Technical Report

CMU/SEI-89-TR-006
ESD-89-TR-006

Proceedings of the Workshop on Executive Software Issues August 2-3 and November 18, 1988

Edited by:

Richard Martin
Scott Carey
Mark Coticchia
Priscilla Fowler
John Maher, Jr.

January 1989

Technical Report
CMU/SEI-89-TR-006
ESD-89-TR-006
January 1989

Proceedings of the Workshop on Executive Software Issues August 2-3 and November 18, 1988

Edited by:
Richard Martin
Scott Carey
Mark Coticchia
Priscilla Fowler
John Maher, Jr.
Technology Transition Program

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Workshop Participants

Mr. Walter Attridge*

Mr. Jay Beacon

Lt. Col. Thomas Bolick

Mr. Stan Brown#

Ms. Lou Burns

Dr. Larry Caplan

Capt. James Cardow

Mr. Clyde Chittister

Mr. Kevin Deasy, Esq.*

Dr. Richard Fairley

Dr. Norman Gibbs

Mr. Larry Gresham#

Mr. Al Hawkins

Mr. Charles Hill

Ms. Rita Hillyer#

Mr. William Hodges*

Mr. Watts Humphrey

Mr. Fred Joh

Lt. Col. Frank Kirschner*

Ms. Anne Martin, Esq.

Dr. Cecil Martin

Mr. Richard Martin

Mr. Don O'Conner*

Mr. Martin Owens*

Dr. William Riddle*

Dr. Win Royce

Dr. William Scherlis*

Mr. William Sweet

Mr. Al Whittaker

Lt. Col. Mike Wiedemer*

Dr. Martin Wolf*

Mr. Fred Yonda*

* - attended only the 2-3 August 1988 session
# - attended only the 18 November 1988 session

Preface

This report is divided into four chapters. The first chapter describes the organization of the workshop itself. The next three chapters describe the results of the workshop by subgroup topics (national strategy, acquisition, and building large complex systems).

Proceedings of the Workshop on Executive Software Issues

Abstract: These proceedings document the results of two sessions of the Workshop on Executive Software Issues held at the Software Engineering Institute on August 2-3 and November 18, 1988. The purpose of the workshop was to define the issues that should be brought to the attention of executives in the defense industry, government, and academia and to propose resolutions to those issues. The issues discussed were divided into three broad categories: national strategy, acquisition, and building large complex systems. Described in the proceedings are the major issues and recommended actions.

1. Workshop Organization

1.1. Purpose

These proceedings describe the results of two sessions of the Workshop on Executive Software Issues held at the Software Engineering Institute (SEI) in Pittsburgh, PA, August 2-3 and November 18, 1988. The purpose of the workshop was to develop a set of software issues and recommended actions of which government, industry, and academic executives need to be aware in order to help address the current software challenges.

1.2. Selection of Participants

In order to develop a consensus as to what the current mission critical computer resource (MCCR) software issues are and their potential solutions, representatives from industry, government, and academia who are knowledgeable about MCCR software issues and who have credibility in the software community were invited.

The participants were selected by the director of the SEI Technology Transition Program, Richard Martin. Several of the participants had attended earlier Air Force Contract Management Division (AFCMD) steering group meetings concerned with software issues. Others were selected because of earlier work they had done with the SEI, their association with the AFCMD Joint Executive Conference (JEC), or their association with the Aerospace Industrial Association (AIA). The participants are listed on page one and in Appendix A.

1.3. Initial List of Issues

At the opening of the first session of the workshop, an initial list of issues was presented. This list was generated from issues developed at an AFCMD JEC software issues steering group meeting, plus issues the SEI had generated for its own use. The SEI organized the issues into three categories (national strategy, acquisition, and building large complex systems) for presentation at the opening of the workshop.

1.4. Workshop Agenda

The first session of the workshop was held for two days. On the morning of the first day, the participants met in a group session and Richard Martin reviewed the goals of the workshop and the purposes of the workshop's results. Scott Carey of the Technology Transition program staff then presented the list of initial issues. There was some discussion of the issues; however, most of the discussion was reserved for subgroup sessions.

