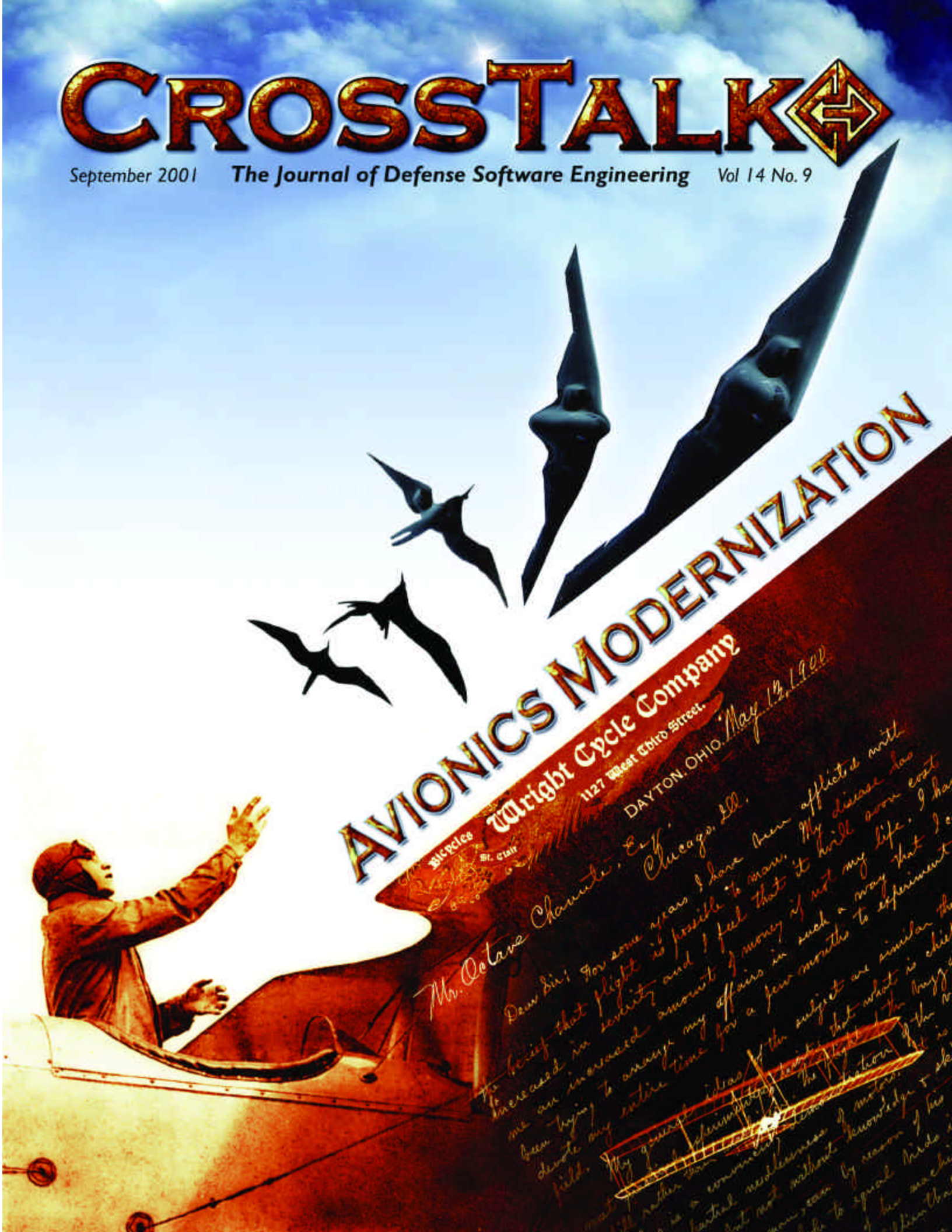
CROSSTALK

September 2001 The Journal of Defense Software Engineering Vol 14 No. 9



Avionics Modernization

4 The Air Force Develops an Initiative to Manage Change in Avionics Systems
In this CROSSTALK interview, Jon Ogg, director of Engineering and Technical Management Directorate, talks about the agency's plans for designing avionics systems that preclude their obsolescence.
by Pamela Bowers

8 Integrated Road Maps Route the Migration to Avionics Open Systems
David G. "Butch" Ardis, technical adviser for avionics systems architecture, talks to CROSSTALK about developing and implementing avionics road maps.
by Pamela Bowers

10 Customizing the Software Process to Support Avionics Systems Enhancements
Here is a look at how different software development approaches and techniques, usually applied in isolation, can be selected, customized, and combined to better meet organizational needs.
by Dr. Paolo Donzelli and Roberto Marozza

15 The Challenges of Software Certification
While the guidance on airborne software certification is mature, the issues with software reuse, military avionics certification, ground-based software, and object-oriented technology are still evolving. This article looks at how these challenges affect the safety critical community.
by George Romanski

19 Avionics Modernization and the C-130J Software Factory
This article presents insight into Lockheed Martin's modernization of the C-130 airlifter family from a largely mechanical aircraft to a software intensive system.
by Richard Conn, Stephen Traub, and Steven Chung

ON THE COVER

Kent Bingham, Digital Illustration and Design, is a self-taught graphic artist/designer who freelances print and Web design projects.

Best Practices

25 Practical Software Measurement, Performance-Based Earned Value
The Northrop Grumman team shares its experience with successful project management that focuses on requirements, selecting the most effective software metrics, and using earned value management.
by Paul Solomon

Departments

3 From the Publisher

14 Coming Events

18 Call for Articles

24 Web Sites

30 STC 2002 Call for Exhibitors

31 BACKTALK

CROSSTALK

SPONSOR	Lt. Col. Glenn A. Palmer
PUBLISHER	Tracy Stauder
ASSOCIATE PUBLISHER	Elizabeth Starrett
MANAGING EDITOR	Pam Bowers
ASSOCIATE EDITOR	Benjamin Facer
ARTICLE COORDINATOR	Nicole Kentta
CREATIVE SERVICES COORDINATOR	Janna Kay Jensen
PHONE	(801) 586-0095
FAX	(801) 777-5633
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk/crostalk.html
CRSIP ONLINE	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 29.

Ogden ALC/TISE
7278 Fourth St.
Hill AFB, UT 84056-5205

Article Submissions We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CROSSTALK.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Software Community: Ready for the Challenges of Avionics Upgrade?

One of the major challenges facing the U.S. military is keeping an ever aging fleet of aircraft ready for the fight. While systems remain in the inventory longer, the future vision of warfare demands more information exchange, and integration of more sophisticated weapons. Dynamic updates to mission plans, increased exchange of data from command and control platforms (e.g., Airborne Waring and Control System) and tactical aircraft is on the horizon. This results in increased avionics complexity and system interdependency. At the same time the military is grappling with how to handle hardware obsolescence, which in turn drives software changes. This month's CROSSTALK examines the impact on software and the contributions software technology is making to solve the problems. Collectively the articles examine top-level strategy, project planning, software development, and project control.

Jon Ogg, director, Engineering and Technical Management Directorate, Aeronautical Systems Center (ASC) provides an interview on the challenges for avionics modernization and the Air Force's approach to the technology insertion and developing affordable upgrade paths. David G. "Butch" Ardis, ASC's technical director for Avionics, then explains the Air Force's route to migrating toward more open systems, the role of system road maps, and some of the technical challenges. He highlights the importance of improving simulation and support tools for test and evaluation of software, and outlines the Air Force's approach for developing crosscutting solutions and synergy between platforms.

In *Customizing the Software Process to Support Avionics Systems Enhancement*, Roberto Marozza and Dr. Paolo Donzelli show how different approaches such as waterfall, Rapid Application Development, and incremental strategies can be combined and customized to satisfy customer goals within resource constraints.

In *Challenges of Software Certification*, George Romanski examines how software reuse, object oriented technology, and legacy military avionics systems impact the certification process. The Federal Aviation Administration's (FAA's) use of DO-178B as a standard for development and verification of airborne avionics systems and the application to military transport aircraft is an important issue for a number of programs. Incorporating Global Air Traffic Management (GATM), a concept for satellite-based communication, navigation, surveillance, and air traffic management is incorporated into an existing military aircraft is an example of why this is important to the Department of Defense. The FAA and the International Civil Aviation Organization, a special agency of the United Nations, established GATM standards in order to keep air travel safe and effective in increasingly crowded worldwide air space. In most cases, FAA certification was not accomplished on the existing systems, so decisions on the scope of certification activities and the extent of testing became critical.

Lockheed Martin's Richard Conn, Stephen Traub, and Steve Chung reveal their experiences in *Avionics Modernization and the C-130J Software Factory*. The article illustrates the evolution of avionics and how requirements are driving increased complexity and the size of software development activities. The software team's challenges of reuse, process improvement, certification (Capability Maturity Model®, ISO, and the FAA), and culture shift are chronicled.

In *Practical Software Measurement, Performance–Based Earned Value*, Paul Solomon of Northrop Grumman Corporation, discusses his organization's performance measurement techniques.

Due to an overwhelming response of article submittals for this issue, the planned theme for December is "Software Legacy Systems." The issue will address techniques for incremental upgrade of legacy systems, software reengineering, and software emulation among other topics. While the challenges for upgrade of military systems are formidable, it is encouraging to see the innovative response of the software community.

Glenn A. Palmer

Lt. Col. Glenn A. Palmer
Director, Computer Resources Support Improvement Program

The Air Force Develops an Initiative to Manage Change in Avionics Systems

Pamela Bowers
CrossTalk

While the U.S. Air Force (USAF) systems are remaining in service longer than planned, the industrial base of prime and original equipment manufacturers and suppliers is declining. Thus the USAF is faced with developing a plan for designing avionics systems that precluded their obsolescence. Weapons in the field need to remain current and supportable. Future systems require a design that facilitates substitution of modern electronics over a system's life. As a result, the Viable Combat Avionics initiative was developed.

ONE on ONE

Earlier this year, Lt. Col. Glenn A. Palmer, program director, Computer Resources Support Improvement Program, and Pam Bowers, CrossTalk managing editor, met with Jon Ogg at Wright-Patterson Air Force Base to discuss the initiative. Ogg is a member of the Senior Executive Service and director, Engineering and Technical Management Directorate, Aeronautical Systems Center (ASC), Air Force Materiel Command, Dayton, Ohio. He provides overall management guidance for the development of systems engineering programs for ASC with annual expenditures of more than $10 billion. He ensures the proper allocation and expenditure of fiscal and personnel resources and provides engineering tools to the program offices.

Ogg entered federal civil service as a project engineer with the Flight Systems Directorate in 1975. He is recognized as the Air Force's leading authority on integrity for programs propulsion and power systems. He spent 15 years in propulsion and has been involved with every phase of a system's life cycle on nearly all gas turbine engines in the Air Force inventory. In addition, Ogg has provided technical and programmatic support to many ASC weapon system programs, including as chief engineer for the F-22 Program for nearly a decade. He has led numerous reviews spanning acquisition strategies, request for proposal preparation, independent cost estimates, technical risk assignments, and flight certification. He helped pioneer the current integrated product process development and product team approach on the F-22 program.

Q: Why has avionics modernization recently gained so much attention in the Air Force?

Ogg: The catalyst for focusing a laser beam on the subject of aging avionics came about in the mid- to late-1990s when it became apparent that the diminishing manufacturing sources (DMS) and out-of-production parts were driving costs to programs. We began to see a subsystem development program nested within a development program driven by part obsolescence.

