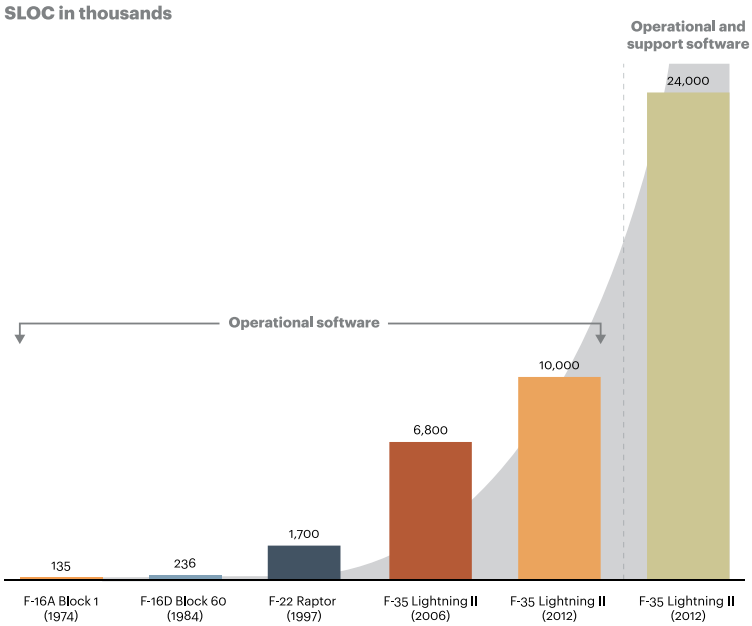
Effective Approaches for Delivering Affordable Military Software

Christian Hagen, A.T. Kearney
Steven Hurt, A.T. Kearney
Jeff Sorenson, A.T. Kearney

Abstract. As the U.S. military shifts its focus from metal and mechanics to developing and integrating control systems—unmanned vehicles, drones, and smart bombs—delivering advanced software-enabled systems affordably will become a vital success factor in weaponry.

Technology is a strategic weapon indispensable to the viability of virtually all industries and organizations. Nowhere is this truer than in the military and the DoD. The DoD is well known for its hardware-based weaponry, which is unmistakable in aircraft, satellites, missiles, and other systems in its arsenal for defending the country. Now, the military is shifting its emphasis from hardware to software. More specifically, the DoD wants to better manage the development of software that plays a growing role in national security, while improving the affordability and quality of this increasingly crucial element of modern weapon systems.

This shift will require the DoD to alter its perspective, processes, and capabilities to avoid the increasing costs associated with software development, modernization, and sustainment. Failure to make these changes will burden the country's capabilities and force the military to operate under complex and tangled systems that will cost billions of dollars to revamp. As one U.S. Air Force general explains it, "The B-52 was judged by the quality of its sheet metal; the F-35 will be judged by the quality of its software."



Notes: SLOC for F-16 and F-22 are at first operational flight. F-35 SLOC figures are from first test flight and current estimates/sources.

Sources: P. Judas and L.E. Prokop, "A historical compilation of software metrics with applicability to NASA's Orion spacecraft flight software sizing," Innovations in Systems and Software Engineering, vol. 7 issue 3, September 2011. p. 161–170; Andrea Shalal-Esa, "Pentagon focused on resolving F-35 software issues," Reuters, March 2012; Robert N. Charette, "F-35 Program Continues to Struggle with Software," IEEE Spectrum, September 2012

Figure 1: The number of SLOC has exploded in avionics software.

As advanced software systems and embedded software technologies become the brains behind modern warfare, our military is paying to develop and maintain exponentially larger systems. When measured by source lines of code (SLOC) created or modified by software developers, the amount of software code in modern war-fighting systems has increased significantly over the past decade (see figure 1).[1]

Like it or not, the DoD is now in the software business.

The Business of Software

The increased demand for software stems from an emphasis on advanced operational capabilities and requirements. For example, the airframe for advanced fighter jets has reached the limit of modern aerodynamic improvement. This limitation, coupled with recent advances in remotely piloted aircraft, places greater demands on Operational Flight Programs (OFPs) to rapidly bring new capabilities to bear. Similarly, smart weapons—such as the GPS-guided Joint Direct Attack Munition, laser-guided munitions, and AIM-9X—require continual software updates to stay smart. Full autopilot, launch accessibility region calculations, and other advanced capabilities can only be enabled using increasingly sophisticated algorithms and software architectures.

