in Solaris 2.5.1 to see if those same functions work the same in Solaris 8, documenting any discrepancies so the information may be distributed program-wide. Each step of operation if not already documented from 2.5.1 may be documented for 8. If steps of operation are in place for 2.5.1, the steps may be updated for slight variances with 8. Separate documentation may overview the new information and uses in Solaris 8. Maintenance guidelines could provide useful sources of information for what to do for common problems encountered during the upgrade and troubleshooting for engineers still familiarizing themselves with the update.

LM could sub-contract with technicians working for Sun/Oracle during the initial implementation stages to ensure the upgrade goes smoothly and have subject matter experts on hand to address any shortcomings or troubleshooting. The technicians could give estimations as to what to expect performance-wise from the upgrade and even what health/performance measures LM should look out for to determine the success of the upgrade.

Most handedly, improvements may be measured by employee satisfaction with the upgrade. Employees should enjoy the benefits the most when working with Solaris 8. Employee’s time spent in Solaris 8 could potentially be tracked. This may give indicators of success if employees are spending less time overall using Solaris 8, indicating its advanced features lessened the time needed to use it. Upgrades to software in general should not cause system malfunctions, however, the end-users can experience the upgrades’ functionality and flow their opinions of it out to higher management.

Within 3-6 months after upgrading at each base, LM should have gathered enough data to resolve any conflicts with the upgrade. Any issues with the upgrade would most likely be discovered at that point in time. TBMCS could continue moving forward with its operational tests with LM feeling confident Solaris 8 is not causing system integration issues.

## Organizational Improvements for Success

Building on the future success of upgrading to Solaris 8, TBMCS would benefit from more timely software upgrade practices. Rather than making the jump from Solaris 2.5.1 to Solaris 8, practices could be put in place so TBMCS would be prepared for each Solaris version update as they are released.

During the upgrade process, a minimum of one computer at each base could be running Solaris 8 while not connected to the TBMCS network. By doing so, engineers who would work with the technology would have opportunities to familiarize themselves with the differences before their base of operations received the upgrade embedded in their systems. This could apply with any new software upgrades, programs, and applications introduced throughout the TBMCS life cycle.

After upgrading to Solaris 8, training materials could be put together to highlight the differences in capabilities and uses. Most engineers would be familiar with its 2.5.1 predecessor and would only need to know the fundamental changes in the upgrade.

Better integration and configuration management practices could be put in place. LM did not have their systems engineering practices worked out well with the government which resulted in testing delays, repeated tests, botched tests, etc. LM may have benefited from implementing more isolated tests and having the bugs/defects worked out well with those before advancing to full system operational tests. On a process level, this would result in a quick, spinal, iterative approach that worked later on in the life cycle of the product.

TBMCS struggled in delivering the proper resources to trainees. The trainees often cited lack of operational knowledge by the facilitator as an inhibiting factor. Training facilitators could benefit from active involvement during the testing of the TBMCS to get a more fundamental understanding of the system. More resources could be procured for the facilitators to learn from. The facilitators themselves could go through the training process to figure out where their knowledge or transferring of knowledge may be lacking.

# List of Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| ABP | Air Battle Plan |
| AF | Air Force |
| API | Application Program Interface |
| CEDAC | Cause-and-Effect Diagram with Additions of Cards |
| COE | Common Operating Environment |
| COP | Common Operational Picture |
| COTS | Commercial Off-the-Shelf |
| CPI | Cost Performance Index |
| CPU | Computer Processing Unit |
| C2 | Command and Control |
| DoD | Department of Defense |
| DPMO | Defects (Deficiencies) Per Million Opportunities |
| DPU | Defects (Deficiencies) per Unit |
| DII COE | Defense Information Infrastructure Common Operating Environment |
| EOB | Enemy Order of Battle |
| GR&R | Gage Repeatability and Reproducibility |
| I/O | Input/Output |
| IPMP | Integrated Pathogen Modeling Program |
| IPsec | Internet Protocol Security |
| IPv6 | Internet Protocol Version 6 |
| ISO | International Standards Organization |
| LM | Lockheed Martin |
| LM-ISS | Lockheed Martin Integrated Systems and Solutions |
| PCI | Process Capability Index |
| MOP | Measure of Performance |
| MP | Multilink Point-to-Point Protocol |
| MSA | Measurement System Analysis |
| QMS | Quality Management Systems |
| RTY | Rolled Throughput Yield |
| SE | Systems Engineering |
| SIPOC | Supplies, Inputs, Processes, Outputs, and Customers |
| SLOC | Single Line of Code |
| SPI | Schedule Performance Index |
| SQL | Standard Query Language |
| TBMCS | Theater Battle Management Core System |
| TOC | Table of Contents |
| TPM | Technical Performance Measure |
| TTPs | Tactics, Techniques, and Procedures |
| V1.0.1 | Version 1 Release of TBMCS |

**References**

[1] “Army Forges Ahead with Common Operating Environment for Mission Command,” [www.army.mil](http://www.army.mil). [Online]. Available: <https://www.army.mil/article/195864/army_forges_ahead_with_common_operating_environment_for_mission_command>. [Accessed: 04-May-2020].

[2] J.R. Collens, “Theater Battle Management Core System: Lessons for Systems Engineers,” 2006 IEEE/SMC International Conference on System of Systems Engineering, vol. 22, no. 3, pp. 1-10, Dec. 2007.

[3] J.R. Collens and B. Krause, “Theater Battle Management Core System Systems Engineering Case Study,” Air Force Inst. Of Tech Wright-Patterson AFB OH Center for Systems Engineering, pp. 1-66, Feb. 2005.

[4] J. Taylor, “Graphing Solaris Performance Stats,” Oracle Jeff Taylor’s Blog. [Online]. Available: <https://blogs.oracle.com/taylor22/graphing-solaris-performance-stats>. [Accessed: 04-May-2020].

[5] M.E. Zaharee, “Training Program Review: Theater Battle Management Core Systems (TBMCS) Training Program Evaluation,” National Defense Industrial Association, pp. 1-27, Aug. 2002.

[6] “PowerPC,” Wikipedia, 29-Feb-2020. [Online]. Available: <https://en.wikipedia.org/wiki/PowerPC>. [Accessed: 04-May-2020].

[7] “Solaris (operating system),” Wikipedia, 27-Apr-2020. [Online]. Available: <https://en.wikipedia.org/wiki/Solaris_(operating_system)>. [Accessed: 29-Apr-2020].