Report of the

DEFENSE SCIENCE BOARD
TASK FORCE ON
DEFENSE SOFTWARE

NOVEMBER 2000

Office of the Under Secretary of Defense
For Acquisition and Technology
Washington, D.C. 20301-3140

This report is a product of the Defense Science Board (DSB). The DSB is a Federal Advisory Committee established to provide independent advice to the Secretary of Defense. Statements, opinions, conclusions, and recommendations in this report do not necessarily represent the official position of the Department of Defense.

This report is UNCLASSIFIED

OFFICE OF THE SECRETARY OF DEFENSE
3140 DEFENSE PENTAGON
WASHINGTON, DC 20301-3140

DEFENSE SCIENCE
BOARD

MEMORANDUM FOR UNDER SECRETARY OF DEFENSE (ACQUISITION, TECHNOLOGY AND LOGISTICS)

SUBJECT: Final Report of the Defense Science Board Task Force on Defense Software

I am forwarding the final report of the Defense Science Board Task Force on Defense Software.

The Terms of Reference directed the Task Force to address:

- "conditions under which procurement of defense software … can appropriately use commercial practices;"
- "what management practices DoD should employ for the most efficient and effective definition of, procurement of, integration and testing of, and maintenance of defense software;"
- "whether DoD should develop any new software tools, technologies, or libraries;"
- "what approach should be used to assure that such developments enter the mainstream of commercial industry."

The Task Force determined that the majority of problems associated with DoD software development programs are a result of undisciplined execution. Accordingly the Task Force's recommendations emphasize a back-to-the-basics approach. The Task Force sounded one note of caution in its report which should be heeded. Numerous prior studies contain valid recommendations that could significantly and positively impact DoD software development programs. However the majority of these recommendations have not been implemented. Every effort should be made to understand the inhibitors that prevented previous recommendations.

I endorse all of the Task Force's recommendations and recommend you forward the report to the Secretary of Defense.

Craig Fields
Chairman

OFFICE OF THE SECRETARY OF DEFENSE
3140 DEFENSE PENTAGON
WASHINGTON, DC 20301-3140

DEFENSE SCIENCE
BOARD

MEMORANDUM FOR CHAIRMAN, DEFENSE SCIENCE BOARD

SUBJECT: Final Report of the Defense Science Board Task Force on Defense Software

Attached is the report of the Defense Science Board Task Force on Defense Software. The Terms of Reference directed that the Task Force:

- Review the findings and recommendations of previous Department of Defense studies on software development and acquisition
- Assess the current environment to identify changes since previous studies
- Assess the current state of software development programs -- both DoD and commercial
- Identify focused recommendations to improve performance on DoD software intensive programs

The Task Force believes that although there are many challenging technical issues, the major factor in successful software development is disciplined execution. Therefore the Task Force recommends a back-to-the-basics approach. The recommendations of the Task Force can be summarized as:

- **STRESS PAST PERFORMANCE AND PROCESS MATURITY**: DoD should strengthen its past performance criteria and restrict program awards to those who have demonstrated successful software development capabilities.
- **RESTRUCTURE CONTRACT INCENTIVES:** DoD should consider adoption of much stronger (commercial-like) financial incentives to reward good performance and penalize unsatisfactory performance.
- **COLLECT, DISSEMINATE, AND EMPLOY BEST PRACTICES:** DoD should encourage the use and sharing of best practices in both the DoD and contractor base.
- **INITIATE INDEPENDENT EXPERT REVIEWS (IERS):** DoD should conduct and integrate IERs into the program development process for all DoD Acquisition Category I-III programs.
- **IMPROVE SOFTWARE SKILLS OF ACQUISITION AND PROGRAM MANAGEMENT:** DoD should upgrade its training program to include software intensive systems. Additionally, DoD should require mandatory training of program managers and key program staff before program initiation.

- **STRENGTHEN AND STABILIZE THE TECHNOLOGY BASE:** DoD should stay abreast of the commercial market and address areas that the commercial market is not adequately addressing.

The Task Force strongly believes that a back-to-the-basics approach will provide significant improvement in the performance of software-intensive DoD programs.

The Task Force would like to express its appreciation for the cooperation, advice, and help by the government advisors, support staff, and the many presenters from commercial firms and government and research organizations.

Mr. Marc Hansen
Task Force Co-Chairman

Mr. Bob Nesbit
Task Force Co-Chairman

TABLE OF CONTENTS

EXECUTIVE SUMMARY

1.1 OBJECTIVES

The Defense Science Board (DSB) Task Force on Defense Software was formed in September 1999 and tasked to:

- Review the findings and recommendations of previous Department of Defense (DoD) -wide studies on software development and acquisition
- Assess the current environment to identify changes since previous studies
- Assess the current state of software development programs – both DoD and commercial
- Identify focused recommendations to improve performance on DoD software intensive programs

1.2 PREVIOUS STUDIES

The Task Force reviewed six major DoD-wide studies that had been performed on software development and acquisition since 1987. These studies contained 134 recommendations, of which only a very few have been implemented. Most all of the recommendations remain valid today and many could significantly and positively impact DoD software development capability. The DoD's failure to implement these recommendations is most disturbing and is perhaps the most relevant finding of the Task Force. Clearly, there are inhibitors within the DoD to adopting the recommended changes.

1.3 CURRENT ENVIRONMENT

The current software development and acquisition environment is summarized as follows:

- **Technology trends.** Technology, particularly technology relating to software, is changing more rapidly than ever before. The change is fueled by the Internet economy, and there is no doubt that the commercial marketplace, not the DoD, is in the driver's seat. With these changes, there is tremendous opportunity for new systems with unprecedented capability. The changes make it necessary to stay abreast of the technology, how to apply it, and how to develop, field, and operate the systems that use it.
- **DoD software trends.** As expectations for systems capability increase in the commercial marketplace, they increase within the DoD. Demands and requirements for more capable, integrated, and user-friendlier systems are increasing. As a result, software is rapidly becoming a significant, if not the most significant, portion of DoD acquisitions. Even traditional hardware procurements such as artillery systems now contain millions of lines of software code.

- **Human resources.** With demands for information technology (IT) growing at a blinding pace, it is no surprise that there is a shortage of qualified IT personnel. Department of Labor information suggests that there are more than 50,000 unfilled software jobs today and that this number is growing at a rate of more than 45,000 jobs per year. Other sources indicate that the current shortage is as high as 400,000 jobs. This will continue to put pressure on the DoD and its industrial base as they try to maintain adequate staffing levels.
- **Software program performance statistics.** Data on software development program performance in the DoD and the commercial market is very difficult to obtain. However, studies on the topic indicate appalling performance in both environments. The Standish Group Chaos Study1, published in 1999 which included government and commercial programs, states that only 16% of programs complete on budget and schedule, 31% are cancelled, and the remaining 53% have cost growth exceeding 89%. In addition, the study indicates that the average final product consists of only 61% of its originally proposed features.
- **Security.** Although a great deal of effort is being applied to this area in the commercial market, there is a need for the DoD to invest in new security technology. The commercial market is attacking data security and beginning to address network security; they have not begun to address software security issues.

1.4 MAJOR FINDINGS AND RECOMMENDATIONS

The troubled DoD programs reviewed by this team exhibited fundamental problems that were readily identifiable, at least in hindsight. Too often, programs lacked well thought-out, disciplined program management and/or software development processes. Meaningful cost, schedule, and requirements baselines were lacking, making it virtually impossible to track progress against them. In addition, there were numerous examples where the acquisition and/or contractor team lacked adequate software skills to execute the program. In one case, a program requiring more than 2 million lines of real-time embedded code was awarded to a contractor who had no meaningful software development experience.

In general, the technical issues, although difficult at times, were not the determining factor. Disciplined execution was. As a result, the Task Force recommends a back-to-the-basics approach. We strongly endorse the following recommendations:

- **Stress past performance and process maturity.** The Task Force recommends that the DoD strengthen its past performance criteria and restrict program awards to those who have demonstrated successful software development capabilities. In addition, we recommend that software programs only go to those who have demonstrated Software Engineering Institute (SEI) Capability Maturity Model (CMM) Level 3 or equivalent processes. Process certification or recertification should be no more the 24 months old. Recent past performance (less then two years) can be considered in lieu of recertification for organizations that have already completed an initial certification. The current DoD policy has an escape clause that allows a risk plan in lieu of

[1] Chaos Report, The Standish Group, 1995, 1999

certification. We strongly recommend this clause be eliminated. Care must be taken to ensure that subcontractors performing key software tasks have the adequate skills.

- **Initiate Independent Expert Reviews (IERs).** The Task Force recommends conducting IERs for all DoD Acquisition Category (ACAT) I-III programs. These reviews are intended to help the program team ensure that: disciplined processes and methodologies are in place, that the program is adequately resourced, that the technical baseline is understood and solid with attendant risks and opportunities identified and managed, and that adequate progress is being achieved. The review team should consist of government, academic, and industry experts who have program and software management skills, technical skills appropriate to the program, and requisite domain knowledge. To ensure objectivity, the team should not include individuals on the program. To be effective, the IERs must be integrated into the program development process. Reviews should be held at key program milestones or at least every six months. Review findings should be reported and actions tracked until closure. IERs are common in industry and have led to significant improvements where used

- **Improve software skills of acquisition and program management.** The Task Force recommends that the DoD update its training programs to include software-intensive systems and that the DoD require mandatory training of program managers and key program staff before program initiation. In addition, the Task Force recommends mandatory government/contractor team training at program initiation and at selected key milestones. Such training not only builds teamwork, but also provides a forum to ensure that program issues and processes are understood and to review recent technological advances that may benefit the program. The Task Force also recommends that a software systems architect be assigned to each program to be responsible and accountable for the software system. With the increasing shortage of software talent, the DoD should consider development of a graduate program for software systems development.

- **Collect, disseminate, and employ best practices.** With the rapid changes in technology, it is important to encourage the use and sharing of best practices in both the DoD and contractor base. The DoD should consider providing awards to teams that exhibit this behavior. The Task Force strongly endorses the following best practices:

  - *Executable architectures*. Sound architecture should be emphasized early in the program. Software architecture is the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time. [2] Software architectures can be described and simulated using formal notations such as the Unified Modeling Language (UML). Critical architecture components/elements can be required with a proposal and tested as executables enabling early design validation and risk mitigation. An executable architecture typically starts with the necessary components defined but implemented as skeletons only. Together with definitions

[2] David Garlan and Dewayne Perry, guest editors, April, 1995 IEEE Transactions on Software Engineering special issue on software architecture

of the interfaces between components, the developer can exercise this architecture (e.g., executing data and control flow) after only one or a few key scenarios have been explored. Subsequent iterations can then leverage this architecture and, early on, redefine or refine it to correct mistakes and also incrementally add more functionality to it.

- *Iterative design/development.* Various methods now allow iterative design/development. These, when done properly, retire risks earlier, provide the end-user capability sooner, and produce systems superior to those developed with the traditional waterfall method.
- *Requirements trade off.* In general, systems are over-specified, and in most cases there is no flexibility to adjust the specifications. The acquisition/development team must have latitude to trade requirements for cost, schedule, and risk. This does not mean that overall system5 integrity can be compromised.