The workshop then was divided into three subgroups, one for each major category (national strategy, acquisition, and building large complex systems). The participants were assigned to a subgroup and a spokesperson appointed. The SEI provided an assistant for each subgroup to help record decisions on issues and solutions. The subgroups were tasked to come up with a refined list of issues to be presented to the entire group after lunch on the second day.

The subgroups worked independently for the next afternoon and morning. There was some shifting of issues between subgroups, but the majority of issues were addressed by the original subgroups.

After lunch on the second day, the entire group reconvened and each subgroup presented its findings. There was some discussion of the issues at this time, yet no major changes were made.

An initial draft of the proceedings from the first session was written by SEI staff members and mailed in October to participants for comments. Comments were received and incorporated, and an updated draft was mailed to the participants the week before the second session of the workshop.

The second session of the workshop was held in one day. Richard Martin reviewed for the participants the goals of the workshop and the prior workshop's results. The participants then went into their original subgroups to consider outstanding comments against the first draft of the proceedings and to develop implementation actions for the recommendations. The group reconvened and each subgroup presented its findings at the end of the day.

A final draft of the proceedings was mailed to the participants for comment in mid-December. This final proceedings incorporate the comments received against that final draft.

2. National Strategy

The working group assigned to discuss software issues related to national strategy identified three categories of issues that are critical to the future of software engineering on a national scale: assuring executive awareness of software issues; national positioning of government, industry, and academia for the future in the software world; and education and training of current and future software practitioners. In this chapter, each of these three categories of issues is discussed.

2.1. Assuring Executive Awareness of Software Issues

Issues:

- Currently, most senior managers have hardware backgrounds and have neither a strong interest in software nor a good understanding of the issues. Hardware development has traditionally been emphasized for the country's technological and economic growth. Software development is now also of critical importance and requires more emphasis.

- The capital investment by industry and government in software development capability is less than that invested in hardware development capability. Senior management generally is not aware of this imbalance nor of the importance of investing in a modern software development capability. Costs include hardware and software, training, and maintenance, as well as the cost of implementing a software development process to use the technology effectively.

- The modular systems development concept is challenging the current practice of having software application domain expertise in subsystem houses (e.g. radar, etc.), and developing both hardware and software there. Separating software development expertise from the subsystem domain expertise may have major effects on the aerospace industry that neither government nor industry has fully appreciated. (See Appendix B for a more complete description of this issue.)

- The effect of modern software development practices on organizations is not well understood. Potential impacts include the way requirements are developed, reporting strategies, government policies and standards, capitalization, and determination of the software skills required and the identification of personnel who can successfully perform the work.

Needs:

Senior management tends to pay attention to important business issues, rather than the technical details involved in developing products. Therefore, ways to help increase management awareness of the software issues in industry are to relate these issues to their impacts on return on investment (ROI), corporate image, product quality, and competitiveness. For senior management in government, awareness can be increased by identifying the threats posed by lack of system performance and product quality and the cost and schedule overruns associated with software development. Proper emphasis must be placed on software-related capital investment.

Recommendations:

- Develop a top-level briefing directed at government and industry executives. The briefing should highlight examples of the impact of software on ROI, image, competition, system performance and quality, cost, and schedule. The briefing should include both good and bad aspects of modern software engineering and its organizational effects, as well as the capital investment required to develop a modern software development capability.

- Inform executives about the flexibility and functionality that software can add to systems.

- Define the implications of the modular systems development concept, and inform executives of its ramifications.

Implementation Actions:

The SEI should:

- Abstract the issues discussed in these proceedings to a few key issues.

- Develop a strawman vision of the nation's direction in software capability.

- Develop a strawman national strategy that includes identifying complementary roles of industry, government, and academia for implementing the vision, and for attracting more people into the software field.

- Develop a top-level briefing of the above actions.

- Develop a top-level briefing directed at government and industry executives highlighting examples of the impact of software on ROI, image, competition, system performance and quality, cost, and schedule.