Additionally, ground systems need to overcome the challenges associated with deploying large-scale software solutions. The Army recently canceled the Future Combat Systems (FCS) program due to budget and schedule issues that were largely the result of complex software development problems. Recent reviews of the F-35 Joint Strike Fighter underscore the challenges of FCS and many other military programs, illustrating the difficulties program offices face in managing, adapting, and even comprehending software development initiatives.[2,3]

In June 2012, the Air Force Scientific Advisory Board released a report that highlighted the growing role of software in sustaining aircraft over the long term and noted that software's utility and complexity has grown faster than the Air Force's ability to address it across a system's lifecycle. So while hardware sustainment costs will decrease as the USAF reduces the number of aircraft, software sustainment costs will not follow suit because a weapon system generally needs the same software sustainment whether it is a fleet of 10 aircraft or 1,000 aircraft.[4]

With shrinking defense budgets and increasingly complex system requirements, government leaders and key contractors are struggling to deliver advanced software-enabled systems affordably. The shift from hardware to software is particularly challenging for modernizing systems and managing sustainment.

Defense System Software: Looking Ahead

The defense and software landscape is dynamic, with several technological, programmatic, and enterprise barriers, among other issues, having a major impact over the next decade (see figure 2).

As software continues to become more important to the military it will be imperative to drive savings and efficiencies into both new development and maintenance. Four areas of advantage—architecture, commercial software, should-cost analysis, and organic sustainment—are indispensable to realizing this savings without impacting the required performance and have the following objectives:

The architecture advantage. Determine the optimal architecture decisions and impact early in the design and planning processes, and continue assessments through the maintenance phase.

- Recognize that selecting the proper software architecture is a cost-effective way to rapidly improve capabilities.
- Understand the impact that modular, federated, and integrated architectures have on weapon systems and future capabilities.
- Determine which architectures allow for cost-effective enhancements.

The commercial software advantage. Upgrade capabilities, accelerate deployment, and improve total cost of ownership (in just the same way as the private sector does).

- Evaluate how and where commercial software can be deployed to use the latest capability at the lowest cost.
- Synchronize, update, or utilize refresh cycles to take advantage of the latest commercial software functionality.
- Use commercial software in both critical and non-critical systems as a replacement for costly custom development.

The should-cost analysis advantage. Perform should-cost analyses to estimate costs more accurately for the software capabilities being produced, integrated, and tested—knowing this can save millions of dollars on key projects.

- Validate contractor proposals by independently estimating the cost of software development.
- Identify improvement opportunities for software developers to reduce costs and deliver benefits to the program and taxpayers.
- Ascertain the software models and comparisons needed to get the most accurate cost estimate.

The organic sustainment advantage. Outline a clear strategy for the development of organic capabilities needed to maintain the weapon systems—and do so affordably.

- Add new capabilities to existing weapon systems to maximize asset value.
- Determine the right partnership between industrial and organic sustainment efforts.
- Build an organic capability to support next-generation weapon systems.
- Optimize investment in software maintenance and modernization.

The Architecture Advantage

How various software-enabled functions are designed and constructed within a weapon system determines the degree of effort required to enhance or modify software and deliver new capabilities. Some integrated architectures are so tightly coupled that making even small changes incurs significant costs. However, if modularity and loosely independent design are applied from the start, maintenance and enhancement efforts will be more efficient.[5] One approach to affordability promotes the use of federated avionics architectures, such as the one exhibited on recent fifth-generation fighter jets.

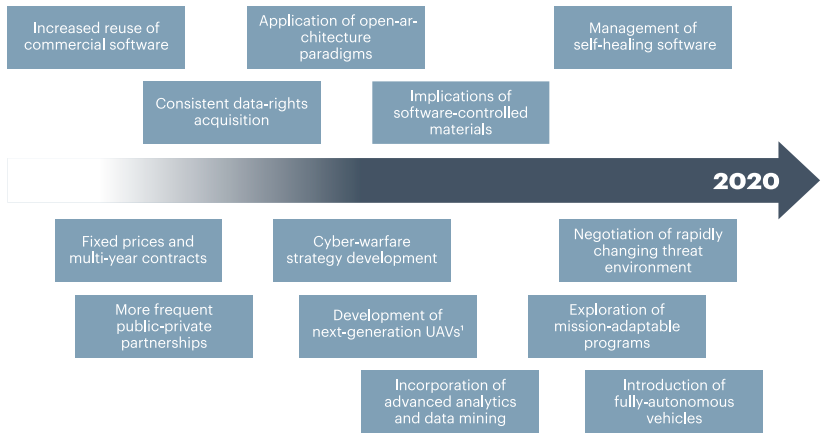


Figure 2: Issues likely to impact defense software between now and 2020.