- **Restructure contract incentives.** DoD and commercial approaches to profits and penalties differ dramatically. With DoD contracts, profits are typically limited to 15%, and, in reality, there is very little penalty for non-performance. In the commercial environment, profits of 30% are common and non-performance can quickly result in contract cancellation and financial liabilities. The DoD should consider adoption of more commercial-like incentives, including the conversion from Cost-type contracts to use of Fixed Price-type contracts, after the requirements and design are solidified and program risk stabilized.
- **Strengthen and stabilize the technology base.** There is no question that the commercial market, not the DoD, is driving much of today's technology. There are, however, areas that the commercial market is not addressing that are critical to DoD success and that need DoD investment. It is critical for the DoD to stay abreast of the commercial market and to influence the market where it can. As a result, it remains important to the defense community to maintain a strong technology base. Although it may be tempting in this environment to cut research when budgets get tight, one must be careful to maintain one's critical base. This is particularly true in today's market where our key researchers are highly recruited by both the DoD industrial base and the commercial market.

In summary, the Task Force believes that a back-to-the-basics approach, as reflected in our recommendations, will provide significant improvement in the performance of software-intensive DOD programs. The most disturbing finding was that recommendations from previous studies had not been implemented. With the dramatic changes occurring in the information technology arena, the DoD must understand why these recommendations were not acted upon and foster an environment that readily accepts and adapts to change or else continued failure, excess cost, and damage to our national security posture will ensue.

2. TASK FORCE OVERVIEW

2.1 TERMS OF REFERENCE

The DSB Task Force on Defense Software was formed in September 1999 with tasking outlined in a Terms of Reference letter[3] from the Under Secretary of Defense for Acquisition & Technology. The Task Force was created to address the problem of defense development and acquisition programs continuing to experience "software problems." These problems have resulted in significant cost overruns, schedule slips, and performance difficulties.

The Task Force was asked to determine:

- "conditions under which procurement of defense software … can appropriately use commercial practices"
- "what management practices DoD should employ for the most efficient and effective definition of, procurement of, integration and testing of, and maintenance of defense software"
- "whether DoD should develop any new software tools, technologies, or libraries"
- "what approach should be used to assure that such developments enter the mainstream of commercial industry"

The Task Force was specifically asked to consider a number of topics related to DoD and commercial software development. These topics included:

- State-of-the-art and best practices
- Development tools
- Incorporation of information security/assurance techniques and practices
- Reusable software components
- Techniques and tools for tailoring available components for use in defense systems
- DoD management of development process
- Software risk management techniques/tools
- Minimum delivery time
- Affordability
- Maintenance and post-deployment enhancement
- Process support tools
- Quality and assured availability
- Use of development and maintenance tools

[3] See Appendix A for a copy of the Terms of Reference

2.2 CAVEATS

Since a comprehensive survey of DoD software-intensive programs and software technologies and methodologies was infeasible given the time constraints, the Task Force relied on inputs from a representative sampling of programs and new technology efforts (see section 2.3). This report is based on those inputs and further investigation by Task Force members.

This report contains no detailed quantitative assessment or evaluation of individual topics. Instead, the Task Force has focused on providing a small number of recommendations that can be implemented relatively quickly.

2.3 APPROACH

The Task Force met for the first time in October 1999 and met once a month thereafter until April 2000. The purpose of the monthly meetings was to receive briefings from and interact with:

- Managers of major DoD software-intensive programs. These programs included large weapons systems, C^3I systems, and management information systems. Programs from each of the three services were examined
- Industry representatives (both commercial industry and government contractors) involved in software-intensive programs
- Managers from DoD software technology programs
- Technical experts in the field of software engineering

The Task Force also met in executive session during each of these meetings to deliberate on these interactions and determine how best to proceed with recommendations. Agendas for each of the monthly meetings are attached in Appendix B.

2.4 MEMBERSHIP

The Task Force comprised representatives from industry, academia, and government, all of whom were either senior managers involved in software-intensive projects or recognized experts in the field of software engineering and development. A list of the Task Force members is included in Appendix C.

2.5 PREVIOUS STUDIES

> *"Many previous studies have provided an abundance of valid conclusions and detailed recommendations. Most remain unimplemented. If the military software problem is real, it is not perceived as urgent. We do not attempt to prove that it is; we do recommend how to attack it if one wants to."*

This quote would be a fitting introduction to this report. However, it is the introduction to the report from the 1987 Task Force on Military Software. Sadly, the findings and recommendations of the current task force are strikingly similar to those found in the 1987 report and many other past studies.

The current task force reviewed past reports by the Defense Science Board, the science panels of the military services, and the National Research Council. These reports included:

- Report of the Defense Science Board Task Force On Military Software - 1987
- Adapting Software Development Policies To Modern Technology - 1989
- The Report of the AMC Software Task Force - 1989
- Scaling Up: A Research Agenda For Software Engineering - Computer Science and Technology Board Research Council - 1989
- Defense Science Board Task Force on Acquiring Defense Software Commercially - 1994
- Report of the Defense Science Board Tasks on Open Systems - 1998

The reports offered a total of 134 recommendations. The recommendations can be classified into five areas:

- Software architecture (a central theme for software reuse, product lines, and greater exploitation of commercial technology and practices)
- Software technology
- Workforce issues (e.g., sufficient staffing, proficiency levels, and training)
- Contract strategy (e.g., tailoring guidelines and incentives)
- Acquisition policy

Figure 2.5a (see page 4) shows the distribution of recommendations along these five areas and this task force's assessments of which recommendations were implemented both in policy and in practice. Of the 134 recommendations, only 3 are in practice and only 18 are in policy. Reviews of the remaining 113 recommendations indicate that most are still valid and can make a significant difference in the DoD’s software development capability.

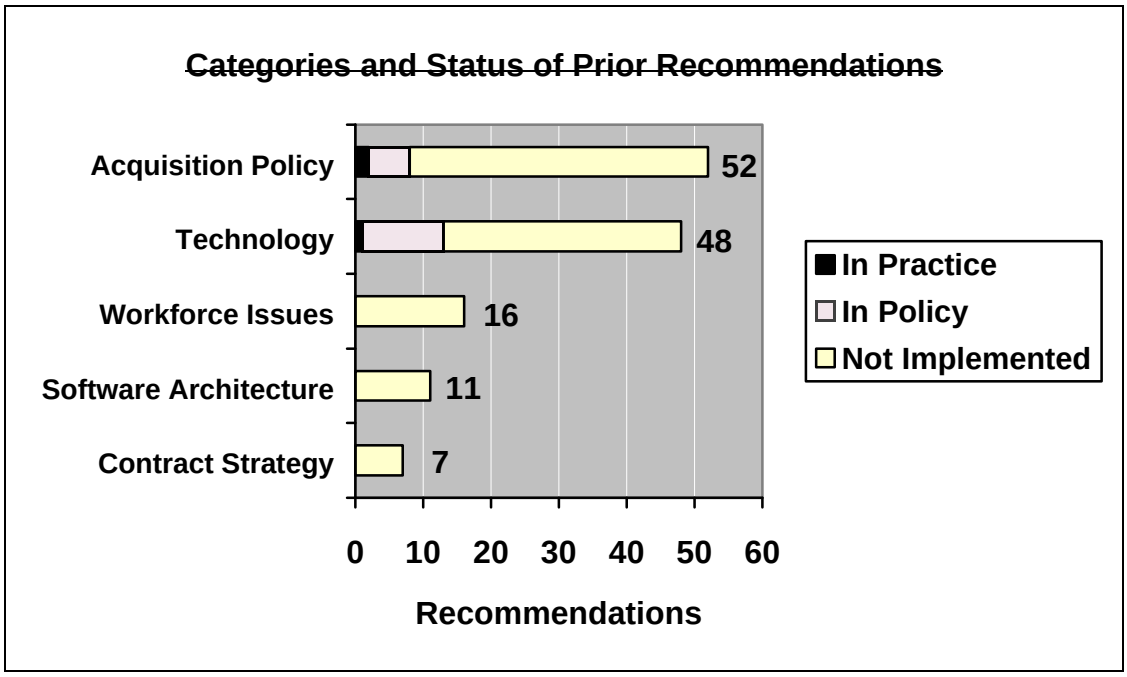


Figure 2.5a – Status of recommendations from prior DoD-wide software task forces

While the current DoD acquisition policy incorporates some of these suggestions (as evidenced in the revised 5000 series of DoD policy guidance and regulations) and reflects what is described later in this report as modern software engineering practice, few of the recommendations have been realized in DoD practice. For example, the following recommendations from the 1987 DSB Task Force on Military Software report have yet to be implemented in DoD practice:

- The DoD should use evolutionary acquisition, including simulation and prototyping to reduce risk.
- The DoD should devise increased productivity incentives for custom-built software contracts, and make such contracts the standard practice.
- The DoD should devise increased profit incentives on software quality.
- The DoD should develop metrics and measuring techniques for software quality and completeness and incorporate these routinely in contracts.
- The DoD should enhance education for software personnel.

A few other observations from past studies are worthy of note. The 1987 DSB Task Force observed that requirements-setting and management are the hardest part of the software task and advocated the use of evolutionary practices. This is still true today. A key difference between successful commercial practice and common DoD practice is the extent to which evolutionary development is the norm in commercial practice. To understand why this disparity, the 1989 USAF SAB study explored the role of the experience of the team of acquirers and developers on large software projects. It observed, from an analysis of 17 major software-intensive systems, that the level of team experience with requirements, architecture and technology, and team

processes and communication patterns on similar systems was the dominant reason for a project's success or failure (as reflected by cost, schedule, and, in many cases, cancellation).

The 1994 study on commercial practices went further and highlighted significant differences between typical commercial and defense software acquisition. It found that the former was based on trust and the later was based more on adversarial relationships. Our task force observed in interviews with DoD and industry officials a significant level of distrust and “non-teaming” between DOD acquirers and developers.

The 1994 study also recognized that modern software architecture methods and product lines could improve cost and cycle time. Technical and management practices for better requirements management were described and recommended long ago, as was the importance of team experience and technical practices related to architecture reuse. These practices and qualities are hallmarks of commercial best practice, but they remain largely underutilized in the acquisition and development of defense software.

This current task force review of past recommendations resulted in the same observation made in the 1987 report: “To a surprising degree, the conclusions of these studies agree with each other and remain valid; the recommendations continue to be wise. The chairman of several study groups briefed us. All had one message: very little action has been taken to implement the recommendations. If the military software problem is real, it is not perceived as urgent by most high military officers and DoD civilian officials. Our Task Force does not undertake to prove that is urgent; we do tell how to attack it if one wants to.”

Our current Task Force could not state this any better.

3. CURRENT ENVIRONMENT

3.1 TECHNOLOGY TRENDS

The breathtaking pace of technology updates, which is fueled by the new Internet economy, has taken even the most technology-savvy organizations by surprise. The real payoff has not been the new gizmos and tools themselves, but the human productivity gains that have been promised since the introduction of the personal computer in the early 1980's. The landscape of technological change includes the following:

- **Growth of the Web.** The most obvious trend has been the explosive increase in Web-based software applications. The speed with which private industry has adopted Web-based application development in the last three to five years is as dramatic as it is unexpected. As of 1999, more than 50% of all Internet traffic was supporting Web-based commercial applications dominating e-mail, file transfers, and other uses.

- **Fewer client-server architectures.** The trend in software architectures for new or migrated mission-critical applications has been away from two- and three-tier client-server architectures toward Web-based and/or thinner clients. The decline of client-server applications owes as much to problems with scalability, deployment, and end-user performance as to the buzz over Web-based architectures. In a sense, more centralized data processing is returning, leaving user interface tasks for the client primarily in medium- to large-scale systems. In addition, legacy systems have not been replaced at the rate once expected. Y2K renovations and the addition of Web-based and other front ends have extended the life of legacy centralized systems.