2.2. National Positioning of Government, Industry, and Academia for the Future in the Software World

Issues:

- There is a lack of vision and focus in the U.S. for promoting a national strategy that continues our leadership in software. As more countries develop technological expertise, the U. S. runs an increasing risk of falling behind.

- The U.S. software technology lead is being eroded by the current practice of foreign military sales offsets.

- There is a need for better definition of the roles of industry, government, and academia for transitioning technology. While limited operational models exist, there is no comprehensive model defining responsibilities on a national level.

- A mismatch exists between research topics in academia and the requirements from practice in industry and government. The research community does not have the proper incentives to meet the needs of industry and government, and little guidance is provided to narrow that gap.

- While the need for software engineering professionals is growing, the total number of young people entering the field is declining. This trend must be reversed in order to meet the software engineering needs of the future.

Needs:

The U.S. needs an overall view of where the nation is headed with regard to maintaining preeminence in software engineering. It also needs to provide the leadership to develop and implement a coherent strategy. Currently, a host of issues influence the direction the software world is taking. The effects of standardization, reuse of software components, capital-intensive software development environments, increasing foreign competition, and other similar developments are not entirely predictable or coordinated. For example, while the government desires reusable software, the procurement system doesn’t reward its development.

As software engineering grows as a discipline, conventions must be developed that are understood by all practitioners. The development of large complex systems presents particular difficulties for projects that require hundreds of programmer years to develop. Techniques and approaches from other disciplines or new research initiatives may be necessary to provide adequate processes, methods, and tools. Also, in order to provide good indices for measuring the way large complex systems are developed, meaningful progress measures must be developed for the software engineering discipline. As yet, there is no national effort to consolidate and coordinate our efforts to lead in the software field as the SEI is doing for the DoD.

Recommendations:

- Develop a vision of the nation’s direction in software capability.

- Identify a champion (national level commission, or person) for software to provide leadership in developing and implementing a national strategy for software capability.

- Define and articulate the complementary roles of industry, government, and academia in technology transition that will result in leading-edge collaboration.

- Develop a national strategy to attract more people into the software field.

Implementation Actions:

See Section 2.1, Assuring Executive Awareness of Software Issues.

2.3. Education and Training

Issues:

- Not enough students are trained in standard software engineering.
- Too few students are currently educated in software systems engineering for later development into people who can look at a whole system to determine the hardware and software trade-offs. Students do not get sufficient exposure to computer hardware engineering, hardware-software systems concepts and integration, government systems requirements, and the relationships between software systems development and maintenance activities.
- It is not enough to train people in "software engineering." Application domain knowledge is as important as software engineering knowledge.
- The responsibility boundary for software engineering education is not well defined between industry and universities. Industry must articulate its position and needs, while universities must move to produce students equipped to address real world problems. Industry, government, and academia must work together creatively to improve software engineering education.
- There is a classification for "computer scientist" in government, but there is none for "software engineer." Nor is there a career path for software engineers in government.
- Continuing education and training of current employees must be provided to prepare them for advances in software engineering practice. It is difficult to convince management to spend money for training. Training dollars typically come from discretionary funds, rather than from a budget line-item, and training decisions often are based on dollars and short-term priorities. Further, government doesn’t support degree-related education for civilians, which inhibits employees from fully advancing in the software disciplines.
- Students are not prepared to make the jump between computer science or software engineering in school to software engineering in the professional world.
- Government employees who manage software contracts need more software engineering experience to better understand their jobs. Some combination of industrial experience for government personnel on a broader basis than is currently available needs to be explored.

Recommendations:

- Identify the responsibilities of industry, government, and academia in providing appropriate education for the software industry.
- Develop a briefing and perhaps a videotape for high school seniors, college students, and counselors aimed at career opportunities in the software engineering profession.
- Create awareness in senior managers of the benefits for retraining and continuing the education of employees for software engineering.
- Develop a draft program for industry, government, and academia for on-the-job training cooperation.
- Develop five-year cooperative education curricula in software engineering.

Implementation Actions:

See Section 2.1, Assuring Executive Awareness of Software Issues.