A classic example was the F-22. Three years into its development we determined the current avionics suite was not going to be producible due to rapidly changing technology, and key suppliers getting out of the business, including Intel. We never thought they would stop producing the I-960 for the F-22. They had committed to long-term plans back in the late 1980s. However in the 1990s, Intel said that the processor technology had far superceded the 25-Megahertz I-960 processor; the commercial industry was out with 200- to 300-Megahertz processors. Our [Department of Defense] market was no longer economically attractive to them.

That triggered a lot of interest at the configuration management level across the Air Force. It became necessary to fund $1 billion into the ongoing F-22 research and development and planned production program to address the out-of-production electronics and avionics.

The event that kicked us into high gear happened in the fall of 1998. At a quarterly acquisition program review the chief of staff tasked the Air Force Materiel Command to study the design of avionics systems to preclude obsolescence.

Well, the irrefutable fact is that you can't preclude obsolescence. You have to figure out how you're going to manage it. In spring 1999, Gen. (Ret.) Skantz wrote on "Aging Avionics Systems" about aging aircraft, especially avionics and the enormous future cost of sustaining these systems, if we did not move to adopt more commercial technologies and practices. Simply put, we needed to do business differently. So from mid-1999 through December 1999, we brought industry in to help put together an approach called Affordable Combat Avionics (now called Viable Combat Avionics) to look at the effectiveness of the architectures while identifying actions necessary to set a course for the future. We found that the more viable or open the architecture, the less costly it was to produce, sustain, or upgrade. (After all, the bottom line is still dollars.)

Q: Why does it cost so much to migrate to new hardware considering that electronics technology has decreased from five-year cycles to one year or less?

Ogg: Many of the current architectures are unique and make software dependent on hardware. So when hardware changes, you have to redo software at an enormous cost.

Today there is a big push on open systems and to insulate or isolate the hardware from the functional/program software. At some future point, the hardware component technology will change. Open systems minimize the dependency of exe-

cuting software on the underpinning hardware. The focus is on making the system more adaptable to future change.

In addition to the F-22 standing out as an example of this problem, we had the F-15, F-16, B-1, C-5, and C-130 – multi-billion-dollar programs – all slated for modernization. The end-user [warfighter] wanted enhanced capabilities and functionality that couldn't be accommodated with existing avionic architectures. So we were faced with modernization that typically spans four to six years due to the need to rebuild existing software for hardware technology that was out of production.

Most of our fielded systems were developed back in the days when we could influence the electronics industry. We had a large portion of their business; today, we're less than 1 percent. We are now faced with having to jump onto their track. Instead of being a leader in the electronics industry, we're a follower. In all cases as we modernize our fielded systems, we're focused on looking at the commercial world; where they are today, and, more importantly, forecasting where they are going in the future.

Clearly it is in our best interest to structure architecture to accommodate the inevitable changes over the course of time. From year to year you'll see technologies changing to provide better capability as well as lower cost. If you can't accommodate it within your architecture, you drive up the development costs and in turn the overall Total Ownership Cost (TOC). That's something we are obviously working to avoid.

Q: Is there an overarching strategy for approaching avionics modernization?

Ogg: ASC was tapped to head up the avionics modernization initiative in the summer of 1999 because we have product line management responsibilities for a majority of the aircraft systems. We sponsored a forum that included industry, multiple programs, and folks at the depots. Here's what we saw as the challenge: Where do the responsibility lines fall: which is industry, and which is us? How should we deal with obsolescence and diminishing manufacturing sources? How do we make it transparent to the ultimate customer, the warfighter?

Modernization of the C-5 and C-130 aircraft constitutes a multi-billion dollar investment. In the current environment, there is no common approach across platforms to leverage any specific architectural definition. Every program has a unique strategy. Furthermore, modernization is done, i.e., funded to provide enhanced capability. Using this approach, the move to an open system must tie into changing the architecture for providing the enhanced capability thereby making it more supportable as a by-product of the enhanced capability funded upgrade. In the past, we have gone to the warfighters to say, "Here's a great investment opportunity. If you spend a dollar today to migrate to an open system, you'll realize a 5-to-1 return during the course of the next three to five years through savings accrued across production and/or support." However, modernization dollars are so limited that the investment does not occur even though the status quo means paying out more in the long run. Colors of monies, i.e., 3600 (Development), 3010 (Production), and 3400 (Support) create major impediments to prudent investment in reducing TOC.

This is where I see the acquisition community taking the lead with our industry partners. Industry can make the up-front investment, and based upon a negotiated share of production or support savings during a specified period of time, the contractors accrue a return on their investment. We structure it over a reasonable period of time so that it represents an attractive investment opportunity for industry. Furthermore, if contractors can drive costs down even further, I say go for it. If, as a result, they realize a higher return, great. We [government] will benefit when we renegotiate the production or support contract in three years or so.

Our solution looks at how we can leverage the total modernization costs across multiple programs. While it may cost a little more to do the up-front work, all the programs on the upgrade platter benefit, and the net result is a savings to the government – the taxpayers. However, it's still tough to get the folks' support because it's likely to cost a particular program more for the initial investment.

It is also important that we look at the second- and third-tier suppliers as the principal folks to do the long-haul support as technology changes. If they are motivated by having a stake in supporting the [sub]system, they'll make the right decisions up-front because they stand to share in the downstream benefits. With open systems you can still have competition at the appropriate level. If contractor X owns a chunk of the avionics suite at the first-tier supplier level, I believe we ought to stay with contractor X. What we ought to mandate is that they maintain at least two or three suppliers for the piece parts. Contractor X is most likely a design/integration/assembly house, but the bulk of components are supplied by second- and third-tier suppliers. When told they will be involved in sustainment, we find a lot of original equipment manufacturers, without customer intervention, determining smart ways to select an architecture that minimizes downstream impacts [costs] as technology changes over time.

> **"A year ago we went to the chief of staff with the need to recognize this [avionics modernization] as a new paradigm. You've got to look across platforms. Without that support, programs will continue to operate as they always have even though it's not the most cost effective and efficient way to do business."**

Q: Where does the avionics modernization program stand at this point in time?

Ogg: In January 2000 we went to headquarters to present a plan for a viable combat avionics architecture template. This was a direct response to the tasking action item from fall of 1998. This is what we captured as the team's charter:

- Present a plan to study the design of avionics systems to preclude their obsolescence. This includes weapons in the field and how to keep them current and fully supportable, and new systems with a design strategy that

facilitates substitution of modern electronics during a system's life.
- Plan how to manage the ever-accelerating rate of change in avionics systems.

We put forth a solution focused on defining a future operating state for weapons systems in the field in terms of an architecture based on open-systems principles (see Figure 1). We advocated an integrated strategy that 1) supports evolutionary acquisition, 2) utilizes integrated change road maps, and 3) promotes designing for affordable change. It requires an institutionalized and evolutionary acquisition program that invests every two to three years. Insurmountable? I think not. Difficult? You bet!

We recognized up front that the primary way money is put into programs today is for increased capability. Warfighters budget and fund to provide enhanced capabilities, therefore we have to leverage this investment in migrating our systems to a more open/affordable architecture.

For future systems, again we proposed using open systems and master plans that advance evolutionary acquisition with an investment about every two or three years. In support of this thrust, we advocated getting the enabling language [requirements] into our solicitations. We used this approach for the first time with the C-130 Avionics Modernization Program solicitations. We required a description of the architecture with test cases. This included scenarios where in a few years certain parts were postulated as going out of production. We asked the offerors to address what it would take to address that situation in the form of time and cost? Or, five years out the warfighter will want a new capability. Tell us how your architecture accommodates providing it, and what the cost will be for development and implementation.

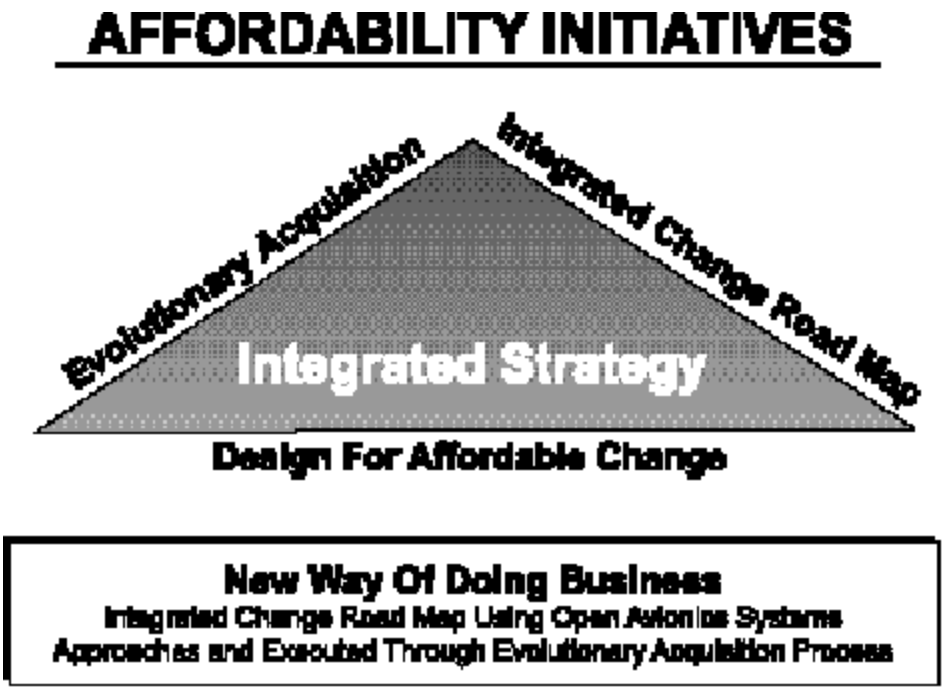
Q: What is necessary to achieving your avionics modernization plan?

Ogg: What we're trying to do is maximize common areas, which include modernization focused upon providing the warfighter with more capability, the ever present out-of-production parts, and inserting new or emerging technologies aimed at reducing the cost of ownership. Some continuing factors plague us: stovepipe thinking (every program does their own thing), avionics complexity and the dependency within systems and subsystems, interdependency across systems, the rapid pace of changing technology, and the fact that we are no longer drive this technology train. And, clearly, the budget constraints that face us all. Our solution was a three-pronged approach:
- Embrace evolutionary acquisition. Accept the fact that every two to three years you're going to have a change. The technology is going to change, and you'll need, or better want to accommodate it at a minimum cost [impact].
- Prepare avionics road maps. Factor in the modernization effort, your technology refresh, and the fact that about every two or three years you'll be faced with DMS problems. Develop and integrate these road maps, taking into account like efforts across multiple platforms. Be a realist and tie as many of these parallel thrusts into modernization [capability enhancement] since this is the principal source of funding. Make these synergistic focused on providing the capability while driving down the future development, production, and sustainment costs, or in short, reducing TOC.
- Design for affordable change by migrating to open systems. Take advantage of open systems' set of protocol and interfaces.

When we presented these needs to the chief of staff in January 2000, he responded with top-down support by issuing interim policy. It explained simply that open systems meant designing for ease of change, which is the ability to accommodate constant turnover of the underpinning technology every few years and to do so in a manner that is affordable. He charged the warfighter/acquisition/sustainment communities with taking lead in migrating the Air Force's weapon systems to a more open/viable/supportable architecture.