Federated architecture. Federated architectures allow information to be shared among modules. The design is component-based and similar to that of the old legacy systems. Federated systems typically have a Central Processing Function (CPF) that coordinates information across the platform. In avionics software, the CPFs do not assume responsibility for data processing but instead organize information necessary for pilots to make decisions, while each subsystem gathers and processes data from its relevant sensors. A federated architecture delivers the same advanced capabilities and sensor fusion of highly integrated architectures but with somewhat reduced overall processing efficiency because of its modularity. However, today's modern hardware has ample capability and speed to overcome any reduction in the processing efficiency of federated architectures.

Despite decreased efficiency, modular components can be upgraded quickly and independently, thereby reducing maintenance costs. Because of the nature of designs featuring tight coupling, developing and maintaining integrated architectures across the software development lifecycle requires significantly more effort. By comparison, the overall required effort and associated costs of software designs for federated architecture are lower by a factor of two or more (see figure 3).

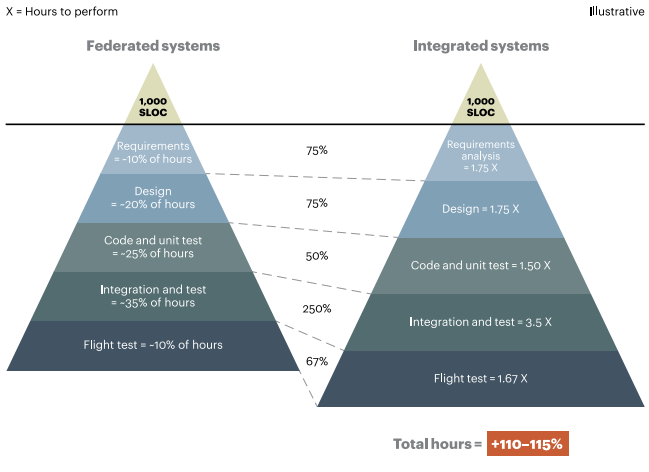


Figure 3: Integrated systems require more effort to maintain than federated systems.

Open architecture. The next significant evolution in avionics architecture will rely on the principles of open architecture, which offer an alternative to today's multitude of proprietary standards. Open architecture is a type of hardware or software architecture that allows components conforming to agreed-upon standards to be added, upgraded, and swapped. Open architecture allows independent parties to design and develop interoperable components that work together under the specified standards. By applying similar functions or skills at the platform level, greater specialization and flexibility becomes possible among developers across multiple contractors—significantly reducing the cost and effort of any software project.

Open federated architecture. A federated architecture is the natural model for open architecture development. The combination of the two paradigms (open and federated) increases the ability of development teams—and even encourages them—to modularize their code and compartmentalize their efforts.

Already, major contractors are developing their own native open architecture standards and models focused on subsystem development. However, the government should promote the development of a general open architecture, supported by all involved contractors across entire systems. Such a coordinated effort would encourage general adoption. The DoD should focus early design on open architectures while striving to achieve needed performance capabilities for the weapon systems. Great benefits can be realized by investing time early in the architecture design phase with contractor and government software professionals who will maintain the system and live in the future with today's choices.

The Commercial Software Advantage

Just as private sector companies have transitioned to commercial software and enterprise resource planning systems over the past two decades, the military is now undertaking a similar transition. When appropriate, using software developed for commercial use instead of expensive, time-consuming, custom-developed software can bring the latest capabilities to the battlefield and support systems at a fraction of the cost. Using commercial software in noncritical applications allows proven best practices to be adopted, thus avoiding problems already solved commercially.

In recent years, commercial software has become more attractive for modifying or retrofitting existing platforms for future weapon systems. Commercial software applications and processes draw from a larger pool of experience to speed up development at a lower cost. Additionally, commercial platforms offer advanced capability and increased performance over custom or embedded systems, which often rely on older technology.

As an added benefit, the extensive use of commercial software presents an opportunity to coordinate upgrades across weapons platforms by sharing common architectures, code, and maintenance efforts. Commonality across systems will allow for more centralized training of both in-house maintainers and operators, which will increase the flexibility of personnel across the services, essential in times of conflict or force reductions. Additionally, larger purchases of commercial software will offer significant negotiating power because of the supplier's near-zero marginal cost of producing or delivering additional units.

Transitioning from custom to commercial software begins by understanding their basic differences. Commercial software has four key advantages:

Faster development cycle. Commercial software is developed much faster than traditional DoD software systems, with comparable system developers being 50 to 100 percent more productive.[6] The shorter cycle allows advanced capabilities to be fielded faster and development teams to respond swiftly to requests for changes.

Advanced capabilities. Modern software development methods are changing rapidly, providing developers with a wider range of tools for coding applications. This allows for the use of the latest techniques to deliver applications that would be impossible to develop with legacy software systems.