3. Acquisition

The working group assigned to discuss acquisition defined five categories of related issues: new acquisition practices, government/contractor interfaces, reusable software, contracting methods, and data rights.

Acquisition policies and activities directly affect the way software is developed and supported for the government. In addition, the approach to acquisition determines the extent to which new technologies, such as reuse and rapid prototyping, are used. The cost, functionality, and quality of the software depends on the nature of these policies and how they are implemented.

3.1. New Acquisition Practices

Issue:

- Current acquisition practices do not adequately stimulate contracting innovations to acquire timely, cost-effective, and functional software products and processes.
- Acquisition practices need to include how to cost software for the entire life cycle.
- The process of tailoring and using DOD-STD-2167A is not well understood.

Needs:

- Inclusion of a Systems Operational Requirements Document (SORD) in Requests For Proposals (RFPs) should be explored.
- Innovative tailoring of DOD-STD-2167A and DOD-STD-2168 needs to be modeled and encouraged.
- The idea of software warranties as part of system warranties needs to be explored because of the congressional pressure and mandate to do so.
- Most software cost models require lines of code as a critical input parameter, but there is no generally accepted definition of what constitutes a line of code. In order to develop meaningful and credible software schedule and cost estimates, we need a definition for line of code and what that measure means in different parts of the life cycle.

Recommendations:

- Develop example Statement of Work (SOW) clauses to support acquisition personnel in using new technologies. Example clauses should be developed, at a minimum, for the following areas: software quality, reuse, product/process metrics, requirements certification vehicles (prototypes, models), competitive development of requirement specifications, and warranties. The clauses must be developed and tried on programs prior to providing them for general use.
- Building on already established references, develop a set of common, accepted definitions of software engineering terminology, including terms such as "software reliability," "software quality," and "advanced development models", to be used to guarantee consistency in proposal preparation and in the construction and support of software.

Implementation Actions:

- Develop a consistent set of regulations and standards that integrate software into the system engineering process.
    - Start with a base of definitions, then develop software terms that correspond to hardware terms (e.g., hot mockup).
    - Update all the regulations and standards to more consistently handle software as part of the system engineering process (DFAR, MIL-Q, MIL-STD-109, MIL- STD-499, MIL-STD-1815A, MIL-STD-1521B, DOD-STD-2167A, DOD-STD-2168, etc.).
- Task a standards organization to develop a line of code definition.
- Ensure that DOD-HDBK-287 (for DOD-STD-2167A) and DOD-HDBK-286 (for DOD-STD-2168) get a full industry review.

3.2. Government/Contractor Interfaces

Issues:

- The state of the art and practice in software measurement is very immature.
- Questions exist about the effective use of new government initiatives, government oversight, management indicators of progress and quality, and the Software Development Plan (SDP).
- The value added by government in creating and supporting software systems is not recognized by government contractors.

Recommendations:

- Government innovations such as the Software Contractor Assessment Instrument (SCAI), the Software Engineering Evaluation (SEE), and the Software Assessment (SWA) should be incorporated into the acquisition process. Corporate executives need to be informed of these assessments.
- The government oversight role needs to be more clearly described.
- Better criteria are needed for selecting management indicators. Current indicators often measure what is easy rather than what is important. Contractors should therefore be encouraged to propose and use indicators with which they have had a history of experience and success. Also, the Software Development Plan is an appropriate vehicle for describing the use of management indicators for a particular development or support effort.
- The Software Development Plan must identify relevant methodologies and tools. It must be a "living" document, updated in a timely manner.
- The way the government plays its role must be improved. A teamwork relationship between the government and a contractor should be encouraged. Government needs to justify data and review requirements and schedule. Event-driven, as opposed to schedule-driven, reviews are preferred, to assure that reviews of meaningful material are held at an appropriate point in time. The government needs to provide improved guidance to contractors for preparing cited deliverables, so that what is expected is known in advance. Government program office personnel need more software acquisition education and training. The User's Operational Concept (AFR 57-1) is an important input to the requirements process and should be used more effectively.