- **Growing interconnectivity of systems and applications.** To a large extent, the network has become an even more important component of system architectures. Although security concerns remain high, especially within the DoD, there has been a clear trend toward linked systems. This has produced a strain on the country's telecommunications infrastructure, worsening end-user performance. National and international telephony and communications networks have responded and are working to support the growing predominance of digital data traffic. Although demand has nearly outpaced supply, the telecommunications infrastructure is expected to support the dramatically increased data traffic within five years.

- **Increased standardization.** Increased standardization has occurred on all fronts, perhaps most noticeably through the de facto standardization on commercially successful products such as Microsoft Office. Other examples of increased standardization include:

  - Networking protocols (e.g., TCP/IP)

  - E-mail protocols allowing increased sophistication when transmitting across networks (allowing not only the use of attachments through SMTP protocol, but also increased ability to retain formatting features).

- The successful use of protocols such as Microsoft's Object Linking and Embedding, which allow end-users to share pieces of different applications within a single document with dramatically improved ease of use.
- Further adoption of SQL, relational databases, and in particular, Oracle and Microsoft's SQL server as the standards for new development.
- Increasing platform independence (through the rising use of Java and other standards-based development tools, including Common Object Request Broker Architecture (CORBA) and Enterprise Java Beans).
- The key role played by international standards organizations such as the Internet Engineering Task Force (ietf.org) for internet standards such as TCP/IP, the International Telephony Union (ITU) for audio, video and data conferencing standards (e.g., H.32x, T.12x), the World Wide Web Consortium (W3C) for web standards *(www.org)* such as HTML, HTTP, POP and IMAP, the 800+ corporation consortium of the Object Management Group (omg.org) providing distributed computing standards, and the Open Geospatial Consortium *(ogis.org)* providing global mapping, image and Geospatial standards.

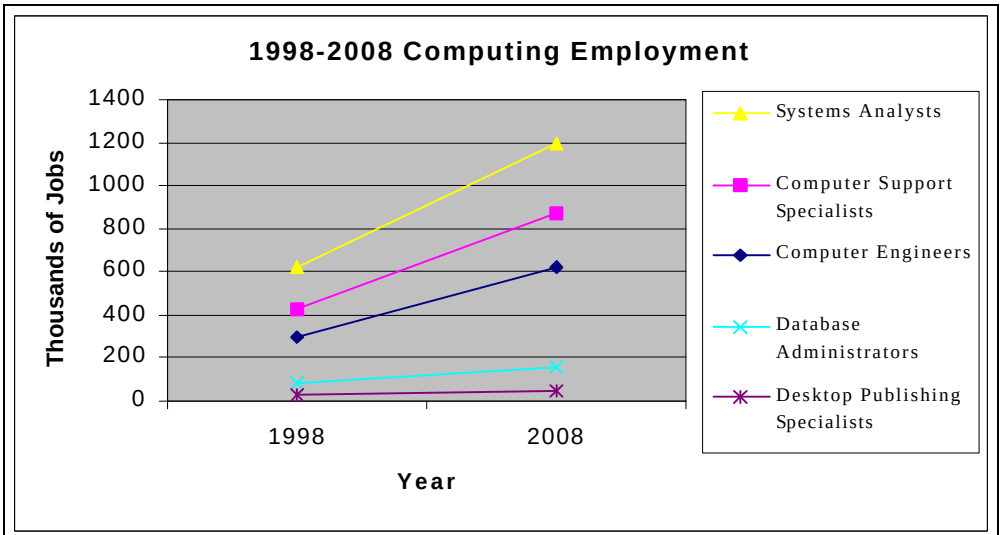
- **Wireless and handheld devices.** Wireless and handheld devices appear to be the focal point of a number of advancing technologies, from Internet-connected automobiles to "personal digital assistants," which combine Internet and telephone access with local computing power. While today there are 200 million PC users, there are 600 million wireless users and 1.5 billion telephone users[4]. By 2004 one billion wireless smart-phones are expected to be connected to the Internet. Secure versions of these devices could be applied to military use.
- **Less custom development, more integration.** An increasing focus on the integration of pre-defined parts or components is driving the professional services and software industries. The Object Management Group's CORBA, Microsoft's Component Object Model, the use of "product lines" in software development, and the latest interface solutions such as "Enterprise Application Integration" all appear to be realizing the 25-year-old promise of reusable code. This trend is also reflected in the increased use of Commercial Off The Shelf (COTS) and Enterprise Resource Planning packages for "standard" applications, – human resources, inventory, payroll, logistics, etc. – in the commercial and public sectors.

3.2 HUMAN RESOURCE TRENDS

Expert software development and acquisition professionals are essential with the DoD's increasing dependence on software in major defense systems. Unfortunately, there are not enough of these professionals. The DoD and its contractor base are increasingly competing with commercial software organizations for a limited pool of qualified software professionals. The commercial software market is one of the highest growth industries, and the insatiable appetite for qualified information technologists has resulted in a severe global shortage of experienced personnel. Consider the following table and statistics:

[4] Source: Nokia/DataQuest

- The Department of Labor estimates that more than 100,000 information technology jobs are added annually. In addition, about 25,000 replacement workers are needed each year.
- The number of software and data processing jobs is growing about seven times faster than the national average.
- The Department of Labor reports that five of the top 10 fastest growing occupations are computer software disciplines (See Figure 3.2.a).



Source: Occupational Outlook Handbook, U.S. Dept of Labor, Bureau of Labor Statistics

Figure 3.2a – Computer Employment Growth

- An analysis of registered undergraduate and graduate students by the National Center for Education Statistics (see Figure 3.2b) shows that the supply of computer science graduates (approximately 36,000 annually from 1991-1999) will only increase to about 60,000 (50,000 of which will be matriculating undergraduates) by 2006.
- The two most related disciplines, electrical engineers and mathematicians fill some of the shortfall. For example, about half of the annual 20,000 electrical engineer graduates and undergraduates end up working in computing jobs. The same is true with math graduates and undergraduates.
- Physics (approximately 5,000 graduates annually) and other disciplines provide a much smaller contribution to the computer professional pool.
- Perhaps 40% of graduating information technology students are foreign students, half of whom return to their homeland.
- Our ability to tap into foreign workers is limited by Congress (typically between 65,000 and 125,000 workers annually). Shipping software development work offshore has security implications and, in any case, global markets from Ireland to India to Israel report similar software worker shortages.

To summarize, the shortfall of computer science professionals grows by about 45,000 each year (see Figure 3.2a) . The economic impacts of these shortages include wage pressure that can lead to inflation and severe understaffing that can lead to lower productivity. Both of these trends diminish U.S. economic competitiveness.

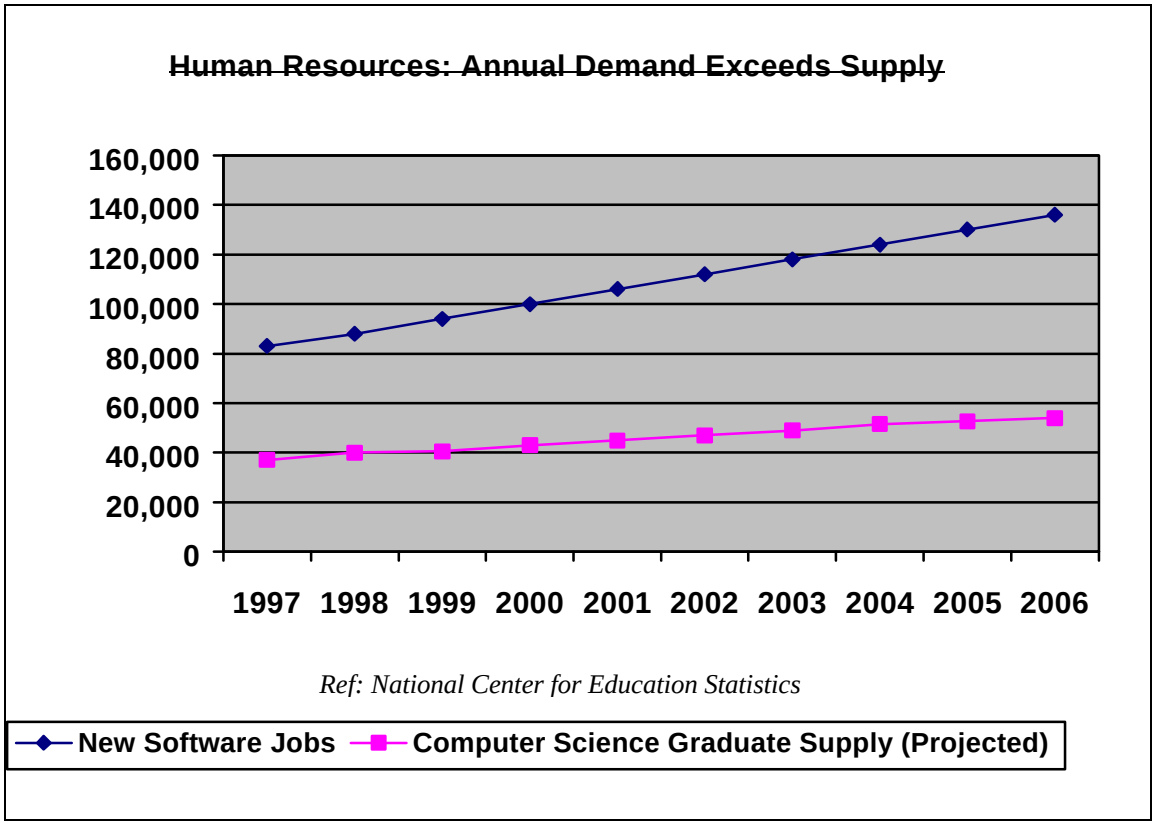


Figure 3.2b – Computer science graduates: supply vs. demands

Already, the United States has one of the most expensive software unit costs in the world at approximately $500 per function point. Software professional turnover rates (approximately 14.5% on average in the United States in 1997) are among the highest industry turnover rates. It is increasingly common to hear of software development organizations that have 20%, 30% or even higher unfilled staffing requisitions. In today's market, it is not unheard-of to find those with key software skills receiving annual 20% compensation increases.

With limited ability to rapidly change worker compensation, it is increasingly difficult for the DoD to hire and retain software talent. The promise of high cash and stock rewards from explosive growth in the dot.coms and the devaluation of defense stocks exacerbate the ability of the defense industry to acquire and retain the same talent, and 58% of hiring managers expect turnover rates to increase. The gap between supply and demand is actually widening and is projected to continue to widen.

In conclusion, DoD systems are increasingly reliant upon software and software professionals. There is a shortage of sufficiently qualified software personnel at all levels and the demand for qualified personnel is projected to increasingly outstrip supply.

3.3 SOFTWARE PROGRAM STATISTICS

Data regarding performance of software development programs is extremely difficult, if not impossible, to obtain. Studies on the topic, however, report appalling results in both the commercial and DoD environments. One such study conducted by the Standish Group[5] looked at both commercial and government IT projects and identified several disturbing statistics:

- Only 16% of all IT projects complete on time and on budget.
- 31% are cancelled before completion.
- The remaining 53% are late and over budget, with the typical cost growth exceeding the original budget by more the 89%.
- Of the IT projects that are completed, the final product contains only 61% of the originally specified features.