To support and gain momentum for our activities, we held up the Aging Avionics office with a focus of providing support to all weapons systems as they embark on this journey. We are surveying other services' and agencies' to gain an appreciation of how they are coping with this challenge. We are educating the warfighters' on the magnitude of this future bill to pay while working to gain their support for our initiatives. And lastly, but most importantly, we are developing language for solicitations focused on open-system principles. With this as a backdrop, we sponsored two affordability studies: the Boeing Open Avionics Systems Integration Study (OASIS), and Lockheed's Systems, Technologies, Architectures & Acquisition Reform (STAAR) program. Both studies are looking at opportunities to capitalize on like improvement efforts across platforms, quantifying the potential savings possible in leveraging these efforts across modernization programs.

Figure 1: *Affordability Initiatives in Open Avionics Systems*



Q: Have you had the opportunity to use the integrated change road maps to assess and grade the current state of the Air Force's architecture?

Ogg: Previously David G. "Butch" Ardis, technical adviser for Avionics Systems Architecture, and the program offices had initiated work on preparing modernization road maps – a concentrated effort over the past nine months.

The process has been somewhat painful and a learning experience for many of the stakeholders. In the year since we received approval to move forward, we've assessed these architectures to determine their viability. We looked at current architectures, funded upgrades, and yet unfunded but planned enhancements, from the perspective of three viability objectives:

- Producibility – The ability to produce the subsystem in the future based upon the current design.
- Supportability – The ability to sustain the subsystem and meet the required Mission Capable Rates.
- Future Requirements Growth – The ability of the subsystem to meet projected combat capability requirements with the current design and avionics architecture.

Ratings were assigned per these evaluation criteria. We broke out each system through its projected life and based on current architecture, it was given a color-coded rating. Questions asked included:

- Can you accommodate what the user has defined to you as requirements growth?
- Can you accommodate increased functionality?
- Can you continue to support or sustain that into the future?
- Is it producible?

Each subsystem was scored. In summary, of the weapon systems where the avionic architectures were assessed, all exhibited some form of viability shortfall. Realizing this the warfighters, in concert with the acquisition community, are making investments to increase capability and in the process working to migrate to more open and affordable systems for the long haul.

Our next step in the assessment process is to see how we can influence the future. We are going to the program offices to see how we could leverage across platforms to make more architecture green, i.e., viable. The real test will be a year or two from now when we will see how we've influenced the programs' paths to achieve a more viable/affordable system.

That's where we are today, about four miles into this marathon. Just how successful we will be in the future depends on our ability to leverage programs and to get the acquisition community, especially our defense industry to step out in front. Customer [warfighter] funding will remain tied to increased capability, not on reducing life-cycle costs, and our ability to drive real improvements in the cost of ownership will be marginal unless we take a step forward and capitalize on these 'investment-return' opportunities. However, there is no such thing as a guarantee for industry, and they will assign these investment opportunities risk factors and rack them with other investment options. To be successful in getting them to invest, we need to be able to allow them reasonable returns. We're convinced that this can be a win-win for both industry and government. First we need to gain the momentum, get a few small wins, and then it will take off.◆

NEW DIRECTION

Operating and Cultural Changes Shadow Avionics Modernization

To accomplish a true avionics modernization program that goes beyond simply increasing capabilities to a true cross-platform open system will require operating and cultural changes in government and industry. What must emerge is an operating environment where the Department of Defense (DoD) has in place the necessary business practices and life-cycle focus to make the problem of diminishing manufacturing sources and out of production parts (DMS/OP) transparent to the customer, says Jon Ogg, director, Engineering and Technical Management Directorate, Aeronautical Systems Center, Wright-Patterson Air Force Base, Ohio. Ogg stresses that the DMS/OP is not going away. He lists the following enablers to accomplish this change in state:

- Prime/Original Equipment Manufactures Proactive DMS/OP Management: This means both government and industry getting into a proactive vs. reactive role in dealing with the ever-accelerating rate-of-change in electronic technologies.
- Long-Term Prime/Original Equipment Manufactures Relationships Fostered by DoD Commitment: These long-term relationships are absolutely paramount. You need to maintain competition at the right level, but not prescribe it at all levels.
- Defense Industry Defined/Supported Open System Architecture-Based Standards: Part of the problem we inherited from the past is the mandated standards across our business. We need to support standard interfaces, protocol and operating systems similar to the way it is done by industry in the personal computer domain. We should not mandate but rather encourage and support industry's development and maintenance of standards.
- Price-Based Procurement/Sustainment: This provides an incentive [business case] for prime/original equipment manufactures investment in the long-term producibility/sustainability of products.

Integrated Road Maps Route the Migration to Avionics Open Systems

Pamela Bowers
CROSSTALK

The Viable Combat Avionics (VCA) initiative was developed to combat aging avionics, including diminishing manufacturing sources and out-of-production parts, and to create more affordable avionics by making it easier to insert new technology and operational capabilities into systems. An integral part of the initiative is the development of integrated change road maps to guide the migration of avionics architectures to a state that enables their viability to be continued. David G. "Butch" Ardis, technical adviser for avionics systems architecture, was tapped to provide technical leadership to the VCA initiative, responsible as chief avionics architect for the aeronautical domain.

ONE on ONE

Lt. Col. Glenn A. Palmer, program director, Computer Resources Support Improvement Program, and Pamela Bowers, CROSSTALK managing editor, met with Ardis at Wright-Patterson Air Force Base to discuss these avionics road maps. Ardis is a Senior Level executive at the Aeronautical Systems Center (ASC).

Ardis began his federal service career as an officer in the Air Force. His military service included two combat tours in Southeast Asia, a teaching assignment at the Air Force Electronic Warfare Officer Training School, and a tour in project engineering at Aeronautical Systems Center. His civilian service includes assignments as technical expert for mission avionics and technical director for avionics engineering at ASC.

Q: As the person working directly to implement avionics modernization, what is your architecture template for change?

Ardis: Our change template for modernization as well as other avionics changes is centered around the Viable Combat Avionics (VCA) initiative. Aging avionics issues, avionics affordability issues, and viability issues depending on specifics, all refer to the same general VCA concepts. Affordably dealing with the changes that are occurring is a combination problem: 85 percent business and 15 percent technical. Business challenges include affordability and obsolescence. Technically, the challenge is to begin using the open-systems principles that will facilitate affordably changing our avionics systems; that is what I have been tasked to do. The technical side starts with using open-systems principles.

The term *open systems* is a multi-headed animal with lots of different definitions. Our first step in applying open-system principles is to get rid of ambiguity and say what it is we want from open systems. The *what we want* then needs to be put into performance terms that actually can be measured. Then when we put those terms into our contracts, we'll start seeing the benefits of applying open-systems principles.

When we talk about applying open-systems principles, we are talking about finding ways to affordably do the things we need to do when we change our avionics. Changes include reliability improvement, obsolescence mitigation, periodic software updates, adding new capabilities, etc. We also need new systems that are affordable to change.

Second, most of our systems are so costly that in most cases any strategy must be incremental or evolutionary in nature. We have to take advantage of the funds that will be available such as those for operational capability improvements.

Third, since we have to make big platform changes, and it must be done incrementally, a strategic context needs to be developed to determine what to put into the current contract vs. contracts years down the road. That was how the idea of making integrated change road maps came about.

Early on, mentioning affordable avionics was met with resistance. The system program directors were skeptical and saw this as a plan to cut their already lean budget. We had to explain that we were not there to advocate taking away their money. Rather, we were trying to get the greatest benefit for every dollar our operational customer spent.

We believe it will be possible to improve the efficiency and effectiveness of the funds invested if we can cut back on the amount of money we spend on diminishing manufacturing sources (DMS), obsolete parts, verification, etc., and can reduce cycle time.

Q: What functions do the avionics road maps serve?

Ardis: The real objective of the avionics change road maps is to set the strategic context for the system. To construct road maps, we began by asking program managers to do an avionics health-assessment based on the affordability of their programs. Is the design producible? That doesn't mean that the configuration won't change during production, but how much pain is associated with any needed changes? Is the design supportable and *growable*? We're asking them to look at their architecture and its implementation right now and then look at the changes operational customers are proposing. What can we do to the architecture to migrate it to a more desirable state?

Once the desired result is defined, the operational capability updates serve as a target of opportunity for shifting and evolving the architecture advancements and affordability.

The basic assumption is that if we cannot get money that is just for the basic architecture changes, then deliver the funded required capabilities changes in a way that mitigates the affordability issues we're having. I'm hopeful that if people think along these lines, as they lay out the planning process to add capabilities, they automatically start thinking about how to migrate architecture with that plan. Given budget constraints, that's the best we can do in terms of migrating to more affordable platforms.

Q: In what role do you see the Air Force Research Laboratory in avionics-related technologies in the next five to six years?

Ardis: Obviously this is Butch Ardis' opinion. There are many technology opportunities for Air Force Research Laboratory (AFRL) investment. Instead of the normal technology discussions, let's restrict this to some thoughts about things that will help us achieve the objectives of more effectively and efficiently dealing with the future viability of our avionics.

Reducing the costs of verification of modern high performance avionics is an area that needs much attention. Our avionics systems are continually undergoing changes with the attendant verification requirements. The highly integrated nature of many of our systems demands that more and more resources be used on verification – especially safety critical applications.

We have some major technical concerns in some of our architecture proposals and difficulty in verifying their performance. This is an opportunity for the labs to help. As we go to these highly integrated, high performance avionics systems it becomes necessary to develop and qualify approaches that reduce the verification resource burden. There are at least two areas for consideration. First, we need to make ease of verification a performance consideration. Avionics system architectures as well as hardware and software architecture developments need to put more emphasis on reducing verification costs. I believe the labs can help us with developing architectures that are easier to test.

Second is the verification processes themselves. The F-22 is an example. It has unprecedented abilities to engage many targets in a beyond visual range environment. Flight testing of its most stressing performance requirements takes many more resources than previous aircraft to support some of the test conditions. We must do as much as possible on the ground to make our flight tests as effective and efficient as possible – you do not want to *debug* in the air. However, the development of ground-based simulations and support tools for verification also takes tremendous resources. To effectively do hardware-in-the-loop testing of the F-22's avionics approach on the ground requires a high fidelity stimulation that captures the relative movements of all objects that the avionics senses and injects signals with the proper time and kinematic characteristics into the front end of the sensors. We really haven't dealt with that level of fidelity requirements before, and it takes a large investment.

Another verification area we could use help with is streamlining certification of architectures where we are doing multitasking and mixing levels of criticality in safety. We have to get the promised *lean* processes qualified. If we're going to cut program costs by reducing testing, then we've got to demonstrate that the underlying fundamental processes are valid. AFRL could help us develop these processes. Otherwise, if we have not *qualified* our verification processes and have to use traditional safety certification processes, we potentially have years worth of testing every time we update software in these mixed criticality, heavily loaded processors.

Agreeably, we need to do a better job of making sure the labs understand what we're trying to do with avionics in the future.

Q: What do you see down the road?

Ardis: Our biggest challenge is coming to grips with what the probable future is for avionics. I believe the objectives of Joint Vision 2020 and the implications of making our avionics systems effectively fit into a systems-of-systems operating concept will represent a major task for many of our legacy and new platforms. There is a lot of thought going into airborne and space-borne sensors that support generating timely information. Communication links are being put in so people can tap into the information. What we have not done adequately is look at how to take advantage of that information inside the cockpit. And what are we are going to do with it once we get that information to the pilot? This will drive future avionics architecture requirements.