Implementation Actions:

- Continue AFSC/PLR efforts to publish a pamphlet on software engineering capability assessment.
- Identify to the JEC, AIA and academia as well as other appropriate organizations, the recommendations above. Inform executives that the software engineering capability/capacity of their software organizations will be evaluated as part of the source selection process.
- Coordinate and focus work to develop software management indicators.
- Investigate how to implement teamwork relationships between government and contractors.

3.3. Reusable Software

Software reuse is seen by many as a way to reduce risk and cost in the acquisition of software systems.

Issues:

- There are impediments to the reuse of software components that are directly related to the existing acquisition practices and procedures. Software is currently covered under copyright laws, and considerable attention is applied to protect the copyrights of the software developers. At the same time, contractors and government contract managers are reluctant to trust existing software for projects. Additionally, acquisition policies and procedures do not address the impacts of reuse or make allowances for including reuse in contracts.
- Past and current software development techniques are centered on development of software components to solve a specific problem, within the constraints of a particular project. The result is continuous "re-invention of the wheel", with all the expected expense and effort of duplicating the process.
- Before being able to put a reusable software library into place, it is essential to define the policies and practices that will govern the library. Currently there are not sufficient criteria established to define what components belong in the library, what management policies guide the selection of components, the qualification of components, or the use of components.

Needs:

- Need to resolve confusion with the copyright of software and software products while optimizing the potential for reuse. All parties, software developers, contractors, and contractor monitors, need incentives to work with "reusable" software. There also must be protection for all parties involved.
- Need to resolve the technical issues inhibiting software reuse and develop the standards and the procedures for reusability.
- Need to establish the framework from which a software library system can evolve.

Implementation Actions:

- Develop software data rights guidelines to provide a better understanding of data rights as applied to software and documentation within a software reuse library system.
- Develop policy on the responsibility covering the software and documentation within a software reuse library system. Specifically address the issues of Government Furnished Software (GFS) as it applies to use in development contracts.
- Explore policy for a software reuse library system on software warranties.
- Examine existing acquisition standards (DOD-STD-2167A and DOD-STD-2168) to identify procedures that could create barriers to reuse.
- Develop a software coding standard for components in the library.
- Develop metrics for measuring "design for reusability." This metric will enable a potential user to gauge the portability of the components and will provide some quality metrics when comparing similar components.
- Develop documentation standards for reuse library components.
- Develop minimal testing standards to be applied to components prior to including them into a software reuse library.
- Develop standards for documenting the testing (procedures, inputs, results) of components in a software reuse library.
- Develop the procedures (methods) for analyzing a specific domain to assess areas of high pay-offs in reuse.
- Develop a cataloging methodology to allow numbering of software and documentation within a software reuse system.
- Develop a retrieval system for a software reuse library that allows fast, accurate definition of the required component in a user-friendly mode. (The retrieval system may be best approached as an artificial intelligence expert system.)
- Determine the best approach to the creation of a library system for weapon system software.
- Determine policy for certification (i.e. software with a trust level of one is highly reliable, level two should be used in all but life-critical applications, etc).
- Determine policy on allowing access to a software reuse library.
- Determine policy for handling classified information within the software reuse library.
- Determine policy for changes to the software in the software reuse library.
- Determine policy of rewards for programs submitting and using software in the software reuse library.

3.4. Contracting Methods

Issue:

Do government contracting methods adequately accommodate realistic schedules, government oversight, and fixed-price versus cost-plus contracts?

Needs:

There is a need for realistic industry critiques of a Request for Proposal (RFP) without the threat of non-compliance, as well as realistic milestones for System Requirements Review (SRR), System Design Review (SDR), etc., leading to Initial Operating Capability (IOC).

Recommendations:

- Provide a climate in acquisitions which will promote greater interaction and feedback on software sections of a system.
- Provide government oversight as a costed and scheduled item.
- Use more competitive cost reimbursement contracts through the allocated baseline (requirement specification) portion of the effort; this method, which allows more competition, is used in Europe. Use fixed-price contracts once an allocated baseline is determined. This may only work for programs that are well precedented.

Implementation Actions:

Apply the appropriate contracting vehicle to the appropriate portion of the life cycle. This is an institutional problem which encompasses more than just software.

