UNDER SECRETARY OF THE AIR FORCE
WASHINGTON

SEP 20 2004

04A-003

MEMORANDUM FOR SEE DISTRIBUTION

SUBJECT: Revitalizing the Software Aspects of Systems Engineering

REFERENCE: Air Force Software-Intensive Systems Strategic Improvement Program (AFSSIP) memo dated 13 Jan 2004.

In multiple programs across our acquisition communities, we have recognized systems engineering challenges over the past few years, and have taken steps to improve the implementation and effectiveness of our systems engineering processes.

This policy memorandum is intended to improve the efficiency and effectiveness of our acquisition processes and software management. These processes are applied as an integral part of our systems engineering and capability acquisition processes. To support our overall agile acquisition objectives, we expect you to address, as a minimum, the following software focus areas throughout the life cycle of your acquisition programs beginning with pre-Milestone/Key Decision Point A activities:

1. **High Confidence Estimates:** Estimate the software development and integration effort (staff hours), cost, and schedule at high (80-90%) confidence.

2. **Realistic Program Baselines:** Ensure cost, schedule, and performance baselines are realistic and compatible. Ensure the baselines support the disciplined application of mature systems/software engineering processes, and ensure software-related expectations are managed in accordance with the overall program's expectation management agreement. The program budget must support the high confidence estimates for effort (staff hours), cost, and schedule.

3. **Risk Management:** Continuously identify and manage risks specific to computer systems and software as an integral part of the program risk management process. Ensure the risks, impact, and mitigation plans are appropriately addressed during program and portfolio reviews.

4. **Capable Developer:** Identify the software-related strengths, weaknesses, and risks; domain experience; process capability; development capacity; and past performance for all developer team members with significant software development responsibilities. Consider this information when establishing program baselines and awarding contracts, and throughout program execution.

5. **Developer Processes:** Ensure the entire developer team establishes, effectively manages, and commits to consistent application of effective software development processes across the program.

6. **Program Office Processes:** Ensure the program office establishes and employs effective acquisition processes for software, is adequately staffed, and consistently supports the developer team in the disciplined application of established development processes.

7. **Earned Value Management Applied to Software:** Continuously collect and analyze earned value management data at the software level to provide objective measures of software cost and schedule. The Earned Value Management System should support and be consistent with the software effort and schedule metrics.

8. **Metrics:** Employ a core set of basic software metrics to manage the software development for all developer team members with significant software development/integration responsibilities. Guidance for the core metrics is provided in the enclosure. Programs are encouraged to implement additional metrics based on program needs.

9. **Life Cycle Support:** Address sustainment capability and capacity needs during the system design and development phase, and balance overall system acquisition and sustainment costs. Ensure you plan, develop, and maintain responsive life cycle software support capabilities and viable support options.

10. **Lessons Learned:** Support the transfer of lessons learned to future programs by providing feedback to center level Acquisition Center of Excellence (ACE) and other affected organizations. Lessons learned information includes original estimates and delivered actuals for software size, effort, and schedule; program risks and mitigation approaches; and objective descriptions of factors such as added functional requirements, schedule perturbations, or other program events that contributed to successes and challenges.

These focus areas will be incorporated as appropriate in your Systems Engineering Plan, Integrated Program Summary, or acquisition plans. We also expect you to address these focus areas as applicable during Acquisition Strategy Panels and PEO portfolio reviews. PEOs may tailor the implementation of these focus areas as required and the appropriate Acquisition Executive will be notified of all tailoring.

Sample language and additional guidance will be available in November 2004 in an Air Force Software Guidebook. Our POCs are Mr. Ernesto Gonzalez, SAF/AQRE, 703-588-7846, Ernesto.Gonzalez@pentagon.af.mil, and Maj Mark Davis, SAF/USAL, 703-588-7385, Mark.Davis2@pentagon.af.mil.

MARVIN R. SAMBUR
Assistant Secretary of the Air Force
(Acquisition)

PETER B. TEETS
Undersecretary of the Air Force

Attachment:
Guidance for Core Software Management Metrics

REVITALIZING THE SOFTWARE ASPECTS OF SYSTEMS ENGINEERING

DISTRIBUTION LIST

AFPEO/AC
AFPEO/C2&CS
AFPEO/WP
AFPEO/F/A-22
AFPEO/CM
AFPEO/SP
SAF/AQX
SAF/AQC
SAF/AQI
SAF/AQL
SAF/AQP
SAF/AQQ
SAF/USA

cc:
HQ AFMC/CC
HQ AFSPC/CC
AF-CIO
AFPEO/JSF
SAF/ACE
SAF/AQR
SAF/ILC
HQ SSG/ED
HQ MSG/CD
AFOTEC

Enclosure - Guidance for Core Software Management Metrics

Program offices and developers should mutually agree on and implement selected software metrics to provide management visibility into the software development process. The metrics should clearly portray variances between planned and actual performance, should provide early detection or prediction of situations that require management attention, and should support the assessment of the impact of proposed changes on the program. The following core metrics are required:

- Software Size
- Software Development Effort
- Software Development Schedule
- Software Defects
- Software Requirements Definition and Stability
- Software Development Staffing
- Software Progress (Design, Coding, and Testing)
- Computer Resources Utilization

These indicators should be tailored and implemented consistent with the developer's internal tools and processes. Program offices and developers should agree upon and establish additional metrics or means of insight to address software issues deemed critical or unique to the program. All software metrics information should be available to the program office, ideally through on-line, electronic means. Additional information is provided below for each required metric.