For example when we upgrade displays, we are trying to push our programs to think aggressively about what is most likely to be done with these platforms in the systems-of-systems operations when we have that type of real-time information available in the airplane. There are going to be some very big changes required in onboard avionics. There will be too much information for the pilot to personally filter through everything. It must be automated.

"The real objective of avionics change road maps is to set the strategic context. Is the design producible ... Is the design supportable and growable ... What can we do to architecture to migrate it to a more desirable state?"

This is another area where the AFRL can help us. How do you go about changing avionics from what truly is network centric as opposed to platform centric? It is a completely different approach to the way we want to do things. I don't think that very many of our platforms have avionics that are going to be compatible. So as we evolve our architectures, I think implementing systems-of-systems requirements and achieving interoperability will drive big changes in avionics architectures and their implementations. This will be especially difficult due to the costs associated with large avionics changes.

We have the technology challenges associated with obsolescence, DMS, processors rapid change, etc., but avionics performance requirements are going to be a real shock when we finally step up to implementing the systems-of-systems requirements.

As for other challenges, road-mapping a major activity is frustrating because we can't show the operational customers solutions that have saved them billions of dollars, yet. We are hopeful that down the road, customers will start seeing the benefits of the performance attributes associated with applying open systems.

A performance-based approach is key to where we are trying to push things. Just specifying open systems alone is not sufficient. We are trying to put things in the contracts that will give us the benefit of open-systems approaches – the major benefits being ease of change and verification of changes.◆

Customizing the Software Process to Support Avionics Systems Enhancements

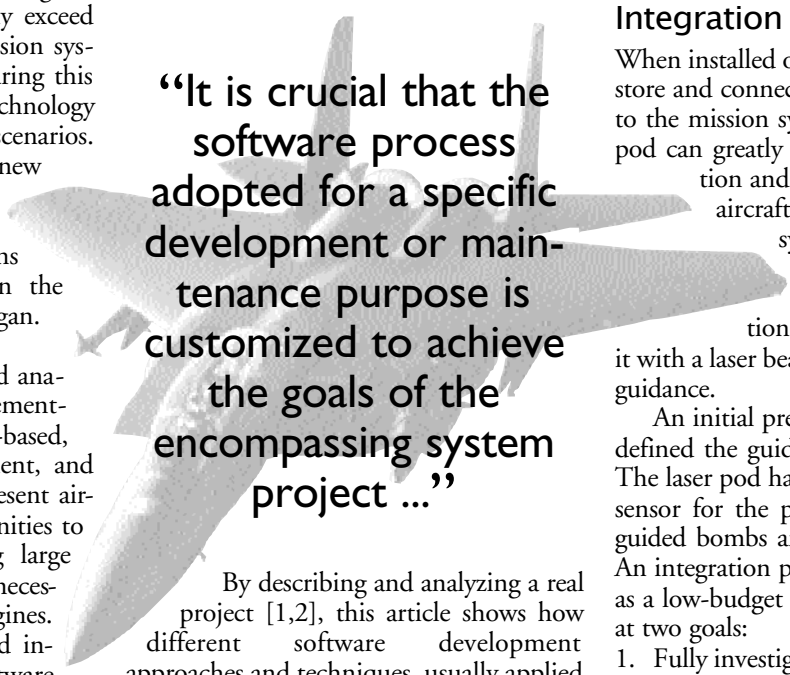
Dr. Paolo Donzelli and Roberto Marozza

When an organization considers enhancing a software-intensive system, particularly an avionics system, selecting the process to be adopted must include consideration of the particular encompassing project. This includes the application domain, the size and complexity of the final product, the hosting system characteristics, etc. Simultaneously, it must be driven by the specific organization's goals, environment, and maturity. By describing and analyzing a real project, this article shows how different approaches and techniques, usually applied in isolation, can be selected, customized and combined to design a software process that better satisfies the organization's goals and meets its constraints. The project was undertaken to investigate the feasibility of enhancing an aircraft mission system by integrating new capabilities, and eventually to identify a quick, low-cost and low-risk solution. An uncertainty-driven product architectural framework together with an ad-hoc simulation-based supporting environment were used to combine in an effective and controlled fashion the waterfall, the Rapid Application Development, and the Incremental Development models.

It is widely accepted that the key point in keeping modern combat aircraft up-to-date is the mission system, i.e., the collection of computers, electronics equipment, and sensors that allow an aircraft to perform its mission. Aircraft operational lifespan has been steadily increasing during the past 60 years, and can now easily exceed three decades (see Figure 1). Mission systems quickly become obsolete during this time due to rapidly advancing technology and fast-changing international scenarios. For example, the capability of new computer systems, or the flexibility required by recent peacekeeping and peace-enforcing operations were not even foreseeable when the design of some modern aircraft began.

As final users, Air Forces are the best candidates to identify and analyze new requirements to be implemented. The availability of standard-based, off-the-shelf sensors and equipment, and the importance of software in present airborne systems offer new opportunities to upgrade aircraft without having large industrial facilities such as those necessary to modify airframes or engines. Thus, many users have developed in-house facilities to evaluate software running on their aircraft, experiment with potential enhancements, and eventually introduce approved modifications.

Since airborne software is vital, yet only a component of a higher complex system, a systemic approach must be adopted in designing the corresponding development process. It is crucial that the software process adopted for a specific development or maintenance purpose is customized to achieve the goals of the encompassing system project, and ultimately of the organization. For military organizations in particular, technical problems posed by the complexity of the target systems often have to be dealt with in combination with specific needs and constraints such as ad-hoc capabilities at time of international crisis, available experience, staff turnover, budget reductions, relations with and between industrial partners, etc.



"It is crucial that the software process adopted for a specific development or maintenance purpose is customized to achieve the goals of the encompassing system project ..."

By describing and analyzing a real project [1,2], this article shows how different software development approaches and techniques, usually applied in isolation, can be selected, customized, and combined to better meet organizational needs. The project was undertaken to investigate the feasibility of enhancing an aircraft mission system by integrating a laser designation pod. Here we describe the software aspects of the integration by focusing on the software process devised to support the integration study, and on the rationale behind the choices made. More details about this specific project can be found in [1,2].

In this article we briefly introduce the integration problem then provide an overview of the proposed software process, and of its rationale. Then we discuss the final results by providing both qualitative and quantitative insights. Finally, we conclude and summarize the benefits of the adopted approach.

Integration Problem Overview

When installed on an aircraft as an external store and connected as an added-in facility to the mission system, a laser illuminating pod can greatly improve both the navigation and attack performance of the aircraft. In practice, such a subsystem will allow the pilot to sight a ground point, obtain the relative position, and eventually illuminate it with a laser beam for subsequent weapon guidance.

An initial pre-feasibility study only had defined the guidelines for the integration: The laser pod had to be used as a targeting sensor for the precision delivery of laser-guided bombs and as a navigation sensor. An integration project was therefore set off as a low-budget short-term activity aiming at two goals:

1. Fully investigate the feasibility of equipping the aircraft with such a subsystem.
2. If feasible, identify an economical and low-risk integration solution.

After an initial assessment of the integration problem, various risk-prone areas were identified.

Complexity of Solution Space

Finding a solution to the integration problem means to define a complete and consistent set of requirements that the new system (i.e., the aircraft equipped with the laser pod) has to satisfy. Only a minor set of these requirements concern functional

aspects, e.g., the data the laser pod has to provide as navigation sensor; most of them are related to non-functional aspects. These include both human factors such as pilot workload, pilot performance, and situation awareness [3], and system quality attributes such as usability, safety, reliability, time, and cost [4].

In comparison with functional requirements, non-functional requirements are highly subjective (e.g., test and front-line pilots can have a different perception of the same problem), strictly related to the particular context, and more difficult to discover, state, and validate without interacting with the final system. This increases the dimension of the solution space, introduces instability in the requirements, and makes it difficult to compare different alternatives.

Complexity of Target Platform

Most of the functions in the mission system are performed via a cooperation of two or more subsystems. As a consequence, modifying or enhancing such functions requires operating on different equipment, which may adopt different hardware and software solutions requiring a broad range of skills not usually available in the same personnel. Moreover, equipment is produced and maintained by different contractors, so that the Air Force is faced with different levels of product visibility, procurement processes, schedules, and costs.

Novel Aspects of Project

Being a single-seat aircraft, the problem of adding the control of the laser pod to all the other tasks usually carried out by the pilot needed to be completely investigated, both for safety and performance reasons.

Project Organization

Based on the initial goals, constraints, and outcomes of the initial assessment of the integration problem (Integration Problem Overview), the project has been organized following some simple guidelines:

- Minimize the impact of integration on the aircraft mission system.
- Exploit internal resources and capabilities, both in terms of personnel and equipment.
- Reuse previous experiences, i.e., lessons learned and available products, e.g., requirements, algorithms, software.
- Allow uncertainty to be part of the project to improve the ability to investigate the solution space.

In practical terms, this has led to making precise choices in organizing the team and defining the software process.

The Team Organization

The project team was structured into three different sub-groups with specific competencies, responsibilities, and workload:

- The software development team, tasked with managing the whole project and developing the necessary software, was composed of two software engineers and four technicians. The group members worked together at the requirements level; subgroups were identified to better deal with the specific needs of the various subsystems affected by the integration.
- The hardware support team, tasked with implementing the avionics integration of the laser pod, was composed of two technicians. In addition, they played the role of logistics and maintenance experts during the requirements definition stage.
- The user group, tasked with collaborating during the requirements discovery and validation phases, was composed of two test pilots. Both of them had a specific experience with laser pod-based operations and were supported by front-line pilots.

Such a team structure provided us with great versatility in tackling personnel-related problems. First, it allowed us to really involve the stakeholders and exploit available expertise. Second, it limited the impacts of staff turnover, unavoidably due to the project schedule, by dealing with them within each group. Third, it incremented the degree of concurrency between the different tasks to be performed within the organization, regarding this or other projects. In particular, both the hardware support team and the user group could better plan their own involvement in this project without hindering other projects. Meanwhile, the software development team, thanks also to the adopted software process (see The Software Process below), was not affected too much by some unavoidable delays that the other subgroups had in providing feedback and support.

The Software Process

The difficulty in adopting a classical waterfall-based process [5], i.e., a process based upon a series of sequential steps that goes from requirements analysis to system delivery, emerged clearly just after a first attempt at defining the initial set of requirements. The waterfall model makes the development process more visible by providing a set of milestones around which the management can plan, monitor, and control a project. However, it is based on the assumption that most of the requirements can be frozen

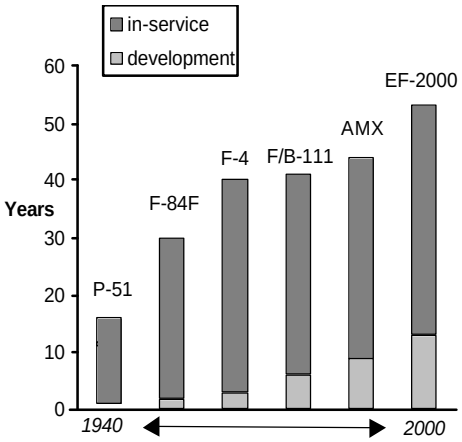


Figure 1: *Aircraft Operational Life*

at the outset of the project, whereas it is well recognized to perform poorly in case of requirements instability [6].