3.5. Software Data Rights

Issues:

There is a perceived problem with software data rights regulations.

- The Department of Defense's acquisition of modern software technologies is being inhibited in some cases by interpretations of the software data rights regulation. It is possible that good contracting practice can handle most presumed problems, but those practices are not always being used.

- Commercial, off-the-shelf (COTS) software is being used commonly now by the government. Industry is concerned about data rights issues in the use of COTS, as well as software developed for reuse.

Needs:

- Need for changes to the software data rights regulation.

- Need to encourage flexibility in the software acquisition process.

- Need to deal with data rights early in procurement. There is a need to get the right people involved earlier in the process.

Recommendation:

- Identify inhibitors in the current acquisition practices for new technologies.

- Develop model contract clauses which span the extremes of rights in technical data.

Implementation Actions:

- Pass the information about these issues to the DAR council.

- Involve executive management with these issues. Make this a JEC and AIA topic.

- Determine a method for reaching university executive management with these issues. The goal would be to institutionalize data rights issues into software engineering and law program curricula.

4. Large Complex Systems

The working group for large complex systems identified four categories of issues which are essential to the future of software engineering and its related issues: software process management, requirements, software development investment, and training.

4.1. Software Process Management

Software process management encompasses concerns in the areas of systems development strategy, executive management direction, quantitative software management, and software reuse.

4.1.1. Systems Development Strategy

Issue:

The growing complexity and dynamics of software systems have outgrown the traditional software development process. Principle issues are requirements definition and change, system complexity, intolerance of latent errors, and changing technology.

Recommendations:

- To address these problems, evolutionary development methods should be used on all large-scale, complex, critical software systems. These methods should take into account user involvement and total systems implications and should include prototyping and modeling.
- Change the procurement policies, as necessary, to allow the use of evolutionary development methods for resolution of critical issues.
- Place greater emphasis and investment on higher quality software products.

Implementation Actions:

- Write RFPs that require evolutionary development methodologies.
- Support research in provability of software correctness. Continue formal testing after delivery. Create a nation-wide repository of software error sources and changes.
- See Section 2.1, Assuring Executive Awareness of Software Issues.

4.1.2. Executive Management Direction

Issues:

- Very few middle- and high-level managers understand software.
- The immaturity of the technology creates rapid change, which makes it difficult to keep management abreast of those changes.
- Executive management is not effectively driving the software process because there are no commonly stated policies on processes and tools, nor are there commonly stated goals for quality and productivity. Further, executive management is not adequately involved in providing process management resources. The immaturity of the software process and the changing technology makes these problems worse.

Recommendations:

- Executive management should establish, issue, and enforce policies on:
    - A basic set of common, tailorable software processes and environments.
    - Software quality and productivity.
    - Software measurement, management, and tracking.
- For a variety of reasons, traditional hardware oriented management methods have often failed to work well with software intensive programs. Executive management must recognize and accommodate the unique nature of software. Some of the key distinctions are:

    - Software design theory is incomplete. It is not based on an underlying physical theory and with one exception makes no use of the power of mathematics. (Exception: the formal logic of mathematical philosophy is used in compiler design and relational database design.) The need for experimental support of the software design process must be recognized and inserted into software development to overcome the lack of theoretical power.
    - Software is generally the most complex component of any system. Most system functionality is contained in the software. Executives usually do not expect either to be the case.
    - Software is often the critical mechanism for system integration. (The semantic and syntactic power of Ada in speeding up and simplifying integration has made this aspect especially clear.) Integration planning must be geared more to software's features than hardware's.
    - Software is the primary mechanism for fixing system deficiencies that arise outside of the software itself. Software development is often disrupted by unforeseen events occurring outside the software development plan. Projects must account for this in the beginning.
    - Software developments rely heavily on documentation. The effort required is often beyond expectations even when supported by electronic documentation. Projects must account for this.
    - Software configuration management is different from hardware configuration management. Proven hardware methods should not be imposed on software.
    - Software, unlike hardware, has no manufacturing component under which development costs can be tucked. Software development costs inevitably must be exposed as one time, paid-up-front costs.
    - Software cost and schedule estimates are indefensible under intense questioning. They tend to erode under cost-cutting pressures more readily then their hardware counterparts. Existing software costing techniques simply transfer the problem from arguments about man-hours to arguments about lines of code.
- Provide comprehensive training at the proper management levels in order to keep management cognizant of the characteristics of software identified above.
- Executive management should provide resources to implement the above recommendations, including: establishment of software engineering process groups, comprehensive training for those groups, and common development environments.