These statistics are more disturbing when one considers the growing importance of software development in system procurements. Software is becoming a more dominant, if not the most dominant, portion of a system acquisitions. One example of this is in combat aircraft. As shown in Table 3.3a, the percentage of functionality requiring software has grown with each successive generation of combat aircraft. Functionality as basic as flight is no longer possible without sophisticated computer systems aiding the pilot. Similar trends can be seen in commercial aircraft systems.

Weapon System	Year	% of Functions Performed in Software
F-4	1960	8
A-7	1964	10
F-111	1970	20
F-15	1975	35
F-16	1982	45
B-2	1990	65
F-22	2000	80
Source: PM Magazine		

Table 3.3a – System functionality requiring software

With this growth in functionality supported by software comes the associated growth in the number of lines of codes that must be developed and maintained to implement new systems. Aside from functionality, this growth in software lines of code can be attributed to several factors:

[5] CHAOS Study, Standish Group, 1999

- **Sophisticated user and database interfaces.** Modern software systems have robust user interfaces that are highly graphical in nature. In addition, many systems have extensive data handling requirements that necessitate efficient interface to Database Management Systems (DBMSs).
- **Modern programming languages.** Well-structured programming languages such as Ada naturally generate more code as a result of the increased type and error checking done within the software. Early generations of software, which were optimized for memory-constrained systems, frequently omitted this functionality.
- **"Excess" code.** Usage of automatic code generators and reusable software modules often results in code that is unneeded and cost prohibitive to remove from the system.

These software trends are causing software development and testing to be a major driver in the schedule and budget for system acquisitions. Given the pervasive utilization of software throughout the overall system, these trends also mean the software developer is often in a better position to perform overall system integration, a role traditionally held by the hardware manufacturer.

3.4 DoD versus Commercial Practices

In comparing DoD and commercial software development practices, the Task Force looked at six key characteristics:

- Success rate
- Best practices
- Complexity
- Process (theory)
- Incentives
- Limiting losses

The success and failure rate of DoD and commercial systems appears to be equivalent. As indicated previously, data regarding performance is difficult to obtain. However, studies reveal appalling performance in both environments.

The application of best practices also seems unimpressive in both environments.

The DoD tends to deal with more complex systems than does the commercial world. This complexity is driven by the requirement to provide greater functionality and higher reliability then commercial systems (see Figure 3.4a). This requirement is no surprise when one considers the critical nature of the systems developed by the DoD – lives and our nation's defense are at stake.

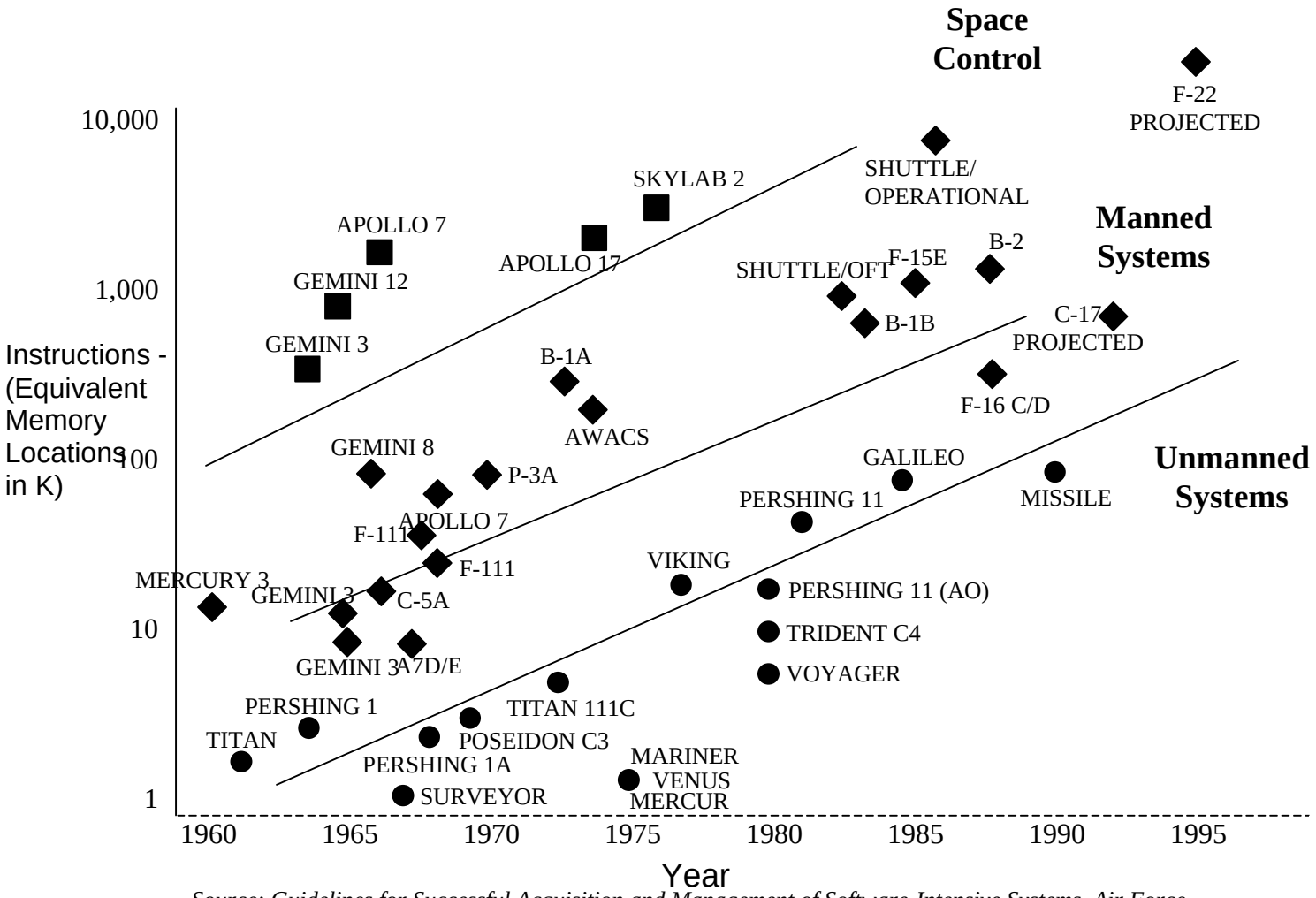
In general, the DoD was superior to the commercial marketplace in both the definition and execution of disciplined processes. This is no surprise given the natural diversity of the commercial marketplace and the more controlled environment within the DoD. One should not take comfort in this finding, however. Lack of disciplined execution is a major issue within the DoD.

The commercial environment exhibited a greater use of incentive-based reward systems (at both the organizational and individual level) and a higher sensitivity to limiting losses. The commercial world is better able to quantify the value of success and the cost of failure and translate them into effective incentive systems. The commercial world is also better able to perform cost-benefit trade-offs and determine when functionality should be reduced in order to meet cost and schedule constraints or to completely end a project.

The Task Force looked at numerous software development best practices utilized on commercial programs. There were several best practices that reoccurred in discussions. They included:

- Allowing program management to trade functionality for cost, schedule, risk, and stability and still meet overall system objectives
- Utilizing executable architecture approaches to validate design and insisting on an architecture-first methodology
- Using iterative-based development processes to minimize risk, validate requirements, and refine operational concepts[6]
- Utilizing short (no more than 18 months) development cycles
- Incentivizing development teams based on meaningful, quantifiable metrics

[6] See Appendix D for more details on iterative development.



Source: Guidelines for Successful Acquisition and Management of Software-Intensive Systems, Air Force Software Technology Support Center, June 1996

Figure 3.4a –Code Size/Complexity Growth

- Setting clear goals and decision points and being willing to terminate projects that fail to meet goals

The DoD is making increased use of architecture-first designs and iterative (Spiral) development processes. The remaining four practices were common in commercial projects but almost nonexistent in DOD projects.

3.5 WATERFALL MANAGEMENT VS. ITERATIVE MANAGEMENT

Most software engineering projects still employ a waterfall model as a software management process. Conventional, waterfall software management techniques work well for custom-developed software where the requirements are fixed when development begins. The life cycle typically follows a sequential transition from requirements to design to code to testing, with ad hoc and excessive documentation that attempts to capture complete intermediate representations at every stage. After coding and unit testing of individual components, the components are compiled and linked together (integrated) into a complete system.

Some of the key tenets of conventional software management include: freezing requirements before design, forbidding coding before detailed design review, completing unit testing before integration, maintaining detailed traceability among all artifacts, thoroughly documenting each stage of the design, inspecting everything, and planning everything early with high fidelity. Enforcing this sequential management approach usually results in significant inconsistencies among component interfaces and behavior that can not be identified until integration. These architectural inconsistencies are extremely difficult to resolve and integration almost always takes much longer than planned. Budget and schedule pressures drive teams to shoehorn in the quickest fixes. Redesign usually is out of the question. Testing of system threads, operational usefulness, and requirements compliance is performed through a series of releases until the software is judged adequate for the user.

About 90% of the time, the process results in a late, over-budget, fragile, and expensive-to-maintain software system. A typical result of following the waterfall model is that integration and testing consume too much time and effort relative to the other software development activities. Most waterfall projects, expend over 40% of their effort and schedule in integration and testing.

The software industry, both commercial and defense sectors, has been evolving the software management process for many years transitioning from the conventional waterfall model to modern, iterative development. Modern software management approaches produce the architecture first, followed by usable increments of partial capability, and then focus on complete precision later in the life cycle. The significant (architectural) requirements and design flaws are detected and resolved earlier in the life cycle, avoiding the big-bang integration at the end of a project. Quality control improves because system characteristics inherent in the architecture (such as performance, fault tolerance, interoperability, and maintainability) are identifiable earlier in the process where problems can be corrected without jeopardizing target costs and schedules.

Some of the key principles of modern software management are: establishing an architecture-first approach and an iterative life-cycle process that confronts risk early, transitioning design methods to emphasize component-based development using visual modeling, establishing a life-cycle change-management environment that enhances change freedom through tools that support round-trip engineering, instrumenting the process for objective quality control, using a demonstration-based approach to assess intermediate artifacts, and planning intermediate releases in groups of usage scenarios with evolving levels of detail.

Where conventional approaches mire software development in integration activities, these modern principles result in less scrap and rework through a greater emphasis on early life-cycle engineering and a more balanced expenditure of resources across the core workflows of a modern process. Demonstrations, enabled by the architecture-first approach, force integration into the design phase. They do not eliminate design breakage, but they make it happen when it can be addressed effectively. By avoiding the downstream integration nightmare (along with late patches and sub-optimal software fixes), a more robust and maintainable design results. Interim milestones provide tangible results. The project does not move forward until it meets the demonstration objectives. This process does not preclude the renegotiation of objectives once the interim findings permit further understanding of the trade-offs inherent in the requirements, design, and plans.

The resource allocations in Table 3.5a reflect experience in numerous waterfall process projects and several successful iterative process projects. These values are deliberately imprecise; their purpose is to relate the relative trends over time.

Life-cycle activity	Conventional Waterfall	Modern Iterative
Management	5%	10%
Requirements	5%	10%
Design	10%	15%
Implementation	30%	25%
Test and assessment	40%	25%
Deployment	5%	5%
Environment/tooling	5%	10%
Totals	100%	100%

Table 3.5a. Resource expenditure allocations.

Several major trends will surface in the coming years:

More automation of implementation activities and reuse of commercial components will reduce implementation activities, resulting in relatively more burden on requirements and design activities and environments.