Lower cost. Only a few software developers have proficiency with DoD custom development standards. As a result, it is expensive to retain a large number of highly experienced developers for even basic projects. Recruiting from the much larger pool of developers with commercial software experience will drive down costs to market rates and yield a more agile pool of developers.

Continual refresh. Commercial software undergoes continual maintenance, which leads to new capabilities with each software release. These upgrades are aligned with functional best practices and offer customers the benefits from global leaders that create leading and scalable functionality for end users and organizations.

Capitalizing on the benefits of commercial software requires adopting a faster, more coordinated refresh strategy. A hardware refresh that is closer to the three- to five-year cycle for commercial platforms should replace the traditionally longer defense modernization schedules, which take 10 or more years for many DoD projects. A shorter cycle ensures that large investments in war-fighting systems remain operational throughout their lifetime.

Executing this more rapid refresh cycle will require secure sources of regular funding to reduce the total cost of ownership. If funding is not provided for a full upgrade and the modification cycle lags, there will be additional costs to maintain legacy configurations. Only a stringent schedule of regular updates will manage risk and reduce overall costs.

To summarize, the use and effective integration of commercial software is not only attractive but entirely feasible for DoD and weapon systems. Improving commercial hardware and software solutions can greatly enhance capabilities, increase the speed to deployment, and significantly drive down costs. The DoD's ability to recognize this shift and manage the vendors and solutions will be crucial over the next decade and beyond.

The Should-cost Analysis Advantage

For most software development projects, estimating the required cost and effort can be challenging. Procurement and acquisition organizations—long the experts at estimating hardware-related projects, upgrades, and modernization efforts—often struggle to estimate and manage software development costs. Several authors have developed approaches and estimates for software engineering effort, project duration, and

quality.[7,8,9,10,11,12,13,14,15,16,17,18] Not surprisingly, given the complexity of software development efforts, others have offered critiques of these approaches.[19,20,21,22,23,24,25]

An inaccurate estimate can cost time, money, and lost capabilities. It is difficult to estimate software development efforts largely because of the complexity of the domain coupled with historically inadequate sizing estimates, a lack of experience with similar efforts, and poorly defined requirements. Poor estimation of software development effort and complexity is a cause of DoD program cost growth and schedule overruns.[26]

Commercial projects consistently go over budget and underdeliver on functionality. Add in defense and avionics complexities, and gauging total software costs becomes quite complex. By applying standard techniques of software effort and cost modeling, an organization can accurately estimate the required work and associated cost to avoid software development overruns. This has recently been emphasized by many senior leaders across each of the services and broader DoD.

A software should-cost review is an opportunity for everyone involved to question the traditional ways of doing business and improve efficiency across the value chain. Furthermore, when targeted to a specific program's needs, should-cost is a valuable tool for reducing costs without eroding supplier profits or cutting capabilities. It is a win-win proposition for both the DoD and its suppliers, providing a means to realize the "do-more-without-more" philosophy championed in recent years by the Office of the Secretary of Defense.[27] The analysis can improve transparency and affordability by breaking the cycle of history-based cost estimation. In doing so, it answers the overriding question: What would the costs of a program be in an efficient, highly competitive environment?

A should-cost analysis provides insight into cost drivers (for use in planning and negotiations) and forces accountability, especially across contracts and suppliers—shedding light on the overall development lifecycle. Typical should-cost software estimation includes the following four elements:

Parametric modeling. Models based on SEER for Software (SEER-SEM) and Constructive Cost Model, which rely on industry standards drawn from experience across multiple projects, help estimate the software effort and determine optimal schedules.

Bottom-up estimation. Each activity is modeled based on a discrete view of component costs against specific industry-standard benchmarks and cost drivers.

Contractor comparison. Industry benchmarks are a good way to estimate what software should cost. Examining the performance of similar contractors helps develop an understanding of best-practice performance.

Program comparison. Benchmarking similar services and common mission profiles across a comparable scope of work can help estimate costs.

Bottom-up modeling of avionics software development starts with the basic building blocks, including people (teams), processes (software development life-cycle phases), and technology (software packages), broken down into the smallest components to gain insight into the system as a whole.