Implementation Actions:

- Executives require quarterly (or more frequent) comprehensive reviews of their current software development methods, error statistics, schedule, performance, and cost.
- Create software development environments and software process groups in all software production areas.
- See Section 2.1, Assuring Executive Awareness of Software Issues.

4.1.3. Quantitative Software Management

Issue:

The complexity of the software process is now exceeding the capability of intuitive management. Quantitative methods are now required in the areas of: process measurement, collection and analysis of historical data, collection and analysis of current data, planning, tracking, control (metrics), and risk management. This is particularly needed for evolutionary development.

Recommendations:

- Software organizations should promptly implement programs to:
    - Define, collect, store in databases, analyze, and use process data.
    - Make quantitative plans and track progress.
    - Conduct quantitative risk analyses.
- DoD contracting organizations should require the above for software-intensive programs.
- Modify procurement practices to provide contractor credit for the use of assessment procedures.

Implementation Actions:

See Section 4.1.2, Executive Management Direction.

4.1.4. Software Reuse

Issue:

While reuse is a promising technique, little progress is likely to be made without a defined software process management strategy to exploit software reuse.

Recommendation and Implementation Actions:

See Section 3.3, Reusable Software.

4.2. Requirements

Issue:

Requirements for software design cannot be adequately defined at the beginning of the design cycle. Systems are too complex, too new, and too unfamiliar; their functionality increases with time; and the user and procurement environment changes too rapidly.

Recommendations:

Institute an evolutionary development concept through changes, as necessary, to government standards and procurement policies (see Section 3.1) to:

- Encourage the general use of rapid prototyping and consider mandating its use for large, complex software systems.
- Require user involvement throughout the development cycle.
- Require basic software training for users and program office staff.
- Promote a team atmosphere between program office, developers, and supporters.
- Emphasize the need for systems design continuity throughout software development.
- Recognize the need to maintain a system design focus throughout the post deployment software support (PDSS) phase.

Implementation Actions:

- Perform a survey to determine the current state-of-use, benefits, and lessons learned from various prototyping approaches.
- Develop a requirement definition methodology for incorporating changes into different software development life-cycle models, such as waterfall.
- Develop basic training courses for program office, user, and software developers.
- Promote changes to acquisition approaches to include users and create a team atmosphere throughout the development cycle.
- Develop a process model for PDSS.
- Develop approaches and guidelines for applying prototyping throughout an evolutionary development life cycle (see Section 3.1).

4.3. Software Development Investment

Issue:

For reasons previously stated, neither buyers nor sellers of DoD systems are generally willing to make capital investments for software development. There is a failure to adequately invest in either end-product or "capital equipment" support software, in tooling for current processes, or in the costs of switching to new processes.

Recommendations:

- Develop a market for investment in software by providing matching funds for competitive prototyping and software exercises. Use of industrial Modernization Incentive Program (MIP) money for process and tool improvement should be encouraged. Demonstration of the value of a new technology and identification of requirements for new tools which stem from current and future processes should be funded. Skill and capability being equal, bidders who demonstrate high levels of investment in software development environments, tools, process improvement, and training should be favored.

- Develop the ability to measure current weaknesses of the software process and the value of proposed improvements (see Section 4.1.3). This can be accomplished by: establishing a baseline and periodically updating it; defining and tracking process metrics to understand costs and error sources; and defining and tracking tool-related metrics to understand improvements which lead to significant cost and error reduction.

- Support selection from alternative software engineering processes and tools. This can be accomplished by instituting a clearinghouse for the evaluation of information which is currently available on industry processes and tools. A tools and processes demonstration and evaluation center for use by the DoD community should be created to support this effort.