- More mature iterative development methods and Web-based architectures will drive deployment activities into a larger role within the life cycle.
- More mature iterative development environments (process and tooling) will enable further reduction of life-cycle scrap and rework.

Because iterative development is more challenging than the simple management paradigm presented by the waterfall model, disciplined software management and common sense will remain one of the paramount discriminators of software engineering success or failure.

3.6 INFORMATION SECURITY

The broad subject of information security can be partitioned into subtopics to include operating system security, network security, and application security. While there is overlap among these terms, operating system security generally deals with the protections and mechanisms inherently provided with the operating system such as ability for least privilege and separation of process space. Network security includes protections and mechanisms such as encryption to protect data while in transmission, firewalls to protect enclaves at the same classification level, guards to separate networks of different classification levels, and network level intrusion detection systems. Application security deals with the protection and integrity of the code itself to include detection of malicious code, software best practices, and using the security features that the software provides. All three areas depend on diligently following configuration management, securely configuring the components, and well trained operators.

Operating system security has been a longstanding topic while network security has really come into focus during the 1990s since the advent of the Internet and emergence of hacking. The subject of application security is still a somewhat more obscure and less-often-researched area, certainly in terms of its visibility in the popular press.

Yet from a system vulnerability prospective, corruption of the applications software would appear to be one of the most likely and effective approaches any serious hostile country might take to disrupt DoD information systems – particularly once the DoD begins to encrypt to the desktop.

The Task Force has concluded that secure software is a priority area requiring focused attention by the DoD and is not, at least currently, an area where the DoD can rely on commercial products or emulate commercial practices.

There are several opportunities and approaches that can be used to corrupt software involved in national security applications:

- Many DoD systems are developed in an open environment. The code sits on unprotected machines that are connected to the Internet either for distributing that code among development team members or for other purposes.
- The DoD is using more and more “shrink-wrapped” COTS software and more and more of that software is being developed offshore.
- Once systems are fielded, code upgrades are frequently downloaded over the Internet without the source of the code being verified or the download route secured. This is particularly true for upgrades of common COTS applications.
- Finally, the most straightforward and time-tested approach involves insider access – corrupted software is installed while the system is unprotected, during routine maintenance, or through a variety of other tools of the espionage trade.

The growth and demands of e-commerce have pushed and will continue to challenge commercial information security advances. However, not surprisingly, these advances have been targeted against the "popular" threat approaches – viruses, teenage hackers, Web-page trashing, etc. Even something as straightforward as denial-of-service attacks had been ignored commercially until a rash of such incidents in late 1999.

The DoD cannot and should not ignore the subject of protecting the security and integrity of the application software that underlies so many of its systems. Research and development, serious integrity testing, improved software security practices, and secure distribution and handling policies must all be addressed.

For example, developers can use tools to automatically detect software defects (e.g., programming errors), which can enhance software security. These include static analysis tools, which parse source code and generate control and data flow information (e.g., procedure call trees and variable set/use reports). There are also dynamic analysis tools, which instrument source or object code and identify various problems, such as memory allocation and array references. In addition to analyzing software to identify defects in general, developers can also analyze software to identify security vulnerabilities and detect malicious code. Finally, there are mechanisms to gain assurance by enforcing security policies at runtime. Additional research is required to build more sophisticated tools and deal with more complex and larger scale software systems.

4. MAJOR FINDINGS AND RECOMMENDATIONS

4.1 MAJOR ISSUES ARE FUNDAMENTAL

The troubled DoD programs reviewed by this team exhibited fundamental problems that were readily identifiable, at least in hindsight. Too often, programs lacked well thought-out, disciplined program management and/or software development processes. Meaningful cost, schedule, and requirements' baselines were lacking, which prevented any possibility of tracking progress against them. In addition, there were numerous examples where the acquisition and/or contractor team lacked adequate software skills to execute the program. In one case, a program requiring more than 2 million lines of real-time embedded code was awarded to a contractor that had almost no meaningful software development experience.

In general, however, disciplined execution was the determining factor, not technical issues. This problem is exacerbated by a never-increasing shortage of qualified software development personnel and rapidly changing technology. As a result of these findings, the Task Force has recommended a back-to-basics approach, which consists of six fundamental recommendations.

The first recommendation, *Stress software past performance and process maturity*, is directed at doing business with contractors who have demonstrated capabilities to be successful on software-intensive programs.

The second recommendation, *Initiate Independent Expert Reviews (IERs)*, is directed at two issues. The first issue is to ensure that software-intensive programs are being appropriately executed and that cost, schedule, technical, resource, and process issues are being adequately addressed. The second is to share scarce technical resources across a broader set of programs.

The third recommendation, *Improve software skills of acquisition and program management*, is directed at ensuring that DoD acquisition personnel are adequately trained on software-intensive programs (not the case today) and that the DoD is taking proactive steps to deal with the increasing shortage of software personnel.

The fourth recommendation, *Collect, disseminate, and employ best practices*, is directed at encouraging those within the DoD to keep pace with and adapt to the changes. Technology is moving at a blinding pace, and with it both opportunities and risks abound.

The fifth recommendation, *Restructure contract incentives*, is directed at employing commercial performance incentive practices, which the Task Force believes will strengthen both the DoD and its industrial base. The Task Force found that the most dramatic difference between the DoD and the commercial market involved performance incentives. In the DoD environment, profits are typically limited to 15% with little penalty for performance failures. In the commercial, market profits of 30% are common and poor performance can quickly lead to termination with significant financial liabilities.

The final recommendation, *Strengthen the Technology Base*, is directed at maintaining a viable research capability within the DoD that focuses on technology not being provided by the commercial marketplace. The commercial market, not the DoD, is clearly driving today's information technology. DoD, however, must stay abreast of the most current technology, and

there are areas that are imperative to the success of DoD which are not being addressed by the commercial market place.

The Task Force believes that the implementation of these six recommendations will provide a significant improvement in DoD software development capabilities and success.

4.2 STRESS SOFTWARE PAST PERFORMANCE AND PROCESS MATURITY

The shortage of software skills throughout the industry, coupled with the pace of technological change, has made finding qualified software development contractors increasingly difficult. The Task Force recommends strongly weighting past performance and development process maturity in the source selection process. Software-specific performance data must be collected during the Contractor Performance Assessment Reporting (CPAR) process and kept in a central data-base. The database needs to include assessments of performance from the DoD program manager, the program executive officer (PEO), and the user community, and it should include the results of the Independent Expert Reviews (see Section 4.3).

A common criteria for performance assessment must be provided. The Task Force suggests something simple, such as, "Would you do business with this organization or division again?" The information in the source selection process must be weighted consistent with the percentage of specified functionality implemented or controlled by software. Attention must also be paid to subcontractors who have significant software responsibilities on the program.

In addition to a favorable past performance rating, the Task Force recommends that the DoD require all software development contractors/subcontractors to demonstrate CMM SEI Level 3 or equivalent processes. The CMM Level 3 or equivalent must have been acquired/re-certified within the past 24 months. Recent past performance (less then two years) can be considered in lieu of recertification for organizations that have already completed an initial certification. To verify its authenticity, the DoD should conduct a verification process of its own using one of the approved evaluation methods or a small assessment team of qualified individuals who could verify the authenticity in a two- to three-day review.

The current directive requiring CMM Level 3 or equivalent for ACAT I program development teams must be strengthened. It needs to apply to primes and major sub-contractors involved in ACAT I programs, and the "escape clause," which permits submission of a risk plan in lieu of meeting CMM criteria, must be dropped.

4.3 INITIATE INDEPENDENT EXPERT REVIEWS

The Task Force found that - Independent Expert Reviews (IERs), as used by the industrial sector, were highly effective in identifying software development problems. We recommend institutionalizing such reviews on DoD ACAT I-III software-intensive programs.

The purpose of IERs is to ensure that programs are adequately addressing issues of cost, schedule, technology, risk, and process. Another is to share scarce senior technical/programmatic expertise across a boarder base of programs. IERs should establish that:

- The program team has a complete understanding of the program requirements

- Specific acceptance criteria have been established for all deliverables
- Plans are in place for major program elements including test, transition, and operations & maintenance.
- A thorough risk assessment has been performed with a risk management plan in place
- Appropriate program management and software development processes are in place.

The IER team should be a small group of professionals with the appropriate mix of experience in program and project management, software technology and software development. The team should be drawn from government, academia, and contractor resources. No IER team members should be directly involved in the program, thus enabling it to serve as a true "Non-Advocacy Review" team. A team of such experts would provide both independent vision and expertise not otherwise available to the program.

The Task Force strongly recommends that ACAT I-III Program Executive Officers (PEOs) require IERs to follow this non-advocacy review philosophy. The PEO or equivalent official should be responsible for establishing the required IERs (see Figure 4.1a). The resulting IER reports should be delivered to the program management teams.

IERs should occur at selected program events, starting prior to the release of the request for proposal and occurring one month prior to key project milestones. An IER should be held at least every six months and not be longer than 1-2 days in length. IER teams may require a half-day of training prior to the review. Example agenda and review topics are shown in Figures 4.3b and 4.3c.

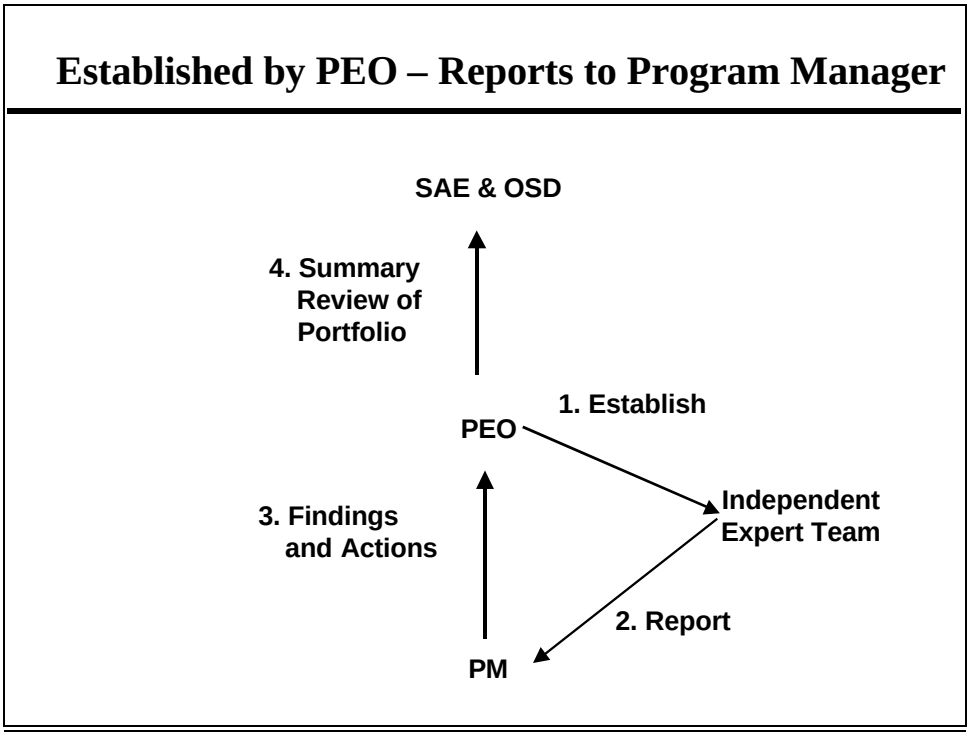


Figure 4.3a – Independent Expert Review Process

Sample Agenda for Independent Expert Review (IER)

