Vignette 6 – JMS: Seven Signs That Your Software (Program) Is in Trouble

Richard Murray

The DIB SWAP study visited the JMS (JSpOC [Joint Space Operations Center] Mission System) program in August 2018. The JMS team was open and cooperative, and the people working on the project were highly capable and well-intentioned. At the same time, our assessment of the program was that it was doomed to failure. Because the JMS program was restructured after our visit, we felt it was OK to spell out the problems as examples of what can go wrong.

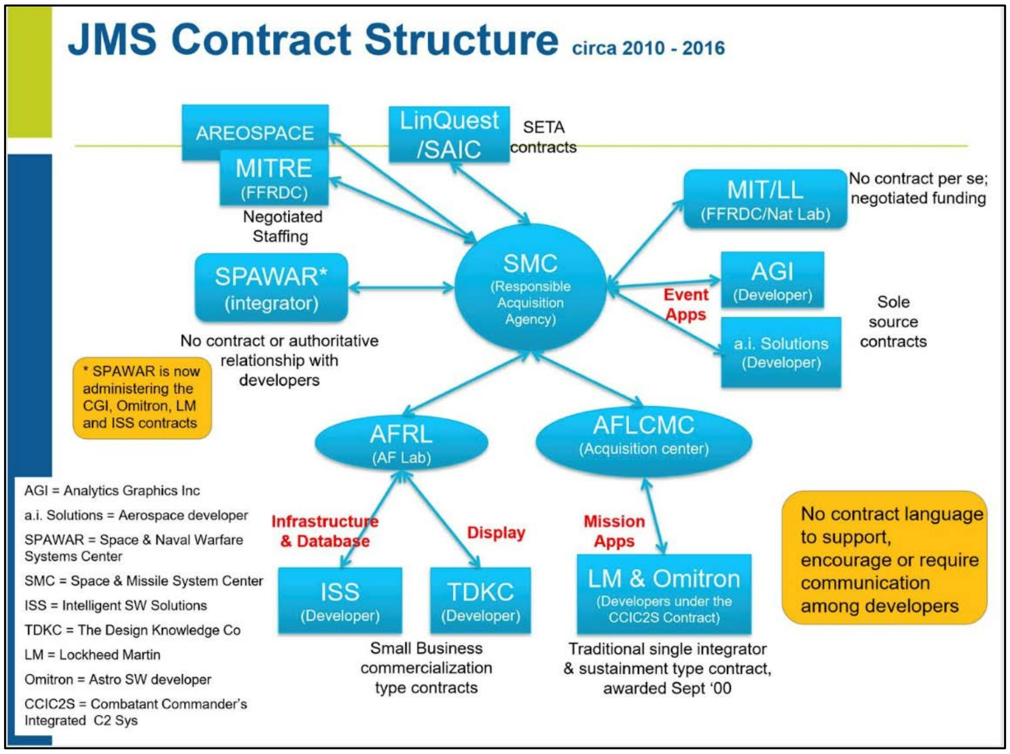
While there were many issues that led to the failure of the JMS program, the following seven are ones that are not a function of that program *per se*, but rather of the process that created it. We thus call these out as general things to look for as indications that your software (program) may be in trouble.

1. The problem is being made harder than it needs to be. JMS increment 2 had a budget of just under $1B. The basic function of the JMS system was to track objects in space. While there are engineering challenges to doing this with the proper precision, the basic problem is *not that hard*. Our sense was that the project could be converted to an "app" within AOC Pathfinder, or something equivalent. Assign 20–30 [50? 100?] programmers (+ 20% program management, administration) to work on it for 3 years at $10–20M/year, with first capability due in 6 months and increments every 2 weeks (based on user feedback). Interface to existing data sources (via software interfaces), run in the cloud, and use a scalable architecture that can get to 1M objects in the next year or two. Make sure that the app architecture can accept a commercial product if one is available that meets the needs of the user (there were some indications this might have already been happening). Target budget: $10–20M/year for first 5 years, $5–15M/year in perpetuity after that.

2. The requirements are outdated. Many of the requirements for JMS increment 2 appeared to trace back to its original inception circa 2000 and/or its restart in 2010. Any software program in which a set of software requirements was established more than 5 years ago should be shut down and restarted with a description of the desired end state (list of features with specifications) and a prioritization of features that should be targeted for simplest usable functionality.

3. The program organizational structure is designed to slow things down. Any software program with more than one layer of indirection between the prime contractor/integrator and the companies doing the engineering work should be shut down and restarted with a set of level-of-effort–style contracts that go directly from the system integrator to the companies delivering code. The system integrator should own the architecture, including the design specifications for the components that plug into that architecture.

4. The program contract structure is designed to slow things down even more. The program had at least a dozen contracts with all sorts of small companies and National Labs. It was apparently treated as a COTS integration problem with lots of pieces, but it was implemented in a way that seemed designed to ensure that nobody could make any progress.


JMS Contract Structure circa 2010 - 2016
AREOSPACE
LinQuest /SAIC
SETA contracts
MITRE (FFRDC)
Negotiated Staffing
MIT/LL (FFRDC/Nat Lab)
No contract per se; negotiated funding
SPAWAR* (integrator)
SMC (Responsible Acquisition Agency)
Event Apps
AGI (Developer)
a.i. Solutions (Developer)
Sole source contracts
No contract or authoritative relationship with developers
* SPAWAR is now administering the CGI, Omitron, LM and ISS contracts
AFRL (AF Lab)
AFLCMC (Acquisition center)
AGI = Analytics Graphics Inc
a.i. Solutions = Aerospace developer
SPAWAR = Space & Naval Warfare Systems Center
SMC = Space & Missile System Center
ISS = Intelligent SW Solutions
TDKC = The Design Knowledge Co
LM = Lockheed Martin
Omitron = Astro SW developer
CCIC2S = Combatant Commander's Integrated C2 Sys
Infrastructure & Database
Display
Mission Apps
ISS (Developer)
TDKC (Developer)
LM & Omitron (Developers under the CCIC2S Contract)
No contract language to support, encourage or require communication among developers
Small Business commercialization type contracts
Traditional single integrator & sustainment type contract, awarded Sept '00

JMS contract structure. [Photo courtesy of former JMS program office]

5. The program is implementing "waterfall with sprints" (otherwise known as Agile BS). The program was implementing "sprints" of ~6–9 months (Agile BS detector alert!). Sprints had hundreds of tasks spread across six development teams. Just coordinating was taking weeks. For a while the program had used 4-week sprints, but infrastructure was not available to support that cadence. Test happened after delivery of software, with very little automation.

6. The program management office is too big and does not know enough about software. We were told there were 200–260 FTEs in the program office. The overall program management should be limited to 10–20% of the size of the program so that resources are focused on the development team (including system architects, user interface designers, programmers, etc.), where the main work gets done. The program office must have expertise in software programs so that it is able to utilize contract and oversight structures that are designed for software (not hardware).

7. OT&E is done as a tailgate process. As an ACAT1 program, JMS was mandated to conduct operational test, a process that nominally required the program to freeze its baseline, do the tests, and then wait 120 days for report. The Operational User Evaluation conducted in early 2018 was terminated early by the Air Force due to poor performance of the system. The OT&E process being used by the program added information to support the termination decision, but it is important to note that had the program not been terminated the tailgate nature of the evaluation was one that would have added further delays.

The JMS program has since undergone major changes to address the issues above, so the criticisms here should be taken as an example of some of the signs that a program is in trouble.