Software Size

The size of the software to be developed/integrated is the most critical factor in estimating the software development effort and schedule. Software size should be estimated and recorded prior to the start of the program, and tracked until the completion of development by all programs involving software development or sustainment. Software size should be estimated and tracked at least to the function or Computer Software Configuration Item (CSCI) level for each spiral, increment, or block. It should be re-evaluated at major program milestones or whenever requirements are changed. The actual size should be recorded at the time a capability (spiral, increment, or block) is delivered. The reasons for changes in software size should also be captured over the development period.

Software size is typically measured in source lines of code (SLOC). For weapon system software development, SLOC is likely the most readily available and the best understood measure. Size should be tracked for new, modified, and reused code. For programs where relatively small changes are being applied to large existing software products, or for development efforts that primarily involve the integration of existing software products, some type of "equivalent lines of code" or some other measure may be appropriate to identify and track the volume of effort required. Whatever measure is used must be clearly defined such that it is easily understandable and can be consistently applied.

Changes in software size may indicate an unrealistic original estimate; instability in requirements, design, or coding; or lack of understanding of requirements. Any of these situations can lead to increases in the cost and schedule required to complete the software. Variations in software size when tracked by spiral, increment, or block, may indicate migration of capability from earlier to later increments. Software size data collected over time will provide a historical basis for improved software estimating processes.

Software Development Effort

Software development effort is measured in staff hours or staff months, and directly relates to software development cost. Estimated software development effort is derived primarily from software size, but also depends on other factors such as developer team capability, tool capability, requirements stability, complexity, and required reliability.

When combined with earned value data and other management information, variances in planned and actual effort expended may indicate potential overruns, lack of adequate staff or the proper mix of skills, underestimated software size, unstable or misunderstood requirements, failure to achieve planned reuse, or unplanned rework as a result of software defects.

Software Development Schedule

Software schedules should be planned to at least the function or CSCI level for each spiral, increment, or block, and should be re-evaluated at major program milestones or whenever requirements are changed. Planned and actual schedules should be tracked continuously from the start through the completion of development. Software schedules should provide insight into the start and completion dates as well as progress on detailed activities associated with requirements, design, coding, integration, testing, and delivery of software products.

Software development schedule durations are measured in months. Like effort, estimated software development schedules are determined primarily from software size, but also depend on other factors such as developer team capability, tool capability, requirements stability, complexity, required reliability, and software testing methods and tools.

Late or poor quality deliveries of low level software products are indicators of overall program schedule risk. Schedules should be examined for excessive parallel activities that are not realistic when available resources such as staff or integration labs are considered, excessive overlap of activities where dependencies exist, or inconsistent detail or duration for similar tasks.

Software Defects

Software defects should be tracked by individual software products as part of the system defect tracking process from the time the products are baselined. Software defects should be tracked at the function or CSCI level or lower, by spiral, increment, or block.

Defects are measured by tracking problem reports. Problem reports should account for missing or poorly defined requirements that result in software rework or unplanned effort. Problem reports may be tracked by category or criticality, including total number of problem reports written, open, closed, etc. These could be further broken down by additional categories, including development phase (requirements definition and analysis, design, code, developer test, and system test) in which the problem was inserted, development phase in which the problem was discovered, or by severity.

Software defect metrics provide insight into the readiness of the software to proceed to the next phase, its fitness for intended use, and the likelihood/level of future rework. Analysis of software defects may also indicate weaknesses in parts of the development process, or may identify certain software components that are particularly troublesome and thus contribute greater program risk.

Software Requirements Definition and Stability

The number of software requirements should be tracked by spiral, increment, or block, over time. The number of changes to software requirements (additions, deletions, or modifications) should be tracked in the same manner. The reasons for requirements changes (new capability or improved understanding derived during development) should also be tracked.

The number of requirements relates to software size and provides an indicator of how the requirements are maturing and stabilizing. Software requirements changes can be an early indicator of rework or unplanned additional software development effort.

Software Development Staffing

Software staffing is tracked using two separate measures. The first tracks the status of the developer's actual staffing level versus the planned staffing profile over time. A separate measure tracks developer turnover (unplanned losses of development personnel that must be replaced). Staffing can also be tracked by personnel type, such as management, engineering, qualification/testing, and quality assurance.

It is common for developers to plan for a rapid buildup of developers at the start of a program, and it is also common for programs to have difficulty ramping up their staff at the planned rate. Late arrival of staff indicates planned up-front work is not being completed on schedule, and will likely lead to delays in delivery, reduced functionality, or both. Turnover adversely impacts productivity through the direct loss of developers, replacement staff learning curve, and the impact on existing staff to support replacement staff familiarization.

Software Progress (Design, Coding, and Testing)

Software progress is used to track over time, down to the lowest level of software components, the actual completion of development phase activities compared to the program plan. A typical approach to progress tracking involves measuring the actual number of software components or units designed, coded, or tested compared to the planned rate of completion.

Failure to complete these lower level development activities according to schedule is an indication that there will likely be impact to program-level schedules.

Computer Resources Utilization

Computer resources utilization is a measure of the percentage of computing resources consumed by the planned or actual software operating in a worst case processing load. Engineering analysis is required to define realistic worst case scenarios the system is expected to encounter. Utilization is measured as a percentage of capacity used for processing, memory, input/output, and communication links. This measure should be tracked as an estimate in the early phases of system development, and actuals as the system continues through development/integration. Monitoring computer resources utilization helps ensure the planned software design and expected capabilities will fit within the planned computer resources, and that adequate reserve capacity is available to permit some level of enhancement in the post deployment support phase. Overloaded computer resources can lead to system instability or other unacceptable performance.