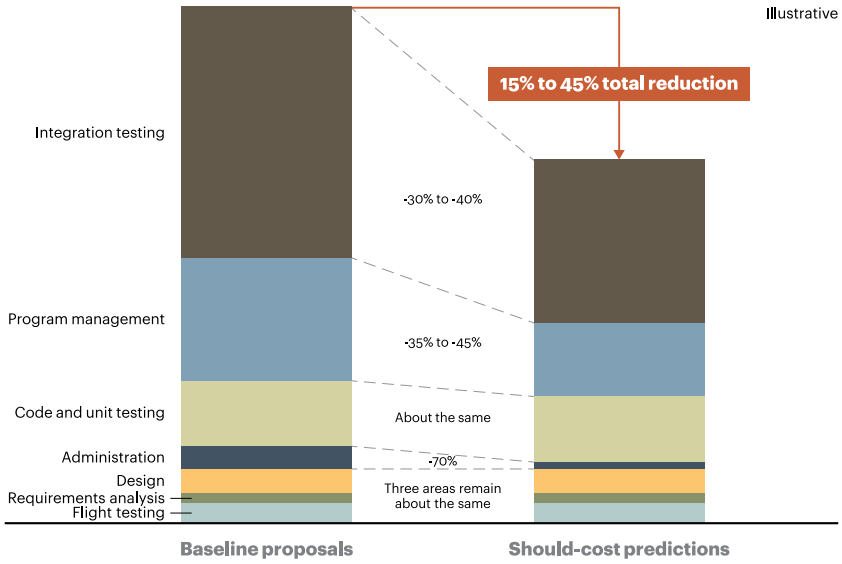
Once the system has been defined at its lowest possible level—typically down to individual programmer teams or subsystem components—cost drivers are applied against specific measurements and activities. A typical measurement might be hours required to develop equivalent source lines of code (ESLOC) for a given team. The cost drivers model the effort required for a team to develop a particular piece of software. Summing these cost estimates over each team, phase, and software package provides a full view of the effort required across people, processes, and technology.

Should-cost: A Detailed Action Plan

A should-cost analysis alone will not reduce costs; rather, the savings are achieved by incorporating the results and implementing key initiatives. A successful software should-cost analysis includes a detailed action plan with specific milestones for reevaluation and an aggressive implementation mindset to prevent the should-cost from being merely a theoretical study (see sidebar: Case Study: U.S. Air Force Avionics Should-cost).

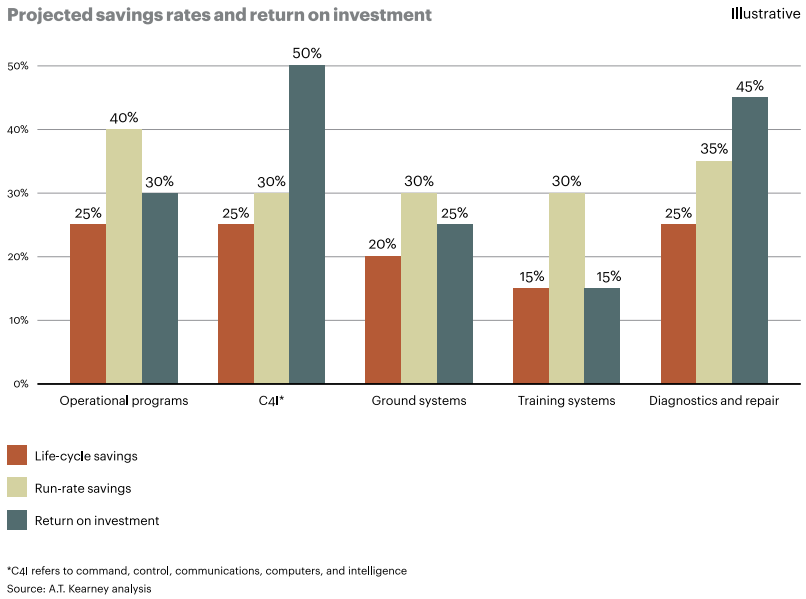
Following are five ways to ensure that a should-cost analysis delivers accelerated, maximum benefits:

Bring best practices to bear. Start the should-cost analysis with an aggressive attitude for challenging the status quo. Look outside the current paradigm for best practices to replicate in a competitive environment. Focus on determining the most efficient cost-to-deliver program requirements—not on the likelihood that these requirements will be accepted. Continually ask "what if?" and "why not?"



Source: A.T. Kearney analysis

Figure 4: Should-cost modeling identifies significant potential savings.



*C4I refers to command, control, communications, computers, and intelligence
Source: A.T. Kearney analysis

Figure 5: Organic sustainment offers considerable savings opportunities.

Perform rigorous analysis. Acquire an in-depth understanding of the root-cost drivers and efficiency potential for major areas, including supply chain, manufacturing, program management, and overhead. Detail the savings potential to support the conclusion and drive tangible action.

Establish the right incentives. Recognize that the proper incentives, benefiting both the government and its suppliers, will encourage people to move beyond the status quo and act with appropriate urgency. Use a collaborative effort to bring the best results. Set up incentives to encourage everyone to perform in a way that improves suppliers' profits while lowering government costs.