In our case, not only was it impossible to state precisely most of the requirements, but there also were quite a few system areas about which only general and soft considerations could be made. For example, we could identify the need for a pilot interface suitable to reduce workload and improve situation awareness, but we could not translate this into a series of requirements able to be implemented.

Thus a specific architectural framework in which to develop the system was proposed. It had to be flexible to accommodate uncertain software development approaches, yet rigid enough to guarantee a visible development process required by the organization. As illustrated in Figure 2 (see page 12), the software system was designed as a collection of interacting components, each of them acting as a focus for a particular requirements area with a specific uncertainty level.

A set of man-machine interface components, a pod-control component, and a network interface component form the architecture.

- The man-machine interface components represent the software to be added onto the displays (e.g., the head-up display) and the control panels of the aircraft cockpit. They modify the pilot-aircraft interface handling all the information to be displayed and the user interactions enabling the pilot to use the laser pod. Here the level of uncertainty was highest: The user group drafted some guidelines about the pilot operations (mainly on the basis of previous experience), but could not be more precise about the interaction with the individual pilot. Even the specific displays and switches to be used were matters for discussion, leaving the number of components to be developed an open issue.

- The pod-control component implements the algorithms to physically control the pod, for example allowing the pilot to steer the pod in a specific direction. Although many of such algorithms were reused (from available literature, code, etc.), the component still presented some degree of uncertainty, mainly regarding its real-time performance in the new avionics environment. In designing and allocating the software, both measurable parameters (computation elapsed time) and pilot judgement (stability of the track image) had to be taken into account.
- Finally, the network interface component was specifically designed to handle the message passing between the laser pod and the physical systems hosting the other components. This resulted in the most stability, in fact, the laser pod, an off-the-shelf item, dictated most of the messages and the component had to be allocated onto the main digital computer.

The adopted architectural framework (see Figure 2) allowed us to combine different development approaches in a controlled fashion.

A full rapid application development (RAD) approach was applied to the man-machine interface components, which were developed as a set of concurrently evolving prototypes [7]. In RAD, the exact opposite of the traditional software approach is held true: Time and resources are fixed, as far as possible, and the requirements are allowed to change. This suited our situation well; the stakeholders did not have a clear initial idea of the system to be developed. This was expected to mature over time, and different solutions appeared to be equally valid. RAD enabled us to rapidly construct primitive versions of the components by heavily involving the members of the user group.

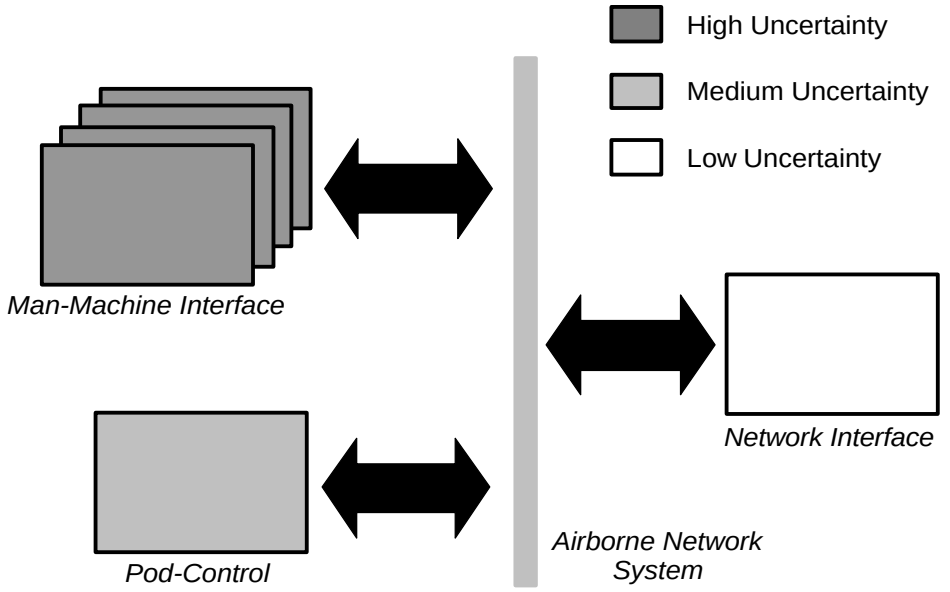
The user group's key role in the development process has been widely recognized. The major advantages of the so-called user-centered approaches [8] come from letting the user participate in and contribute to the design process from its first inception onwards. Through the user group, the users felt that they were an active part of the development team. They took part at the initial design sessions, and their feedback was incorporated to correct, refine, and enrich the emerging system properties until the final set of components was obtained. For the pod-control component, a throw-away prototype [9] was used to quickly compare the different algorithms available before setting off on a more traditional waterfall-based approach. To develop the network interface component, a waterfall-based process was applied from the beginning.

The components were developed in parallel with tight collaboration among the team groups and a continuous exchange of results, experience, and products. An ad-hoc Avionics Simulation Station was developed [10] that supported the whole integration study (see Figure 3). Apart from the Software Development and Testing Stations used to modify the software and test it directly on the airborne computers, the Avionics Simulation Station consists of a mixture of real and simulated equipment. For example, while all the main sensors are simulated (inertial navigation system, radar, etc.), a real laser pod is used together with real units associated with the cockpit (displays, control panels, etc.). The cockpit allows a pilot to form part of the overall station so that the effects of the investigated laser pod integration on pilot performance can be determined.

The Avionics Simulation Station enabled us to work with a variable combination of components running on their allocated computers, and components still at a prototype level. In particular, the network interface component was developed directly on the main digital computer. The other components were implemented using a high-level equipment simulation tool to evaluate them in a real environment without affecting the aircraft equipment. Such a simulation tool allowed us also to adopt for the prototypes the same programming languages used by the potential hosts, highly simplifying the subsequent porting phase.

The prototypes were used to enrich the Avionics Simulation Station with the laser pod. Then the Avionics Simulation Station was employed to analyze the pod operating procedures with the members of the user group. Only when the prototypes reached a stable state, were they moved to the corresponding computers to finalize evaluation and testing. This environment guaranteed the right trade-off between flexibility and rigor to support the integration study. Different alternative solutions were easily investigated (e.g., a different set of components or different allocation of the same components), while the use of real equipment provided early feedback on the project's technical feasibility. For example, the real-time performance of the pod-control component on the selected target computer could be assessed while the user group was still defining the interface to be adopted. Furthermore, it strongly reduced our dependency upon the equipment manufacturers, for example by enabling us to involve them only at the final stage (the porting step), thus reducing the associated costs and delays.

Figure 2: *An Uncertainty-Driven Software Architectural Framework*



Although RAD is a good method to deal with unstable requirements, it suffers when applied to big projects or when requirements instability is not confined to a specific area. In both cases, it can in fact lead to an explosion of project complexity and of associated risks. Whereas the initial architectural framework (see Figure 2) enabled us to manage instability in order to increase our control over the project, we decided to combine RAD with an incremental development method. In other words, the initial unstable set of requirements was broken down into three more manageable subsets:

- Set A regarding the basic laser pod control functions (e.g., in-flight and on-ground pod test, basic pod orientation, etc.).
- Set B incrementing Set A with the laser

pod-based navigation functions (e.g., use of the pod to acquire an on-ground point position).
- Set C incrementing Set B with the laser pod-based attack functions (e.g., use pod capabilities to support a specific ground attack mode).

In moving from Set A to Set C, the complexity of the man-machine interface clearly increased. By requiring more information to be displayed and more controls made available, the incremental approach combined with RAD allowed us to gradually involve the user group, which had time to face new problems (new requirements) starting from a previously established platform (already implemented requirements). Besides facilitating requirements capturing and formalization, such an approach led also to better-designed software, for all the involved components.

The resulting software process is schematized in Figure 4, which summarized what has been said so far. As shown in Figure 4, the process is based on three main phases. Phase 1 is the initial requirement analysis phase during which the decisions of adopting the architectural framework depicted in Figure 2 and of dividing the requirements into incremental subsets were made. During Phase 2, the system evolved as a set of components and prototypes by first implementing the requirements set A (version A), then the set B (version B), and finally the set C (version C). For each version, using Avionics Simulation Station led to correction and refinement loops. We only passed on to Phase 3 when a high confidence in version C was reached. Here the components still at a prototype level were ported on the real equipment, and the final system test and evaluation performed.

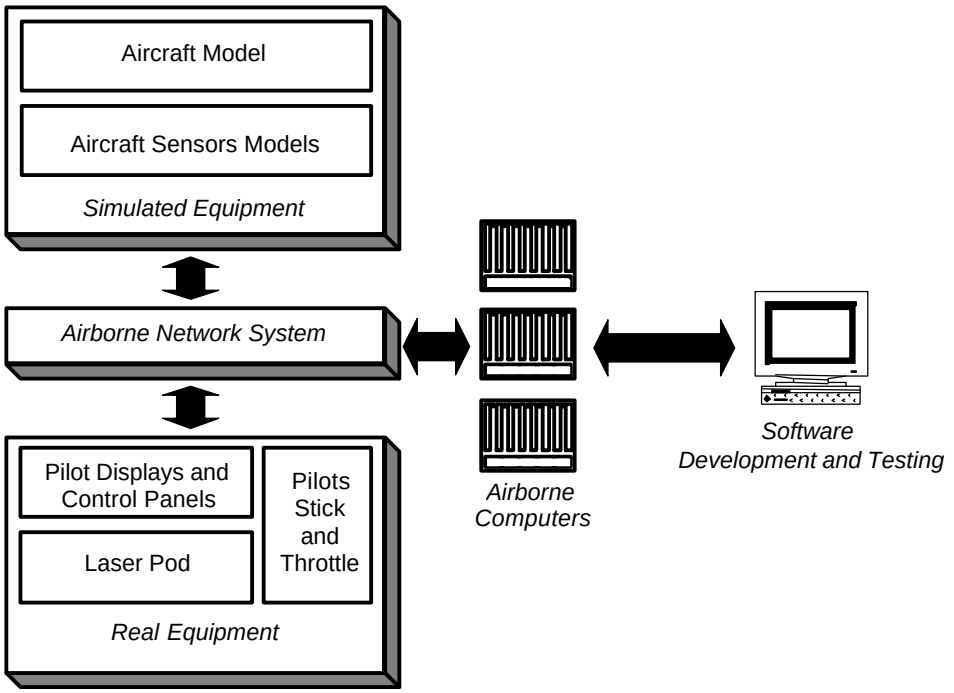


Figure 3: *Architecture of the Avionics Simulation Station*

Project Results

The integration study and the supporting software process allowed us to fully investigate the integration feasibility (goal 1), and then to identify what we considered an economical and low-risk integration solution (goal 2).

The flexibility of the process and product enabled us to find both quantitative and qualitative answers from the early stage of the project. Aspects such as the real-time performance of the modified mission system, the compatibility of the new software with the target equipment, and the pilot needs were carefully investigated. In particular, it was assessed that it was feasible for the laser pod to be operated by the pilot while flying the aircraft.