- IER objectives, introductions
- Overview of program
- Measure of project success
- Contract overview
- Solution and approach
    - System architecture
    - Development process
- Software management plan
- Subcontractor Management Plan
- Cost/schedule/risk
- Process/tools/environment
- Software test plans
- Staffing requirements
- IER team caucus/outbrief preparation
- IER team outbriefing

Figure 4.3b – Sample Agenda for Independent Expert Review

Sample Independent Expert Review Topics

Project management

- Acquisition planning (including affordability, schedule, requirements definition, total ownership cost and technology refresh planning with cost)
- Program schedules (task activity network, and deployment schedule)
- Risk identification, assessment, and mitigation activities(technical, cost, schedule, Top 10 risk list, and other areas)
- Cost and schedule estimates(including costs associated with risk)
- Requirements development and management
- Milestone accomplishments versus plan
- Earned value (CPI, SPI, and TCPI)
- Schedule compression
- Contract incentives (award fee, share line, etc.)
- Next major milestones (plans, schedule, and risk)
- Personnel (staffing requirements versus plan, and turnover)
- System acceptance criteria
- Deployment planning
- Training requirements and status

Product construction

- Design process (model-based notation, component-based design, etc.)
- Development process (requirements, design, code/unit test, and software integration, including review process)
- Software practices and tools
- Defect-finding techniques
- Build planning and status

Product integrity

- Scenario development and requirements traceability
- Project architecture
- Test planning and progress
- Quality control techniques
- Product stability testing
- Capability-based testing

Figure 4.3c – Sample Independent Expert Review Topics

4.4 IMPROVE SOFTWARE SKILLS OF ACQUISITION AND PROGRAM MANAGEMENT

Our Task Force found that inexperience and/or unqualified personnel at all levels are a major contributor to DoD software problems. This is particularly disturbing since software frequently constitutes a majority portion of DoD program budgets and risks. DoD program managers typically have a 2-week software course at the Defense Systems Management College (DSMC) (most "opt-out" of this) and one page in the program managers handbook as a guide for software development. In comparison, pilot training is 18 months and linguist training is 14-16 months.

A range of methods can be applied to address this problem. For example, the DoD could rely upon the commercial and defense contractor sector to provide expert talent. However, we believe it is essential that the government build a cadre of managers and leaders with deep technical mastery of, and broad operational perspective on, software-intensive systems.

Our Task Force views the software problem as a war, and our warriors are qualified personnel. While we have expert system acquisition managers, we have few expert software acquisition managers (e.g., only 1 out of 92 pages of the DSMC Program Managers Handbook is devoted to software.)

If we are to establish an elite information force, the following actions should be taken:

- **Institute mandatory software-intensive systems training for program managers and key staff on all ACAT programs**
  Prior to program initiation, and then at appropriate intervals, program managers and leaders require training in key software technologies and concepts. Training should be tailored to a program's specific system software needs (e.g., real-time, security, and interoperability).Training (e.g., at the War College, DSMC) should include support from industry. Finally, it is important to update continuing professional software systems education at IRMC, DSMC and National Defense University to incorporate new concepts such as spiral/iterative development and acquisition, executable architectures and component-based development..

- **Require collaborative government/contractor team training at program start and at critical milestones**
  The partnership of government and industry expertise is essential to success. Empirical studies suggest that team training improves the likelihood of software system success. Teams based on trust can be cultivated by fostering communication and collaboration via joint training and joint work.

- **Require annual software technology refresh training for joint government/contractor teams**
  Tracking technological change is essential to preventing system obsolescence. While technology tracking and assessment should be a continuous process for technologies critical to defense systems, the Task Force recommends at least annual technology refresh training for joint government/contractor teams.

- **Develop a graduate-level program for software systems development and acquisition**
  The DoD should foster programs that create a new supply software-system personnel and/or retrain expert acquisition personnel in software-intensive systems. The DoD

should collaborate with academia to create a graduate-level program for software intensive systems development and acquisition. The program should include semesters addressing software foundations and advanced elective topics. It should also include a final semester thesis focused on issues of key concern to the DoD (see sample curriculum in Figure 4.4a). Graduates should be deployed to the hundreds of ACAT programs.

- **Require that each ACAT program office appoint an expert software systems architect**
  Given the centrality of software in modern weapon systems and the increasing need for system interoperability, a best practice is to appoint an expert software systems architect who reports directly to the program manager. The software systems architect is someone who can be the consumer's advocate by thoroughly understanding the system domain (e.g., strike aircraft, nuclear submarines, advanced tanks, etc.) and is an expert in software systems (e.g. distributed real-time computing, networking, database systems, information assurance). In collaboration with the program manager and contractors, the architect must have responsibility for software technology selections, software system properties (e.g., interoperability and performance), and software technology refresh.

Syllabus for Graduate-level Program
in Software-intensive Systems Development and Acquisition

Semester 1 – Foundations

- Software architecture
- Developing and testing software systems
- Modeling and analyzing software systems
- Managing software development
- Software Acquisition and Development Best Practices

Semester 2 - Advanced topics (electives)

- Networks and distributed systems
- Operating systems
- Database systems
- Real-time systems
- Human computer interaction
- Information assurance
- Software process
- Co-design of software/hardware

Semester 3 – Leadership

- DoD acquisition case studies
- Capstone: student team simulated acquisition/development
- Additional electives (see Semester 2)
- Thesis topics (e.g., COTS integration, E-commerce, domain specific architectures)

Figure 4.4a – Syllabus for Graduate-level Program in Software-intensive Systems Development and Acquisition

4.5 COLLECT, DISSEMINATE, AND EMPLOY BEST PRACTICES

The Task Force determined that software-intensive defense programs could substantially improve program success and reduce costs and schedule by leveraging practices used successfully in the industrial and commercial sectors.

The Task Force concluded that ACAT I-III defense software programs would benefit significantly by an intense effort to utilize the basic principles of disciplined software development with a core focus on three fundamental areas: management of the project, construction of the software product, and the integrity and robustness of the product during construction. Defense programs should implement such fundamental practices, starting with those defined by the AIRLIE Software Council of the Software Program Managers Network (see Figure 4.5a).

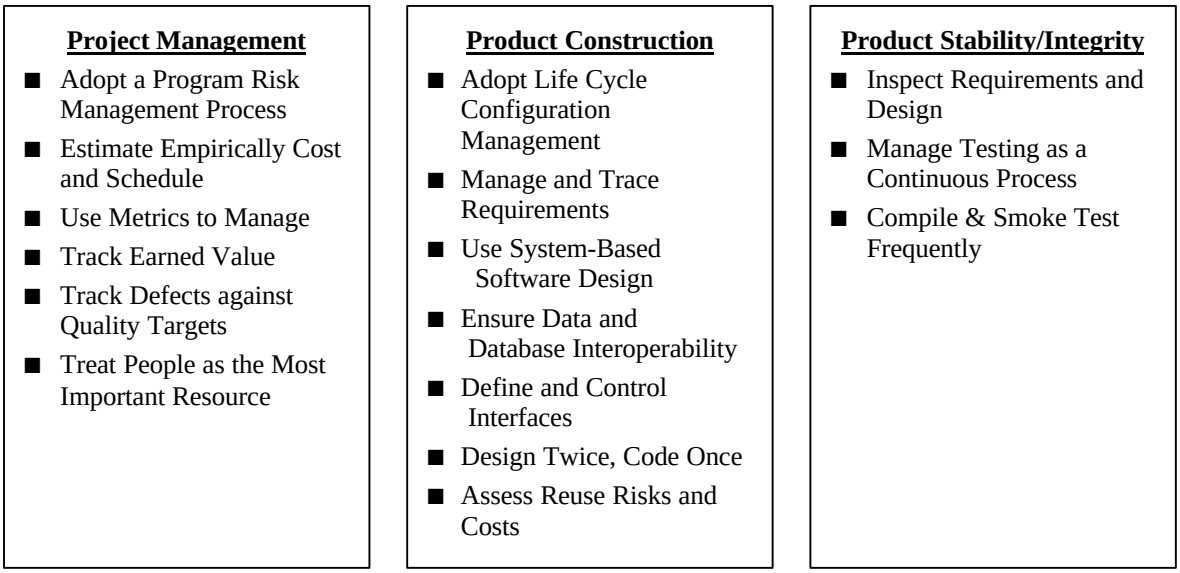


Figure 4.5a – Fundamental Practices of Software Development

In addition, the Task Force recommends implementing the following best practices:

- **Iterative processes, executable architectures.** The Task Force concluded that as software development proceeds special emphasis should be placed on the use of iterative development processes and the development of executable architectures within the context of an architecture-first approach. Iterative releases should be planned in usage scenarios with evolving levels of detail, and a demonstration-based approach should be utilized to assess intermediate artifacts. An iterative life-cycle process should be used to confront risk early and should be coupled with design methods that emphasize component-based development. Design artifacts should be captured in rigorous model-based notation. Project planning and management should be accomplished in the context of a change-management environment.
- **Limit development time.** Limit the time allowed to develop and demonstrate a software program sub-process to no more that 18 months (nominally). Programs should transition product artifacts into an executable demonstration of relevant scenarios to stimulate earlier convergence of system level integration and a more tangible understanding of design tradeoffs and an earlier elimination of architectural defects.
- **Requirements tradeoff.** The Task Force realized that software-intensive defense programs typically have insufficient time and resources to implement stated requirements. If funding is fundamentally constrained, DoD Program Managers should be provided every opportunity to explicitly trade required functionality for schedule, time, project/product stability, and risk without compromising the overall system objectives.

The Program Manger should be given the opportunity to renegotiate design requirements after a software design iteration, when the design trade-offs are better known and understood.

- **Minimize complexity.** As a guiding principle DoD software programs should aim to minimize complexity and maximize understandability of software design.
- **Establish processes, goals and decision points.** Software development teams should follow a set of processes that are based on industry best practice and these processes should be amended to permit improvements based on past performance experience. Programs should have clear goals and decision points. Decision points should be used to review progress and achievement and to determine the future direction of a program.
- **Better use of metrics.** Metrics, the Task Force concludes, are often ineffectively used by defense programs. This prevents the programs from assessing project health and progress. Although various methodologies exist for identifying of project-specific issues, certain fundamental metrics are essential to assessing software-development projects and identifying emerging problems. A best practice that all major defense software–intensive projects should adopt is core metrics collection. Core metrics are intended to supplement, not replace, whatever other program specific metrics the program manager determines useful. The core metrics are:

    - Progress
        - Earned value (planned versus Actual – Cost Performance Index, Schedule Performance Index, To Complete Performance Index)
        - Milestone slippage (aggregate slippage against plan)
        - Segment completion against plan
    - Staffing
        - Key vacancies and turnover
    - Requirements
        - Implementation coverage (percent implemented in design and test)
        - Volatility (percent change over time)
    - Quality
        - Defects (open, closed, and age profile)
        - Testing (planned versus conducted versus passed)
    - Product stability:
        - Structured peer review coverage (percent baselined products inspected)
        - Rework (corrective effort on baselined product)

- **Recognize and reward success.** The DoD should recognize the best software development in each Service and DoD activity. The annual H. Mark Grove Award for Excellence in Software Management, awarded by the Software Program Managers Network, is an example of an award that could be more effectively supported and endorsed by the DoD.