Translate opportunities into tangible action. Convert cost-management opportunities into realistic action plans with clear time lines and responsibilities. Use multiple approaches to reduce costs, including negotiation, investment, joint-process improvement, and contract restructuring.

Track performance against the cost-reduction plans. Implement a target assurance program to identify cost-reduction targets and milestones. Review progress regularly to understand performance slips and ensure that mitigation steps are in place. Make sure progress is transparent, credible, and well managed.

A should-cost method offers a tangible value proposition for avionics software development. In fact, 15 to 45 percent of the cost of avionics development can be removed from the system by shedding light on the development process, isolating opportunities at the subcomponent level, and increasing the fact base for negotiation (see figure 4). The cost savings are long-term because the improvements carry over multiple development cycles and modernization efforts within a program.[28]

The Organic Sustainment Advantage

Software maintenance is a growing portion of the post-development work needed to enhance and sustain weapons platforms. As software becomes more common in acquisitions, contractors' past sustainment efforts must give way to more cost-effective and efficient government-led sustainment. With multiple service-life extension programs in effect for legacy platforms, increased government software sustainment will free contractors to focus on modernizing to keep pace with the rapid advances in sensor and weapons technology. In most instances, the government can maintain a stable sustainment organization at a considerably lower cost than primary contractors.

In-house maintenance. The government mandates that a significant portion of weapon systems sustainment be conducted in-house to preserve a captive capability that can be called upon in times of war.[29] This recent legislatively enforced transition to organic sustainment makes sense and offers significant savings opportunities—for instance, the government can often provide lower wage and overhead costs in addition to payback periods of fewer than five years. For example, the three Software Maintenance Groups (SMXGs) across the Air Force provide services from engine software design[30] to full OFP block-cycle development on platforms that range from the F-16 to MILSTAR satellite control stations.[31] Estimates for sustainment initiatives range from 15 to 30 percent in lifecycle savings over the next two decades when compared to full contractor sustainment. Additionally, annual run-rate labor savings are estimated between 30 and 40 percent following transition for representative programs (see figure 5).

Finally, through increased use of public-private partnerships, programs can anticipate further benefits to core hour requirements and capability targets while reducing risk through extended contractor support.

Organic resources are typically capable of conducting much more sophisticated sustainment activities than is often pre-

sumed. The DoD should undertake capability studies to evaluate the potential of the existing organic resources and organizational readiness to accept future workloads. Modernization and software upgrade schedules should be aligned and the result embedded into a capability road map to govern overall schedules and prioritization.

Data rights acquisition. To enable in-house software maintenance, appropriate government rights to the data are critical. Funding for data rights acquisition should be earmarked as an essential expenditure. It may be necessary to fund acquisition independently to prevent potential mingling or re-appropriation that favors imminent challenges over long-term needs. Furthermore, in-house training on the use and relevance of software data rights across program offices and the DoD will ensure that all data and data formats acquired are usable throughout the system's life.

Necessary skill sets. Several key skills will be needed over the next decade. Advanced work in OFPs, command and control systems (C4I), and advanced ground control stations will augment and replace workloads currently in test stand and control, and verification and validation. Software architecture and engineering skills will be required earlier in the software development lifecycle. Software sustainment will inevitably include ongoing enhancement, which requires early participation in the design process. New skills in requirements analysis, functional design, and software architecture will be required to field these new capabilities. Internally developing and supporting this work will be vital to the ongoing success of organic software sustainment organizations.

By developing a targeted transformation road map to grow the organic sustainment organization and capabilities, the DoD can meet key software maintenance goals that will drive next-generation weapon systems. The organic growth road map should prioritize the transition of avionics and C4I programs to preserve knowledge of the most important and fastest-changing weapon system software. Transitioning programs may result in a 30 to 40 percent run-rate savings following a transition to an 80 percent organic capability.

Transformation Is Key to Winning the Race

A radical reshaping of software policies and practices will help the U.S. military avoid falling behind in this arena. From remotely piloted aircraft and smart bombs to autonomous vehicles and advanced fighter jets, software is crucial to the success of today's weapon systems. Focusing solely on developing and maintaining military hardware is no longer an option. With shrinking defense budgets and increasingly complex systems, the defense industry and services must fight to deliver modern software-powered weapons affordably. To deliver on this ambitious objective, the military must drastically transform its approach to software. New organizational structures, operating models, and tools will be essential to modernizing and sustaining the U.S. weapon systems. The mission is to deliver benefits today while keeping pace in the race to the future—or risk losing it.