The software produced consisted of about 11,000 lines of code (LOC); 1,500 LOC of Ada were written as ancillary code to adapt the prototypes to the Avionics Simulation Station. The impact on the total airborne software was relatively small (with an average of 1 percent modified and 5 percent new software on the various airborne computers), thus sensitively reducing the effort required to reevaluate and test the existing functions.

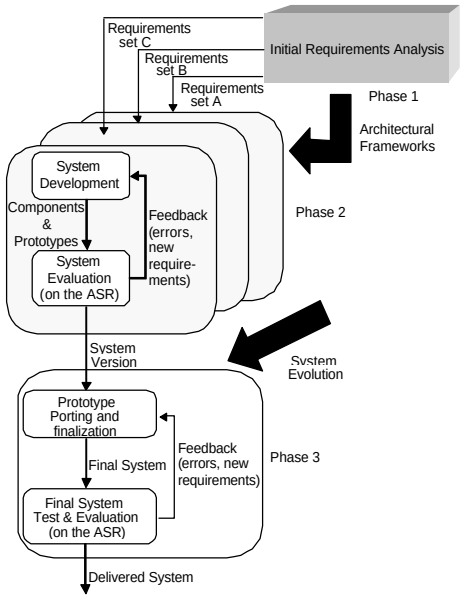
The development lasted for 10 months during a calendar time of 15 months. This difference was due mainly to preemption of personnel for other tasks (about three months) and to some bureaucratic delays with partner industries (about two months). Most of the effort (about 90 percent) was spent on Phase 2, with 45 percent required for development of version A, 30 percent for version B, and 15 percent for version C. Quality confirmation of the components and prototypes forming the system version C, and of the adopted process Phase 3 required only 5 percent of the effort. For the porting, three people from the manufacturers were involved for a limited number of meetings and a total of six days of actual software development. The final testing and evaluation revealed only some minor defects.

Conclusions

This article described the software process adopted to support the integration study of a new weapon system onto an existing military aircraft in the context of a low-cost, high-confidence-of-success project.

To integrate two systems means to identify the synergistic combination that best exploits them both. At the initial stage, only some general guidelines are usually fixed, whereas many different solutions appear equally valid and worth investigating. Such uncertainty drove the design of the initial software architectural framework in which the waterfall, the RAD, and the incremental software development models were combined in an effective and controlled fashion. The process provided the necessary mix of visibility, flexibility, and performance, taking into account the available personnel, time, and funding, as well as increasing the organization's experience and improving collaboration with the industrial partners.◆

Figure 4: *Adopted Software Process*



COMING EVENTS

October 15-18
SEI 16th Annual
Software Engineering Symposium
Washington D.C.
www.sei.cmu.edu/symposium

October 22-26
Systems Engineering
and Supportability Conference
San Diego, CA
http://register.ndia.org/interview/register.ndia

October 28 - November 1
5th Annual DoD Symposium and Exhibition
Kansas City, MO
www.ndia.org

October 29 - November 2
6th Annual Expeditionary
Warfare Conference (EWC)
Panama City, FL
http://register.ndia.org/interview/register.ndia?~brochure~270

November 6-8
TechNet Asia-Pacific 2001
Honolulu, HI
www.afcea.org

November 12-16
5th International Software and Internet
Quality Week - Europe 2001
Brussels, Belgium
www.qualityweek.com

November 13-15
1st Annual CMMI Technology Conference
and User Group
Denver, CO
djenks@ndia.org

February 4-6, 2002
International Conference on COTS-
Based Software Systems (ICCBSS)
At the Heart of the Revolution
Lake Buena Vista, FL
www.iccbss.org

March 19-21, 2002
Federal Office Systems Exposition 2002
Washington D.C.
www.fose.com

April 28 - May 3, 2002
STC 2002
"Forging the Future of Defense
Through Technology"
Salt Lake City, UT
www.stc-online.org

References

1. Donzelli, P., Marozza R. "Laser Designation Pod on the Italian Air Force AMX Aircraft: A Prototype Integration," Proceedings of the NATO/RTO (Research and Technology Organization) Systems Concepts and Integration Panel Joint Symposium on Advances in Vehicle Systems Concepts and Integration, Ankara, Turkey, April 26-28, 1999, published by NATO-/RTO, BP 25, 7 Rue Ancelle, F-92201 Neuilly-Sur-Seine Cedex, France, April 2000, ISBN 92-837-0011-2.
2. Donzelli, P., Marozza R. "Un Laser Pod Anche Per l' AMX," Rivista Aeronautica (Italian Air Force Journal), Dec. 2000, Roma, Italy. <www.aeronautica. difesa.it>.
3. Prince, C., Salas E., and Emery L. "Situation Awareness: What Do We Know Now That the 'Buz' Has Gone?" Engineering Psychology and Cognitive Ergonomics, Vol. 3: Transportation Systems, Medical Ergonomics and Training, pp 215-222. Edited by Don Harris, Ashgate Publishing Ltd, Aldershot, England, 1999.
4. Chung L., and Nixon B. "Dealing with Non-functional Requirements: Three Experimental Studies of a Process Oriented Approach," Proceedings of the International Conference on Software Engineering, Seattle, WA, USA, 1995.
5. Mills, H.D., O'Neill, D., Linger, R.C., Dyer, M., and Quinnan, R.E. "The Management of Software Engineering," IBM System Journal, 24 (2), 1980.
6. Bohem, B. "Anchoring the Software Process," IEEE Software, Vol. 13, No. 14, July 1996.
7. DSDM Consortium. Dynamic Systems Development Method, Version 3. DSDM Consortium, Ashford (UK), 1997.
8. Norman D. and Draper S., "User Centered System Design," LEA, Hillsdale, N.J.,1986.
9. Stytz M.R., Adams T., Garcia B., Sheasby S.M., and Zurita B. "Rapid Prototyping for Distributed Virtual Environment," IEEE Software, Vol. 14 No. 5, Sept./Oct. 1997.
10. Donzelli, P., Moulding M.R. "Developments in Application Domain Modeling for the Verification and Validation of Synthetic Environments: A Formal Requirements Engineering Framework," Proceedings of the Spring '99 Simulation Interoperability Workshop, Orlando, FL, March 1999.

About the Authors

Paolo Donzelli, Ph.D., is an advisor with the Department of Informatics of the Office of the Prime Minister in Italy. A former serving engineering officer with the Operational Testing Centre of the Italian Air Force, Dr. Donzelli was a senior research fellow with the Computing Information Systems Engineering Group, at RMCS, Cranfield University (UK). Dr. Donzelli has a variety of interests in the software engineering area, and his Ph.D. thesis was in software process quality modeling.

Office of the Prime Minister
Department of Informatics
Via della Stamperia 8
00187 Roma, Italy
Phone: (39) 0335-736-5194
Fax: (39) 06-6779-4736
E-mail: p.donzelli@palazzochigi.it

Roberto Marozza was a serving engineering officer with the Italian Air Force before moving into industry. He has been program manager of the on-board mission software for the Anglo-Italian EH-101 ASW helicopter, and has also worked in some airborne software projects for the Tornado and the AMX attack aircraft. Recently, he was involved at Aerospatiale in the software requirements definition of the ATV, the orbiting vehicle that will transfer unmanned payloads from ground to the International Space Station. Marozza has a Laurea Degree in electronics engineering.

Banca d'Italia
Largo Guido Carli 1
Frascati, 00040 Roma, Italy
Phone: (39) 0348-654-2067
E-mail: rmarozza@libero.it

The Challenges of Software Certification

George Romanski
Verocel, Inc.

The safety critical community – those involved in developing and verifying safety critical systems – is very conservative and adverse to change. Meanwhile, technology is changing rapidly, and there is pressure to adapt systems to improve their efficiency and safety. This presents a number of challenges. The community has already addressed some; others are in process. While the guidance on airborne software certification is mature, the issues with software reuse, military avionics certification, ground-based software, and object oriented technology are still evolving.

Computer-controlled systems pervade our lives. We take many of them for granted, although many are critical to our safety. Dire consequences may result if the software in an automobile's computer-controlled braking system failed during a highway maneuver. Software faults in the automobile's computer may cause the brakes to malfunction. What can be done to make sure that this software does not contribute to the cause of an accident?

An extreme measure would be to limit the speed of automobiles to a walking pace. When automobiles were first licensed to travel on public highways, a person carrying a red flag had to walk in front of them. Clearly this restriction is impractical, and was abandoned many decades ago. An alternative is not setting up the computer system as the sole means of control. A computer-controlled braking system could include hydraulic/mechanical controls that provide backup operations when the computer system fails. However, this approach may be impractical for some complex systems.

A bicycle can turn corners much faster than a tricycle, but it requires active control by the cyclist to maintain balance. Similarly a high performance airplane may be built with an unstable flight profile. For example, a paper plane with wings that are level may swerve and dive during flight, but a plane whose wings tilt up slightly glides smoothly. Fighter planes are often built unstable to make them more agile but they include movable surfaces. A computer system controls the movable surfaces and induces stability through control algorithms.

A modern transport airplane may be built to have a stable flight profile, but computers are used to control and optimize aircraft flight. These types of computer systems and their software have a direct impact on the safety of the aircraft and its occupants. A level of assurance is required to provide confidence in this software. In commercial avionics systems a document called "Software Considerations in Aircraft Systems and Equipment Certification" is used to provide assurance. In the United States, this document is called DO-178B and is published by Requirements and Technical Concepts for Aviation, Inc. (RTCA) [1].

> "Although DO-178B is referred to as a guidance document, it is treated as a standard that imposes requirements on the development and verification of airborne avionics systems."

DO–178B

Although DO-178B is referred to as a guidance document, it is treated as a standard that imposes requirements on the development and verification of airborne avionics systems. A Federal Aviation Regulation lists DO-178B as a means of compliance that is acceptable to the software regulators in the avionics community. In the United States, the regulatory body is the Federal Aviation Authority (FAA). While DO-178B is not the only means of compliance, compliance with the objectives of DO-178B must be shown if a different approach is used.

In Europe, a similar statutory regulation called ED-12B is published by EUROCAE. This is the same document as the DO-178B and was produced by an international consensus-based committee representing practitioners as well as regulators.

DO-178B is intended to describe the objectives of the software life-cycle processes, the process activities, and the evidence of compliance required at different software levels. The software levels are chosen by determining the severity of the failure conditions, which may affect the aircraft and its occupants [2]. The failure conditions are named and have corresponding levels identified by letters. For each level there is a set of process objectives that must be satisfied. An example of a process objective is A-7.3 "Test Coverage of low-level requirements is achieved." The number of process objectives by software level is shown in Table 1 (see page 16).

The certification agencies have received many requests to clarify the intent of DO-178B since its publication in December 1992. A consensus-based committee called SC-190 (and WG-52 in Europe) was formed and tasked to propose clarifying text. After four years of work by 150 registered members, a document called The Annual Report for Clarification of DO-178B (DO-248A) was published by RTCA, Inc. [3]. This document provides corrections (typographical and editorial), answers to frequently asked questions (FAQs), and discussion papers.

A subset of the SC-190 committee is continuing with a new document for use in Communications, Navigation and Surveillance/Air Traffic Management systems (CNS/ATM). This will result in a new software assurance document intended for ground-based (and space-borne) systems. There are subtle differences between the way the two application domains are treated. Airborne systems undergo a certification process while ground-based systems go through an approval process. In the following text certification materials describe materials that support certification or approval. Here are some typical FAQs published in DO-248A (shortened for brevity):

Q: Is recursion permitted in airborne applications?