4.6 RESTRUCTURE CONTRACT INCENTIVES

While software is often the predominant cost driver and capability provider in military systems, current contracting practice fails to take sufficient steps to ensure the success of software within programs. In commercial practice, development teams are highly incentivized to create products that get to market quickly, have limited defects (e.g., excessive call rates diminish profit), and are popular. This increases profits. Competition is fierce, rewards for success are high, and penalties for failure are painful.

Many aspects of the classic DoD acquisition process degenerate into mutual distrust. This makes it very difficult to achieve a balance between requirements, schedule, and cost. A more iterative model, with a closer working relationship between customer, user, and contractor, allows tradeoffs to be made based on a more thorough understanding on all sides. This requires a competent and demanding program office with application and software expertise and a focus on: delivering a usable system (rather than blindly enforcing standards and contract terms); and allowing the contractor to make a profit with good performance. At the same time, a more iterative model requires a contractor who is focused on achieving customer satisfaction and high product quality in a business-like manner.

As part of the adversarial nature of the current acquisition process, there is considerable focus on ensuring that contractor profits are within a certain acceptable range (typically 5%-15%). Occasionally, excellent contractor performance, good value engineering, or significant reuse results in potential contractor profit margins in excess of “their acceptable initial bid”. As soon as customers (or their users or government SETA organizations) become aware of such a trend, pressure is applied to employ these “excess” resources on out-of-scope changes until the margin is back in the acceptable range. As a consequence, the simple profit motive that underlies commercial transactions and incentivizes efficiency is replaced by complex contractual incentives (and producer-consumer conflicts) that are usually sub-optimal. Contractors frequently see no economic incentive to implement major cost savings, and certainly there is little incentive to take risks that may have a large return. On the other side of the ledger, contractors can easily manage to consume large amounts of money (usually at a small profit margin) without producing results and with very little accountability for poor performance.

Our belief is that incentives as high as 30% for successful programs and corresponding disincentives (early cancellation) for failing programs can save the DoD money and foster a healthier. Good contractor performance needs to be more profitable and poor contractor performance needs to be more financially painful. In addition, we should reward effective software development teams, incentivize contractors to increase efficiency and continue to emphasize past performance and demonstrated software development proficiency in source selection. It is fundamentally important to identify and implement contract incentives (including type of contract) that have the effect of encouraging defense contractors to reduce rework and substantially improve their development and maintenance processes; the typical current contracting approach of cost-type contracting does not provide such an incentive.

Software system requirements (the problem space description) should evolve together with the software system design (the solution space description) and the overall project plan (the time and resource constraints) and be managed through aggressive contract incentives. The developer should be given the flexibility to trade requirements for time, stability, and risk as the design evolves. The DoD should manage this through interrelated contract incentives covering

functionality (e.g., feature set), cost and schedule (i.e., resources in dollars and time), and quality (e.g., performance and call rate).

Awards and penalties should be based on quantifiable, analytical results (e.g., earned-value method) as well as frequent demonstrations of progress. Incentives should be agreed upon up front by all stakeholders — developers, acquirers, and users. The entire team needs to be involved in the quest for success all along the life cycle.

Based on the above, our task force therefore recommends:

- Making award fees for success high, as high as 30%
- Developing model contract language that incorporates these incentives
- Recognizing software acquisition excellence (i.e., by annually recognizing software acquisition excellence in defense programs).

4.7 STRENGTHEN AND STABILIZE THE TECHNOLOGY BASE

This section addresses the technology aspects of a program for reducing the cost of the DoD's software life cycle, increasing DoD assurance, allowing collaborative co-evolution of requirements together with architecture, and rapidly adapting software to fluid circumstances important to National Security. For simplicity, the phrase "software development technology" is used to designate these aspects.

Improvements to process, training, incentives, and procurement are critical, and yet improvements to the process without improvements to the technology cannot address the staggering intrinsic complexity necessary for achieving and maintaining a national competitive advantage. In our recommendations the Task Force focused on automation to assist in reducing the complexity of software development. The Task Force recommendations are:

- **The DoD should leverage commercial technology, not duplicate it.**

  DoD research and development must continue to address technological advancements in areas not covered by commercial technology. The DoD must also continue to leverage commercial technological advances, since the commercial IT market is doing substantial, short-term, product-focused, commercial IT research and development.

  There are priority defense areas that either are not as important to commercial businesses, are longer term, or are not addressed by commercial technology. For example, commercial technologies do not provide the level of assurance, freedom from computer crashes, and security that is adequate for national security purposes. Commercial technology also does not provide adequate capabilities for the embedded, distributed, survivable, reconfigurable DoD systems. We discuss this research and development gap later.

  For the most part, current DoD research and development organizations do target militarily relevant problems that will not be solved commercially. These organizations also focus on tracking and leveraging commercial tools, practices, and infrastructure. They must continue to be agile and opportunistic in quickly adapting commercial technology and infrastructure when appropriate.

- **The DoD must retain key researchers.**

  Perhaps the most important problem regarding the people issues in software technology research at universities, research organizations, and defense industries is the pressure for research and development talent to depart for lucrative commercial jobs, especially in today's Internet world. And people working at the latest Internet start-up (to provide software to sell products) will not be making a contribution to the DoD software development technology base or, most likely, not even the COTS tool base.

  To compete effectively in these software job markets, DoD research organizations must offer secure, stable, and financially appropriate opportunities. It is difficult for even the most attractive universities and institutions to keep top talent when the software research funds are unpredictable. These funds must not be seen as accounts that can be adjusted frequently to meet other needs.

  For example, when funds for military operations or cost overruns are taken from research (even if given back later), the research programs are disrupted. Software research is very difficult and requires long-term commitment, education, and involvement. When researchers see the instability and decide to depart for the start-ups (possibly for good reasons such as to provide security for their families) the significant investment in their knowledge and expertise is lost to the research community. The relative risk of lucrative startups decreases when research funds are also considered risky or unstable. It can take several years for new researchers to become productive in difficult projects.

  Any industry must deal with volatility, especially the Defense Industry. But our point is that very complex software development research cannot simply scale up rapidly after having been scaled down. There are few people who have the talent to solve the most complex of software technology problems, and we must work to protect their efforts and to ensure their contributions in the future.

  For example, we note that the planned PITAC support for embedded systems research, an area of key DoD significance, was reduced by Congress from $70 million to $30 million. We believe there remain other significant research areas and people in the Software Technology area that remain inadequately funded.[7]

- **The DoD needs to address intrinsic complexity with technological solutions.**

  The development process, at the software target code level, required to maintain our national security through competitive advantage, is increasingly intrinsically more complex. As discussed earlier in this report, DoD software applications are becoming ever larger, and complexity increases more than linearly with software size. Very complex issues can occur not only at the large, system level (integration or architecture problems), but also at the critical component level (e.g. a missing case, deadlock, livelock).

  Solving the software problems of today and the future requires improvements in process and technology. There are many examples where software complexity allows errors to escape the developers, but where newer technology can uncover the problems. Some

[7] The exact numbers are not publicly available, but we believe these estimates to be very realistic, and accurately reflect the state of research funding, emphasizing the need and relevance for stronger and more stable support.

examples from the past include: on-chip software with divide errors (or on-chip cache multiprocessor cache controller software with errors) that have been uncovered with model checking, Java errors that were discovered by inference and formal specifications, and spacecraft with livelock errors that newer analysis methods can find. A well-known, relatively small and supposedly secure public key protocol was used for many years and then found to have a security problem. Other current problems are abundant. For example there is no safe solution yet for the distributed denial-of-service attacks, which most recently have crippled Internet service.

As complexity and combinatorial difficulty increases, the need for more advanced technology will increase. The gap between system complexity and our abilities is increasing, exacerbated by difficult requirements for distributed, embedded, real-time, life-critical, survivable systems.

Technology solutions can reduce both the *development* complexity and the *apparent* complexity by providing automation to tame the increasing *intrinsic* or *inherent* complexity of software. This reduction in complexity can be accomplished via smarter, higher assurance tools. These tools will fill in and guarantee more of the design detail, leaving the higher level, often more domain specific, yet simpler, requirements and architecture decisions to the human designers. This reduction in apparent complexity allows improved evolvability and adaptability at the requirements level.

The technical directions below represent our current insights about the directions to pursue.

General Approach. Automated assistance of the software development, evolution and maintenance process can be made possible through correct, abstract, reusable and evolvable software artifacts: descriptions of desired or actual software properties, such as requirements, constraints, specifications, architectures, aspects, and code.

These software artifacts can be expressed in very precise, machine-amenable forms.

Terms to denote these formal descriptions of software artifacts include design elements, design aspects, or design factors. The degree of formality of the methods and descriptions must be appropriately matched to the required assurance levels ("*appropriate-weight*" formality).

The greater the accuracy and assurance of these reusable design elements, be they architectures, composition rules, design principles, etc., the lower the errors and costs, and the greater the assurance, scalability, and reusability in the software development, maintenance and evolution processes. Greater accuracy and precision may be achieved through semantically well-defined and more formal descriptions than today's commercial tools and practices support.

Automation. The development and maintenance process involves manipulating these design elements via assured methods for composition, refinement, optimization and adaptation, using codified design principles for algorithms and data structures.

The technology for manipulating design slices must provide *automated* assistance for the assembly, specialization, and adaptation of the components to the problem at hand, while creating a complete adaptable and replayable design process record for the developed software. Thus computers themselves will assist in applying codified software knowledge, including both general design principles and specific recorded design decisions, to the development and evolution of highly complex software.

Support for Assurance, Scalability, and Evolution. It is important that these descriptions and methods cover levels of abstraction from the highest-level requirements and system architecture levels to the lowest levels of code and processor architecture.

Design elements thus function as abstract components or knowledge-level components (e.g., a particular type of architecture or a class of optimizations). Assurance, as well as scalability, of complex software is enabled through correctness of these abstract artifacts (re-used or generated) and through the correctness of the design and composition methods for these abstract components.

This codified software knowledge includes both general software knowledge and domain specific knowledge (domain models). The latter will enable automated assistance of the design knowledge for an evolving, product-line approach.

Support for assurance and certification is a critical area that needs to be addressed *proactively* during each step of the development process. After-the-fact or post-mortem complete assurance and certification methods, needed after every change, are very expensive, and can involve re-discovery of the design decisions made in the development process. While there is certainly a key place for independent testing and certification methods, the earlier that support for assurance and certification is introduced into the development process, the lower the costs.

Testing of large software systems. Tools exist for the automated testing of small software systems but are not available for large scale (i.e., multimillion lines of code) systems. There have been a few proprietary tools that address large scale systems and initial results are promising, improving defect densities by a factor of two (from 500 to 250 defects per million lines of code).

General application areas. General application areas of high relevance include very large and highly complex systems; embedded systems involving physical constraints; ubiquitous and distributed sensing, actuation and computation; secure and high assurance software; robustness despite compromised interdependent software agents.

The technology base must focus on the underpinnings, i.e., the methods and tools necessary to achieve applicability to these problems, and then use these areas as the testing grounds to demonstrate true applicability and transition to the Defense community.

Opportunity. Increases in commercial computing power are enabling two fields of research that have been perhaps too computationally intensive to address fully in the past. First, there is enough computing power today to enable increasing levels of run-time adaptation to dynamically changing situations. Second, there is enough computing power to enable work to begin on the inductive inference or automated abstraction of the software artifacts described earlier. The challenge is to mechanically infer some of these artifacts, such as design patterns, by examining large numbers of existing systems. This is a very difficult combinatorial problem that should be more approachable today.