Case Study

U.S. Air Force Avionics Should-cost

Aerospace engineers and designers have pushed the bounds of the possible with the mechanical design and construction of military aircraft, creating machines that are both breathtakingly beautiful and devastatingly effective. Now, software developers are pushing the limits of traditional avionics in modern systems to keep older aircraft competitive and create innovative capabilities in the new ones.

On the F-22 increment 3.2A program, according to a recent DoD Better Buying Power fact sheet, the Air Force successfully identified and implemented cost-saving initiatives of 15 percent (equaling $32 million) to address areas in the software development process that were above industry benchmarks.

Approach

The analysis involved the processes and costs of a proposed half-billion-dollar modernization effort for the Air Force fleet. This software-only modernization effort involved adding several new tactical capabilities to ensure the success of current and future missions.

The project began with an extensive evaluation. In interviews with the primary contractors responsible for developing the modernization software, the team analyzed the development processes, lifecycle, estimation techniques, system components, software development roles, and unique system components involved in an enhancement effort.

From the data collected, the team created a component-based model of the system, down to the smallest development effort. The software effort and cost was evaluated based on the components, development process used, historical productivity, and technology platforms. The result was a complete should-cost model of the entire development effort—from the top down to the subsystem components in terms of people, processes, and technology.

The final model estimated the timing (in hours) for each phase of the software development lifecycle. It created transparency into the development process, targeted DoD areas for improvement, and streamlined cost negotiations for the modernization effort.

Results

The should-cost analysis identified a 15 to 30 percent gap between the current software development process and a targeted should-cost approach. The gap was further substantiated through a parametric model using standard software industry tools such as SEER and COCOMO.

From interviews and data audits, the team identified improvement areas, such as contractor handoffs and inefficient testing during the integration phase, and created action items to reduce hours and costs, achieving the goals set out by the should-cost analysis. A collaborative effort between the military and its contractors successfully improved capabilities at a reduced price—providing value to all parties, including taxpayers.

ABOUT THE AUTHORS

Christian Hagen is a Partner in A.T. Kearney's Strategic Information Technology Practice and is based in Chicago. He advises many of the world's largest organizations across multiple industries, including government and defense contractors. He specializes in helping clients leverage software and information technology to increase efficiencies and gain competitive advantage. Christian has led several global studies for A.T. Kearney and authored nearly 50 published papers on low-cost competition, software engineering, e-commerce, technology innovation, and strategy.

E-mail: christian.hagen@atkearney.com

Steven Hurt is a Partner in A.T. Kearney's Public Sector and Defense Services. Steve has worked with several of the USAF's highest visibility programs to drive affordability in both software and hardware sustainment. Specifically, Steve has focused on should-cost analyses, business case analyses, contract negotiations, and developing business intelligence aimed at reducing cost.

E-mail: steven.hurt@atkearney.com

Jeff Sorenson is a Partner in A.T. Kearney's Public Sector and Aerospace Defense Practice based in Washington, D.C. Jeff retired as a Lieutenant General with more than 20 years of acquisition experience. He successfully developed, procured, and delivered over 30 different military systems including battlefield intelligence automation, night vision, and tactical missile systems. As the Army's Chief Information Officer (CIO)/G6, he transformed Army network information technology capabilities to enhance business and war-fighting command, control, and communication systems.