Failure Condition	DO-178B Software Level	Process Objectives
Catastrophic	A	66
Hazardous	B	65
Major	C	57
Minor	D	28
No Effect	E	0

Table 1: *Relationship Between Failure Conditions, Levels, and Objectives*

A: Yes! But it must be bounded.
Q: Is source-code to object-code traceability required?
A: Yes, if the applicant is providing coverage analysis at the source-code level and the assurance is at level A. No, if coverage analysis is provided at the machine-code level.
Q: If some run-time functions are inlined, is coverage still required of the run-time functions?
A: Yes! Coverage analysis is required of all of the code that may be reached within the address space.
Q: Can compiler features be used to simplify coverage analysis at object code?
A: Yes! For example, short circuit conditions may be used. However, as the compiler feature is being used as a verification tool, this feature of the compiler must be qualified as a verification tool. (Qualification is the process of assuring that a tool can be used in place of a verification activity performed manually.)

One of the most discussed topics is the use of previously developed software (PDS). Commercial off-the-shelf (COTS) software is considered PDS as it may be developed independently of any specific airborne application. Operating systems (OS) may be considered to be COTS software and may have been developed using in-house development processes that are not necessarily compliant with DO-178B requirements. These pose a burden on the user of the OS to reengineer the verification evidence required unless it is made available by the OS developer.

DO-178B makes provisions for reengineering requirements, design information, tests, and review of all artifacts in accordance with the DO-178B objectives. The processes must be documented in a set of plans and evidence must be available to show that this was performed in a controlled way. DO-178B provides an alternative means mechanism allowing developers to present evidence that is not typical. The developer has the burden of proof that the materials presented are acceptable alternatives to a risk adverse audience.

Operating System Issues

An OS takes control of the target machine on which it runs. It shares resources between processing threads. The threads may be represented as processes, tasks, or co-resident applications, depending on the nature of the OS. The shared resources include processor time, interrupts, memory, and input/output transactions to name a few. The OS has visibility and control over application programs and would have a direct impact on system behavior if the OS were to malfunction. Because of this close connection between the OS and the applications, the OS must have certification evidence at least to the same software level as the systems that it supports. This means that all certification criteria that apply to the applications also apply to the OS. In particular, there must be a level of confidence that the OS itself behaves deterministically, and that the underlying applications will be controlled in a deterministic way.

The level of determinism may be based on functionality, resources, and time. Functional determinism can be demonstrated through testing only if the results of a function are the inevitable consequence of its inputs. The function inputs are the parameters, but may also include global variables, and possibly external states based on interrupts. Clearly the more variables there are, the more difficult it is to demonstrate functional determinism through testing alone.

Use of resources must be bounded, otherwise their consumption will grow unchecked. This includes the use of dynamically allocated memory, like the heap, but also includes a bound on stack space. The OS used in safety critical systems may prevent use of dynamic memory allocation or may restrict this after the system completes its initialization. If the OS does not offer such precautions, then the application developers must be very cautious to ensure that memory creep does not cause the system to malfunction at some point in the future. Application code as well as the OS affect stack growth. An operating system will allocate space for the applications and may allot some of this space for control data structures such as task control blocks. After this allotment, the operating system functions typically consume little of the stack, and release it as soon as the call is finished. It is up to the application programmer to estimate the worst-case stack usage space. Users typically allow large margins to the space allocated to ensure that the resource is not entirely consumed.

Time is a difficult resource to measure and allocate. In an effort to improve processor throughput, hardware engineers have added many improvements to modern computing devices. Cache memory, pipeline processing, and co-processors may improve performance tremendously, but they also make it very difficult to put an absolute bound on the worst-case execution time for a particular function. Nevertheless, the performance improvement through the use of cache memory can be so dramatic that in many situations it becomes the overriding concern. The performance of an OS function may be dependent on the contents of instruction and data cache memory.

This varies depending on the execution paths taken through the applications as they run. In practice, timing measures of OS functions add little to the question of deterministic behavior. Typically, the application execution time is measured under estimated worst-case conditions to determine if the time bounds can be met. Measures of tasking performance under these worst-case conditions can be used to calculate the total throughput, and then determine whether deadlines of the tasks cooperating in the application can be met. This leaves the burden of timing measurement and verification to the application developer.

The COTS Reuse Argument

Software may have been developed and in service for a long time with no problems. If we have a record of the behavior, then intuitively it seems that we should be able to trust this software. Use-of-service history as a means of obtaining approval credit is an attractive option to help reduce certification costs. The notion is that if an application is used and its service history is recorded (frequency, severity, and distribution of faults found) then by extrapolation, similar behavior would be expected in applications running the same software in equivalent environments.

Low fault rates in the past may mean expected low fault rates in the future. This reasoning is particularly attractive for com-

panies that developed systems for military airplanes who want to reuse the same technology for commercial aircraft. However, this approach should be used with caution. At the higher software assurance levels, requirements for coverage analysis are intended to provide a measure of the absence of unintended functionality. By showing what code is executed in the application and what is not executed, the shortcomings of the requirements-based testing process can be estimated. If any code is discovered that cannot be traced to a requirement, then this dead code must be removed. Use-of-service history will not show presence of unintended functionality, so it does not satisfy the coverage analysis objective.

Coverage analysis is also used to show the effectiveness of testing. The type of coverage analysis required by DO-178B depends on the software level. At level C, only statement coverage is required. At level B, all entry points and exit points as well as all decisions and their outcomes must be covered in addition to all statements. At level A (the highest), coverage requirements of level B in addition to Modified Condition/Decision Coverage (MC/DC) is required.

MC/DC analysis is a unique requirement to DO-178B. Its goal is to show that each condition has its intended effect on the outcome of a decision. Applicants trying to reduce the scope and cost of the test and analysis effort have applied a number of different interpretations. To ensure that the interpretations are common, the SC-190 committee produced clarification through a discussion paper. This paper is included in the DO-248B document. Here are some valid approaches: perform coverage analysis at the machine-code level, show source-to-object code traceability together with coverage analysis at the source-code level, or develop multiple voting systems and use different languages where each compiler used is created by a different developer.

A draft policy for reusable software components (RSC) has been developed to allow certification evidence and its approval to be reused when it exists. The intent of this policy is not aimed at reusing components in different variants of a particular system being deployed. If a certified system exists, certification credit can be taken for components when moving the system to a different aircraft. This is already covered by other regulations.

A developer producing a software component and developing certification material in the absence of a specific avionic system may be creating a reusable software component. The FAA does not charge for its approval services, so it does not deal with such developers directly. (Otherwise the burden of dealing with software suppliers directly would be overwhelming.)

Airframe or subsystem manufacturers (such as the developer of an aircraft or a flight management system) may establish a certification liaison with the FAA as an applicant. As part of the delivery of materials for review, the applicant may submit DO-178B compliant materials for the RSC. This submission will also include information such as proposed software level, identification of the processor, and identification of the compiler used. Once approval of the airframe or subsystem is obtained, the FAA may provide a letter to the RSC developer and to the applicant documenting certification credit. This letter either reduces or eliminates certification effort required on a new project.

Military Avionics

To reap the benefits of a wider audience and participation by practitioners outside the Department of Defense (DoD) domain, the agency gradually moved towards standards that were co-developed with industry members. DO-178B is a consensus-based guidance document that has been adopted by the DoD for certain safety critical systems. Development of certification evidence in accordance with DO-178B is not undertaken retroactively; new projects and updates to projects do adopt this guidance document in place of a military standard. The initial draft policy focuses on transport aircraft. Fighters, bombers, and unmanned vehicles are excluded. The policy is directed at Communications Navigation and Surveillance/Air Traffic Management (CNS/ATM) systems, both airborne and ground based. The FAA owns the U.S. airspace. The U.S. Air Force is required to show that its transport planes do not degrade airspace safety during peacetime.

It could be argued (in jest) that military pilots have parachutes and cannot sue the government if an airplane malfunctions, so the software levels for the systems can be lowered. In practice, the software quality is taken seriously, but not all of the objectives of DO-178B are applicable in the DoD setting. The regulations, policies, and procedures within the FAA have evolved to encompass the DO-178B document. Certification liaison procedures are part of the approval process documented in DO-178B. An airborne system development and certification project is encouraged to form a relationship with an Aircraft Certification Office of the FAA very early in the life cycle of the project. Throughout the project lifetime, FAA personnel and/or designated engineering representatives (DERs) oversee all steps through the project phases. DERs are engineers who have been accepted through an approval process to act on behalf of the FAA. These engineers may provide guidance to the developers and have the authority to approve the materials developed for certification.

This certification liaison process is still to be developed within the DoD. On military projects, the contractual and approval processes and adherence to military standards have been used to measure a project. The DoD approach has been to contract with a supplier to develop a system subject to the provisions of an agreed contract. The FAA aircraft certification approach is much more open-ended. It allows applicants to spend their money seeking certification approval from the FAA.

The DO-178B guidance document lists objectives that must be satisfied, but it does not prescribe how. Through the certification liaison and DER review/approval process, the process plans should be developed and agreed upon. As long as certification materials are produced according to the documented processes, they should be acceptable during the final audits before approval.

Ground-Based Systems

The ground-based community (CNS/ATM) faces a similar challenge to the DoD as both funding and approval are bestowed from the same organization.

The Air Traffic Management systems have growth challenges. Many of the control centers use systems that are becoming obsolete, while at the same time air traffic continues to increase. The projected growth is 6 percent annually in Europe and 4 percent in the United States. This comes at a time when the capacity loading is already very high.

The promise for the future is to improve capacity and safety through the introduction of free flight. Current technology allows commercial airplanes to fly from one airport to another inside prescribed corridors at prescribed heights. This reduces the workload of air traffic controllers and allows them to focus on maintaining separation between aircraft. The free-flight system will allow an aircraft to choose its preferred climb from the departure airport, its preferred path, and its preferred descent to the arrival airport. Clearly if each aircraft were to take this

approach independently, the result would be chaotic and dangerous.

By disclosing its intended behavior, an airplane may join the set of aircraft managed by a ground-based system. There is much data to be accumulated, shared, and tracked to avoid possible conflicts. Static information must be uploaded to the plane describing the local terrain, airways, and other airport information. Dynamic information is uploaded as required throughout the flight, including weather, possible warnings, capacity constraints, and special use airspace schedules (e.g., military requirements). Given this information the pilot can produce a flight plan that results in a filed flight trajectory. This can be treated as an object, which will then be used by ground-based systems.

During flight, the pilot may wish to change the flight plans, but can only propose a change that must be approved by the ground-based system before it can be adopted. Furthermore, the actual trajectory is recorded and transmitted by the aircraft, so that the ground-based systems can track it as an object. The accumulation of this airspace data allows traffic density predictions to be calculated, and dynamic route structure objects to be produced [4].

These objects – produced, consumed and manipulated by computers – may be modeled and even implemented through some Object Oriented Technology. There are languages that support these concepts and provide a direct way of manipulating them. The implementation of the free-flight initiative has still not addressed such issues. The FAA is evaluating the problems of Object Oriented Technology.