Current Promising Technologies. DoD research has promising young and emerging software technologies that should be nurtured. Examples include scalable correct-by-construction development techniques, very large libraries of well-factored software design knowledge, scalable composition and software generation based on interacting software and physical models, specification-carrying code, decision procedures for data and algorithm design, proof-carrying code, software model checking, precise formal specifications, co-inductive reasoning, aspect-

oriented composition, provably correct and automated recovery from legacy code of architecture and defects, game theory and other innovative techniques for predicting emergent behavior of large multi-agent systems, co-evolution of requirements and architectures, adaptive software, inductive inference of software design theories, and automated synthesis of glue code for component integration. Progress in these areas is most promising for reducing or handling complexity. A recent NSA experiment using such formal, incremental derivation of implementations from specifications suggests that this approach has a promising future for high assurance software.

The following table shows, in very abbreviated form, some of the current capabilities the DoD has today in software tools and methods, what capabilities it needs, and what is the research and development gap between current capabilities and DoD needs.

Research and Development Software Strategy

Function	DoD Requirement	Current Tools	DoD Needs
Requirements	• Evolving • High Complexity • Embedded	• GUI Builders • Spiral Refinement	• Design capture and Evolution Tools • Auto Code Generation from Requirements • KB Embedded Sys Design
Architecture	• High Assurance • Predictable Performance • Process Modeling	• CASE Tools • Drawing Tools • 3-Tier Architecture	• Executable Spec Languages • KB Design Environments • Architecture Recovery • Semantically Well Defined Code
Integration/ Components	• Heterogeneous, Distributed • Secure • Massive Legacy • Interoperability	• Component Libraries • DCOM, CORBA, XML • Digital Certificates • App Service Providers	• Application Knowledge Libraries • Secure Mobile Code • Performance and "ility" critics and transformations
Implementation	• Cross Platform • Multiple Languages • 5+ M LOC	• Visual Programming • Application Generators • Domain Modeling Tools • 1-5 M LOC	• Smart Domain Modeling Tools • Collaborative Engineering • Simulation-based Design • KB software assistants
System	• Survivable • Life Critical • Distributed, Evolvable	• Fixed, 3-Tier Architectures	• Domain Ontologies/Models • Survivable and Secure Systems • Adaptive Systems
Management	• Continuous Improvement • Life Cycle Performance	• Configuration Mgmt • SEI's CMM, TSP	• Knowledge Capture • Documentation Generation • Intelligent Learning Environments. • Semantic Progress Metrics

Figure 4.6a – DOD R&D Software Strategy

Based on this analysis, the Task Force recommends that software research be focused in the following areas:

- **Architecture** Start a new program with both basic and applied research in executable specifications, software architecture recovery (e.g., reverse engineering) and knowledge-based, automated, formal, forward engineering of large-scale (~ 5 million+ LOC) software systems, allowing co-evolution of inter-consistent requirements, architecture and software.
- **Components/Integration** Initiate a new research program to accelerate development of abstract design-component based systems in addition to code-component based systems, addressing automated discovery, composition, generation, interoperability, and reuse across hundreds of systems. The effort should model and compose not only design elements or factors and components including both software and physical systems, but also the design tools themselves. The composition methods should handle both legacy components and generated components, including glue code.
- **Scalability** Initiate a basic research program, emphasizing scalable solutions, for building software/system engineering environments that support scalable design, simulation, and semi-automated generation of highly complex software systems (e.g., very large scale (~10M LOC), with real-time, embedded, distributed, life-critical, rapidly deployable, adaptive, and/or low bandwidth requirements). The program focus should include the design of semantically well-defined high-level architecture, requirements, and development domain libraries.
- **Proactive assurance** For all of these programs, support research for creating intrinsically high assurance development methods (not just after-the-fact checks) that achieve and maintain defined substantial, high levels of assurance, security, and survivability.

APPENDIX A.

Terms of Reference

ACQUISITION AND TECHNOLOGY

THE UNDER SECRETARY OF DEFENSE

3010 DEFENSE PENTAGON
WASHINGTON, DC 20301-3010

24 SEP 1999

MEMORANDUM FOR CHAIRMAN, DEFENSE SCIENCE BOARD

SUBJECT: Terms of Reference-Defense Science Board Task Force on Defense Software

You are requested to form a Defense Science Board (DSB)Task Force on Defense Software to determine the conditions under which the procurement of defense software (i.e., operational software, support software, and software tools) can appropriately use commercial practices and to develop a strategy for defense software procurement that substantially incorporates such practices. The proposed strategy should specifically address DoD use of commercial software products and determine what management practices DoD should employ for the most efficient and effective definition of, procurement of, integration and testing of, and maintenance of Defense Software. The Task Force should determine whether DoD should develop any new software tools, technologies or libraries; and if so what approach should be used to assure that such developments enter the mainstream of commercial industry.

The scope of this effort should include all DoD systems that are software intensive. It should address all stages in the life cycle of a software component from initial procurement to evolutionary upgrade of software or of software/hardware combinations. This Task Force should not be constrained by existing DoD standards nor by current DoD procurement strategy. Instead, alternatives should be considered and proposed based on merit.

To assess the utility of the recommendations, the Task Force should identify and apply objective measures such as elapsed development time, software life cycle cost, management risk, as well as measures of software product quality.

The Task Force should consider at least the following topics:

Technical: State-of-the-art and best practices, development tools, incorporation of information security/assurance techniques and practices, reusable software components, and techniques and tools for tailoring, integrating, and testing available, e.g. commercial, software components for use in defense systems.

Management: DoD management of the development process, software risk management techniques and supporting tools, minimum delivery time, affordability, maintenance after product delivery, post-deployment product enhancement, software process support tools, quality and assured availability, and use of development and maintenance tools.

The Study will be sponsored by the USD (A&T)and DDR&E. Mr. Bob Nesbit and Mr. Marc Hansen will serve as co-chairmen. LCOL David Luginbuhl, USAF, will serve as Executive Secretary; and Commander Hughes will serve as the Defense Science Board Secretariat representative.

The Task Force will operate in accordance with the provisions of P.L. 92-463, the "Federal Advisory Committee Act," and DoD Directive 5105.4, the "DOD Federal Advisory Committee Management Program." It is not anticipated that this Task Force will need to go into any "particular matters" within the meaning of section 208 of Title 18, U.S. Code, nor will it cause any member to be placed in the position of acting as a procurement official.

Jacques S. Gansler

APPENDIX B.

Briefings Provided to the Task Force

Briefings Provided To Task Force

DATE	TITLE OF BRIEFING	BRIEFER
10/13/99	Chairmen's Welcome Remarks	Mr. Bob Nesbit Mr. Marc Hansen
10/13/99	Standards of Conduct	Mr. Dave Ream
10/13/99	Terms of Reference General Guidance	Chairmen
10/13/99	Guidance	Dr. Hans Mark
10/13/99	COTs, Technology Refresh and Design to Affordability Processes	Mr. Robert McCaig
10/13/99	Crusader Software Development	Mr. Larry Yung
10/13/99	Guidance	Dr. Jacques Gansler
10/14/99	Benefits from Implementing a Process Improvement Program	Mr. David Putman
10/14/99	F-22	Mr. Ron Dubbs
10/14/99	Cobra Gemini Software Understanding	Ms. Penny Chase Mr. Ed Wingfield
11/9/99	Achieving Software Savings Briefing and Discussion	Mr. Norm Brown
11/9/99	Open Systems Joint Task Force Activities Briefing and Discussion	Col Mick Hanratty
11/10/99	DFAS Briefing and Discussion	Ms. Joanne Piper Arnette
11/10/99	Science Advisory Board task force on "Insuring Successful Implementation of Commercial Systems" Briefing and Discussion	Mr. Jeff Grant

DATE	TITLE OF BRIEFING	BRIEFER
12/8/99	DUSD (S&T) Perspective	Dr. Delores Etter
12/8/99	Microsoft Briefing and Discussion	Mr. George Spix
12/8/99	Software Evaluation Initiative Briefing and Discussion	Dr. Paul Ferguson
12/9/99	Theater Battle Management Core Systems Briefing and Discussion	Lt Col Ken Francois
12/9/99	Patriot Software Briefing and Discussion	Mr. Dean Mullis
1/12/00	Software Engineering Institute Meeting Overview	Steve Cross
1/12/00	Commercial experience with product line practice	Linda Northrop
1/12/00	Motorola	John Teresinski
1/12/00	1994 DSB Study	Larry Druffel
1/12/00	Design for upgrade	John Foreman
1/13/00	Personal and Team Software Process	Watts Humphrey
2/16/00	Challenges of Embedded Software	Dr. Janos Sztipanovits Dr. Shankar Sastry
2/16/00	Executable Architectures	Mr. Walker Royce
2/16/00	Building More Reliable Software	Mr. Brad Martin Mr. Jim Widmaier
2/17/00	Composition for Embedded Systems / Software Enabled Control	Dr. Helen Gill
2/17/00	Software Integration	Mr. Bob Olshan Mr. Don Winter Mr. Peter Lawrence

DATE	TITLE OF BRIEFING	BRIEFER
3/8/00	Report on meeting with Dr. Gansler	Mr. Bob Nesbit Mr. Marc Hansen
3/8/00	Task Force Overview	LTCOL Dave Luginbuhl, USAF
3/8/00	Draft Slides for Outbrief - Current DoD Environment Trends	
3/8/00	System Complexity	Dr. Steve Cross
3/8/00	Technology	Dr. Cordell Green
3/8/00	Human Resources	Dr. Mark Maybury
3/8/00	Performance Trends Down	Mr. Marc Hansen
3/8/00	Commercialization	Mr. Walker Royce
3/8/00		
3/8/00	HR	Dr. Mark Maybury
3/8/00	Technology	Dr. Cordell Green
3/8/00	Contracting	Mr. Michael C Dyer
3/8/00	Program Mgt	Ms. Brenda Goodwin
3/8/00	Architecture	Mr. Bob Nesbit Mr. Walker Royce
3/8/00	Previous Studies	Dr. Steve Cross
3/9/00	Architecture	Mr. Bob Nesbit Mr. Walker Royce
3/8/00		
3/9/00	Human Resources	Dr. Mark Maybury
3/9/00	Technology	Dr. Cordell Green
3/9/00	Contracting	Mr. Michael C Dyer
3/9/00	Program Management	Ms. Brenda Goodwin
3/9/00	Architecture	Mr. Bob Nesbit Mr. Walker Royce
3/9/00	Previous Studies	Dr. Steve Cross

APPENDIX C.

Task Force Membership and Advisors

Task Force Membership and Advisors

Co-Chairmen

Mr. Marcus Hansen	Lockheed Martin Corporation
Mr. Robert Nesbit	The MITRE Corporation

Members

Dr. Steve Cross	Software Engineering Institute
Mr. Michael Dyer	Lockheed Martin Corporation
Ms. Brenda Goodwin	Price Waterhouse Coopers, LLP
Dr. Cordell Green	Kestrel Institute
Dr. Anita Jones	University of Virginia
Dr. Taylor Lawrence	Northrop Grumman Corporation
Dr. Mark Maybury	The MITRE Corporation
Mr. Walker Royce	Rational Software Corporation

Government Advisors

Dr. Norm Brown	C4I, Navy
Dr. Jack Ferguson	Software Intensive Systems, DUSD (S&T)

Executive Secretary

Lt Col David Luginbuhl, USAF	Department of Energy

DSB Secretariat Representative

CDR Brian Hughes, USN	Defense Science Board