E-mail: jeff.sorenson@atkearney.com

REFERENCES

1. SLOC is used here as the best available metric across DoD programs for estimating the required software development and maintenance effort.
2. United States Government Accountability Office, "Joint Strike Fighter: Restructuring Places Program on Firmer Footing, but Progress Still Lags." GAO-11-325, April 2011; p. 29-31, <http://www.gao.gov/new.items/d11325.pdf>.
3. United States Government Accountability Office, "Joint Strike Fighter: DOD Actions Needed to Further Enhance Restructuring and Address Affordability Risks." GAO-12-437, June 2012; p. 18-20, <http://gao.gov/assets/600/591608.pdf>.
4. Starosta, G. "AFSAB Urges Strategic Shift In Air Force's Long-Term Sustainment Plans." InsideDefense.com, June 2012; <http://insidedefense.com/Inside-the-Air-Force/Inside-the-Air-Force-06/15/2012/afsab-urges-strategic-shift-in-air-forces-long-term-sustainment-plans/menu-id-82.html>.
5. Modularity is the degree to which a system is made up of independent units that can be combined into a larger application.
6. Jones, C. Applied Software Measurement, McGraw-Hill, 2008, Table 3-1, p. 189.
7. Stark, G. (2011). "A Comparison of Parametric Software Estimation Models Using Real Project Data." CrossTalk – The Journal of Defense Software Engineering. January 2011.
8. Boehm, B. (1981). Software Engineering Econo mics. Englewood Cliffs, N.J., Prentice Hall.
9. Boehm, B. (2006). "Minimizing Future Guesswork in Estimating," IBM Conference on Estimation, Atlanta, GA, Feb. 2006.
10. Jones, C. (2007). "Software Estimating Rules-of-Thumb," <http://www.compaid.com/caiinternet/ezine/capers-rules.pdf>, Mar. 2007.
11. Jones, C. (1997). Applied Software Measurement, 2nd Ed., McGraw-Hill, NY.
12. Rone, K. et al. (1994). "The Matrix Method of Software Project Estimation", proceedings of the Dual-Use Space Technology Conference, NASA Johnson Space Center, Houston, TX, Feb.
13. J.W. Bailey and V.R. Basili. "A Meta-Model for Software Development and Resource Expenditures," Proceedings of the 5th International Conference on Software Engineering. New York: Institute of Electrical and Electronic Engineers, 1983.
14. ISBSG, International Software Benchmarking Standards Group, <http://www.compaid.com/caiInternet/ezine/ISBSGestimation.pdf>.
15. ISBSG, International Software Benchmarking Standards Group, <http://www.isbsg.org/isbsg.nsf/weben/Project%20Duration>.
16. McConnell, S. (2006). Software Estimation: Demystifying the Black Art, Redmond, WA, Microsoft Press.
17. International Function Point User's Group (IFPUG), Function Point Counting Manual, Release 3.1, 1990.
18. Jones, C., "Achieving Excellence in Software Engineering," presentation to IBM Software Engineering Group, March, 2006.
19. P. Oman, "Automated Software Quality Models in Industry," Proceedings of the Eighth Annual Oregon Workshop on Software Metrics (May 11-13, Coeur d'Alene, ID), 1997.
20. G. Atkinson, J. Hagemeister, P. Oman & A. Baburaj, "Directing Software Development Projects with Product Measures," Proceedings of the Fifth International Software Metrics Symposium (Nov. 20-21, Bethesda, MD), IEEE CS Press, Los Alamitos, CA, 1998, pp. 193-204.
21. T. Pearse, T. Freeman, & P. Oman, "Using Metrics to Manage the End-Game of a Software Project," Proceedings of the Sixth International Software Metrics Symposium (Nov. 4-6, Boca Raton, FL), IEEE CS Press, Los Alamitos, CA, 1999, pp. 207-215.
22. Kemerer, C. F. (1987), "An empirical validation of software cost estimation models," Communications of the ACM, Vol 30, No 5, pp. 416-429.
23. Jorgensen, M., and Sheppard, M. (2007), "A Systematic Review of Software Development Cost Estimation Studies," IEEE Transactions on Software Engineering, Vol 33, No 1, Jan. pp 33-53.
24. Fenton N. E., and Pfleeger, S. L. (1997), Software Metrics: A Rigorous & Practical Approach, 2nd Ed., London, PWS Publishing.
25. Jorgensen, M. (2004), "A Review of Studies on Expert Estimation of Software Development Effort," Journal of Systems & Software, Vol 70, No. 1-2, pp. 37-60.
26. Lucero, Scott. "Software Sustainment Challenges in Defense Acquisition." Office of the Deputy Undersecretary of Defense: Acquisition and Technology. April 2009. p. 3. Accessed 30 May 2012. <http://www.acq.osd.mil/se/webinars/2009-04-21-SECIE-SW-Sustainment-Lucero-brief.pdf>.
27. Carter, Ash. "Memorandum for Acquisition Professionals," Office of the Undersecretary of Defense: Acquisition, Technology, and Logistics. 14 September 2010, p. 1.
28. For more on how should-cost analysis can improve affordability, see "Should-cost Review: A Pragmatic Approach to Affordability," at <http://www.atkearney.com/index.php/Publications/should-cost-review-a-pragmatic-approach-to-affordability.html>.
29. United States Code, Title 10, Chapter 146, Section 2464, "Armed Forces: Core Logistics Capabilities," p. 1447, <http://uscode.house.gov/pdf/2010/2010usc10.pdf>.
30. Schroeder, Brian. "76th SMXG – More than just code writers," Office of Public Affairs: Tinker Air Force Base. 15 August, 2011. <http://www.tinker.af.mil/news/story.asp?id=123268153>.
31. 309th Software Maintenance Group. "Software and Hardware Engineering Solutions!" Ogden Air Logistics Center: Hill Air Force Base. Accessed 30 May, 2012. <http://www.309smxg.hill.af.mil/brochure/309SMXGbrochure.pdf>.