# 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. 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. 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 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.

### 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 test, increased costs, and schedule slips.

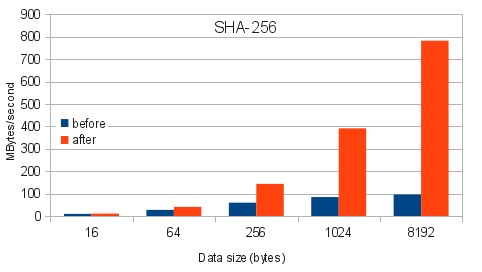
After the early growing pains of the software and other related issues, TBMCS went on to perform capably in Operation Enduring Freedom and Iraqi Freedom and went through four subsequent releases over three years since the release of V1.0.1. Clearly, LM demonstrated its process improvements and systems 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 2.8, 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.

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.

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 I/O, Solstice DiskSuite, IPMP, first support for IPv6 and IPsec, mdb 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, mdb Modular Debugger, and Role-Based Access Control. The upgrade to 64-bit UltraSPARC, large file support, and mdb 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 X presented below shows the performance of Solaris 11 SPARC server, showing the difference between bit allocations.



*Figure X-X. Solaris 11 SPARC 11 Performance.*

Although Figure X 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 600MP 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 Common Operating Environment (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 Air Force 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 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.