- Require strong error prevention and detection programs throughout the life-cycle. Processes which reduce errors to acceptable levels should be demanded. The development of a program of incentives for early error prevention, detection, and removal should also be instituted. The goal is to achieve warranted software.

- Change software-related accounting practices by developing a program for limited-rights licensing of support software (including warranties, upgrade plans, etc.), and by developing software depreciation schedules which foster investment in support software. Cost/benefit analyses (including suggested pricing structures) must also be developed. This would allow software development costs to be recovered under manufacturing costs.

- Provide funding to develop tools and methods to facilitate the use of rapid prototyping.

Implementation Actions:

- Create an agency to administer matching funds for tasking research and development efforts in software process support. (Perhaps Techmod for MIP funds?)
- Fund a clearinghouse for the evaluation and distribution of information, industry process and tools.
- Investigate and recommend accounting practices that will foster investment in software.
- Let contracts to develop generic design prototypes for certain domains.

4.4. Awareness

Issue:

Current awareness programs are ineffective in keeping senior management and technical staff competent in the areas of software system development technologies, methodologies, and tools.

Recommendations:

- Have organizations such as the Microelectronic and Computing Center (MCC), Software Productivity Consortium (SPC), AIA, and the SEI establish a list of software management concepts, system development concepts and technologies, and software methodologies and tools that should be understood by executives. This is required to enable executives to make better software decisions.
- To improve management awareness of software costs, schedules, and quality, a briefing program should be developed for senior management on benefits of quantitative software management.

Implementation Actions:

- Task certain organizations to prepare awareness videotapes in relevant software topics. Rent these to interested organizations to aid in amortizing the cost.
- See Section 2.1, Assuring Executive Awareness of Software Issues.

Appendix A: List of Participants and Addresses

Appendix B: Modular Systems Issue Description

A management awareness issue is the impact of changes in the design philosophy of modern avionics. Traditionally, avionic subsystems have been developed independently by separate subcontractors. Any processors involved were embedded directly in the subsystem and the software was developed by the subcontractor. All the software algorithm domain expertise was maintained by the subcontractor. The integration task of the prime was to ensure the constraints of data communication, space, weight, and power were complied with. There were few requirements for the subsystems to work together in a more integrated way.

A new concept in avionic systems has become more prevalent. First pushed by the Air Force in the Pave Pillar Program and now by the Advance Tactical Fighter (ATF) designs, a modular avionics system is now the design of choice. This design separates the avionics subsystems from the processors. The subsystems sit on a central data bus where they share the general purpose processors.

This organization has a number of advantages. First, it allows the data from a number of subsystems to be processed together to create a more integrated view to the user. Second, it allows the reconfiguration of processing among identical processors, thereby providing fault tolerance and a degraded system capability. While these advantages speak strongly in support of this hardware architecture change, there are more subtle negative impacts.

The home for specific software algorithm domain expertise has been the subcontractors. Now, as the processing moves out of the subsystems, the expertise moves as well. Prime contractors must take on a larger role not only in integration, but in the development of software running on the general purpose processors. Traditionally, a number of prime contractors may have relied on a single subcontractor for expertise in a particular area. Now each prime contractor must obtain that expertise for itself in order to do the software development on the general purpose processors. This activity is likely to dilute an already scarce resource of domain-knowledgeable programmers.

This shift in responsibilities also has an impact on the subcontractors. They see a major portion of their business disappearing. Do they just become algorithm houses, selling their people to the prime contractors?

Another impact is the stochastic nature of the resulting system. Since multiple functions are dynamically handled by a limited number of processors, the particular hardware-software-application combination is often unpredictable and possibly not reproducible. This means that such systems are often not testable by traditional means. Unless the software in such systems is of high quality, such systems will likely suffer periodic and unpredictable operational failures.

The impacts of the change in design philosophy are subtle. After examining the issues, the conclusion may be that the benefits outweigh the impacts. Yet it is important to be aware of the DoD and industry impacts this change is likely to have.

Table of Contents