Object Oriented Technology

There is pressure from industry to use object oriented paradigms in the development of safety critical software. The expectation is that, as in other industry sectors, such programming will lower the development costs. There is some reluctance by regulators to approve this type of programming as it introduces concepts of information hiding, polymorphism, and inheritance. This makes the coupling between code and data less obvious to an auditor. It may invoke run-time support code that creates and destroys these objects dynamically, depending on the scope of the objects during execution. The timing and resource usage of such run-time programs make the application less deterministic, complicating the analysis and approval of such systems. It is expected that ultimately some compromise will be reached and a subset of the object oriented programming paradigm will be adopted, thereby satisfying the concerns of determinism and providing the benefits of this new technology.

Conclusion

Although a number of challenges remain, the industry is very focused on safe air transportation. It is through tremendous vigilance and determinism that the industry has a good safety record. It can be improved, and these on-going initiatives will contribute to safer flight.◆

References

1. DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA, Dec. 1, 1992.
2. AC 25-1309-1A, Advisory Circular, Federal Aviation Administration.
3. DO-248A. Annual Report for Clarification of DO-178B. RTCA, Oct. 6, 1999. (DO-248B to be published in 2001.)
4. National Airspace System Concepts of Operations. RTCA, Dec. 13, 2000.

Resources

- For a complete listing of RTCA documents please see <www.rtca.org>.
- The FAA Flight Standards Service provides links to the regulatory Web sites at the following Web site <www.faa.gov/avr/afs/fars/far_idx.htm>.

About the Author

George Romanski has specialized in the production of software development environments for the past 30 years. Romanski was vice president of Technology at EDS/Scicon, vice president of Engineering at Alsys and director of Safety Critical Software at Aonix. Romanski also serves the safety-critical industry as a member of the HRG (Annex H Rapporteur Group) for the Ada95 ISO standard addressing safety and security issues as well as the Requirements and Technical Concepts for Aviation (RTCA)/SC-190 committee working to provide clarification of DO-178B for avionics and ground-based systems. Romanski is president of Verocel, a company specializing in the verification of software, and in the development of tools that help in this process.

Verocel, Inc.
234 Littleton Road, Suite 2A
Chelmsford, MA 01886
Phone: (978) 392-8860
E-mail: romanski@verocel.com

Avionics Modernization and the C–130J Software Factory

Richard Conn, Stephen Traub, and Steven Chung
Lockheed Martin Aeronautics Company

The rollout of the first production C-130 aircraft, the C-130A, took place on March 10, 1955. Since then, more than 2,100 C-130s have been built in dozens of variations and are flown by more than 60 nations worldwide. They carry troops, vehicles, and armaments into battle. They drop paratroopers and supplies from the sky. They serve as airborne and ground refuelers. They serve as flying hospitals, hurricane hunters, and provide emergency evacuation and humanitarian relief. They perform airborne early warning and maritime surveillance. They've worn skis in Antarctica and have helped recover space capsules. In May 1992, the 2,000th C-130, a C-130H, was delivered. In September 1992, formal development of the C-130J began. Unlike its predecessors, the C-130J is a software intensive system employing modern avionics that have made significant improvements in its performance. By March 2001, the C-130J flew with a complete compliment of mission computer software setting 50 world records. This article presents insight into Lockheed Martin's modernization of the C-130 airlifter family.

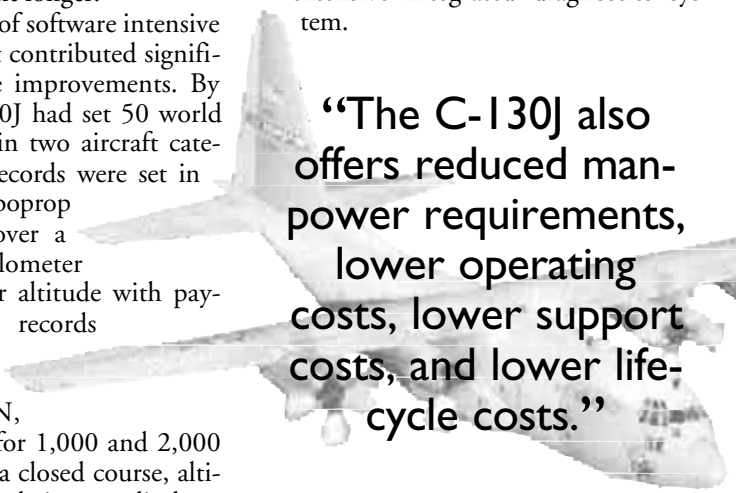
The C-130J looks like the earlier models, but it is really a brand new airplane with improved performance [1]. A key difference is that the C-130J is a software intensive system, where the earlier models were largely mechanical aircraft. Compared to the production C-130E, here are the C-130J improvements:

- Maximum speed is 21 percent greater.
- Climbing time is 50 percent less.
- Cruising altitude is 40 percent higher.
- Range is 40 percent longer.

The introduction of software intensive systems to the aircraft contributed significantly to all of these improvements. By June 1999, the C-130J had set 50 world aeronautical records in two aircraft categories. Twenty-one records were set in the Class C-1.N, Turboprop category for speed over a 1,000 and 2,000 kilometer closed course and for altitude with payload. The other 29 records were set in the Short Takeoff and Landing, Class N, Turboprop category for 1,000 and 2,000 kilometer speed over a closed course, altitude with payload, and time-to-climb to 3,000, 6,000, and 9,000 meters.

The C-130J also offers reduced manpower requirements, lower operating costs, lower support costs, and lower lifecycle costs. Here are the three key distinguishing features of the C-130J:

- A new propulsion system featuring four Full-Authority Digital Engine Control Allison AE2100D3 engines that generate 29 percent more thrust while increasing fuel efficiency by 15 percent.
- Advanced avionics technology featuring two holographic heads-up displays and four multifunctional heads-down Liquid Crystal Displays for aircraft flight control, onboard systems monitoring and control, and navigation; the displays are night vision imaging system compatible.
- Two mission computers and two backup bus interface units provide information flow and dual redundancy for the onboard systems, including an extensive integrated diagnostics system.



"The C-130J also offers reduced manpower requirements, lower operating costs, lower support costs, and lower lifecycle costs."

The C-130J family started with the 382J, a commercial aircraft that was created specifically to achieve Type Certification by the Federal Aviation Administration (FAA). FAA Type Certification was at Level A (the highest level) of the DO-178B standard. This milestone established that the C-130J family has complied with the safety critical requirements of the FAA should we later have a commercial customer. Once FAA Type Certification was achieved, the C-130J was derived from the 382J, establishing the military baseline software for all future variants of the aircraft. Each major version of software for the C-130J is called a block, and more than 96 percent of the 382J software (Block 2) was reused in creating the C-130J military baseline (Block 3). Ninety percent or more of the military baseline software (Block 3) has been reused so far for each variant of the aircraft (Block 4):

- Block 1: basic airworthiness software.
- Block 2: safety-critical 382J aircraft software.
- Block 3: military baseline of the C-130J aircraft software.
- Block 4: custom variants of the C-130J aircraft software.
- Block 5: Block Upgrade Program.
- Beyond Block 5: Hercules Improvement Plan for software/systems will address future C-130J upgrades as a continuous process and product impr-ovement activity and to address new and changed customer needs.

Each block provided a foundation of reusable software for the following blocks. As of March 2000, our level of software reuse typically exceeded 90 percent for most of our products:

- Block 3 military software baseline - 96 percent reused from Block 2.
- Block 4 software for the Royal Air Force - 95 percent reused from Block 3.
- Block 4 software for the Royal Australian Air Force - 95 percent reused from Block 3.
- Block 4 software for the United States Air Force - 97 percent reused from Block 3.
- Block 4 software for the Italian Air Force - 90 percent reused from Block 3.
- Block 4 software for the Tanker vari-

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.

ant - 90 percent reused from Block 3.
- Reuse on the C-27J aircraft, the C-5 Aircraft Modernization Program, and proposed for the C-130 Aircraft Modernization Program is yet to be measured but is expected to be equally high.

The first flight of the C-130J was April 1996 with a minimum of onboard software. The C-130J flew with a complete mission computer software suite (Block 5.3) in March 2001. The new software is expected to be installed in the deployed worldwide fleet of C-130J aircraft during a one-year period beginning the summer of 2001 after Air Force qualification testing is completed at the Air Force Flight Test Center at Edwards Air Force Base.

Plans for reuse of C-130J software and technology were laid out during the early days of the software development effort. The C-130J's advanced avionics technology and mission computer software are already being reused in the C-27J aircraft, the C-5 Aircraft Modernization Program, and Lockheed Martin's proposed Joint Strike Fighter. C-130J avionics and software reuse has also been proposed for the Lockheed Martin's C-130 Aircraft Modernization Program that is intended to incorporate newer technology into the older C-130 aircraft in the fleet.

The C–130J Aircraft as a Software Intensive System

The C-130J aircraft is an integrated collection of software systems produced by more than 25 suppliers. These systems, which are developed in compliance with the Lockheed Martin C-130J Tier I Software Development Plan, are integrated with the devices on the aircraft such as the engines, pneumatics, flight station displays, and the radar. A common Tier I Software Development Plan helped to enforce commonality between all the suppliers, making integration of their products into the air vehicle easier.

The Lockheed Martin C-130J Software Integrated Product Team develops the air vehicle and ground-based data system software also in compliance with the Tier I plan. Thus Lockheed put the same commonality requirements on itself as it did its suppliers. All suppliers, including Lockheed, produced their own Tier II Software Development Plans per directions in the Tier I Software Development Plan.

The air vehicle software consists of the Mission Computer (MC) Operational Flight Program (OFP) and Bus Interface Unit (BIU) OFP. The MC OFP manages the overall software operations within the C-130J aircraft and executes within a normal or backup mode. Both modes of the MC OFP include the primary roles of maintaining a central database, providing executive control for all software functions, providing interfaces to the MIL-STD-1553 data buses, and performing fault detection/fault isolation.

The BIU OFP operates in conjunction with the MC OFP in performing the integration of the C-130J avionics. The BIU OFP operates within a normal mode or an MC backup mode. The primary roles of the BIU OFP during normal mode operations are monitoring health, storing and validating critical data, and providing interfaces to non-MIL-STD-1553B data sources. The primary roles of the BIU OFP during MC backup mode operations include acquiring the role as bus controller and performing critical functions.

The ground-based data system software includes the Ground Maintenance System (GMS) and the Organizational Maintenance System (OMS). The GMS is a ground-based computer system that provides a central database for maintaining line-replaceable unit (LRU) configuration information and archived aircraft history for each tail number in the C-130J fleet or squadron. The GMS processes the maintenance-related data recorded to on-board removable memory modules on the C-130J aircraft.

The GMS provides an automated or manual flight crew maintenance debrief function and reads data stored on the removable memory module. The GMS validates the downloaded data, runs automatic fault isolation routines, calculates health and usage parameters, and generates maintenance work orders as required. The system processes structural and engine data to monitor component life and supports configuration control and status reporting of the air vehicle. The GMS maintains a variety of printed reports to support aircraft maintenance. The GMS is also hosted on the Portable Maintenance Aid, which is loose equipment for each C-130J. This capability is provided to support the need to forward deploy the aircraft for operations away from its home base.

The OMS provides the user interface between the maintainer and the C-130J aircraft systems for performing organizational level maintenance on the aircraft. The OMS supports the maintainers by accessing electronic technical orders, troubleshooting aircraft failures, evaluating status of aircraft systems, checking configuration of aircraft systems, and uploading and downloading files to and from the aircraft systems. The GMS interfaces with the OMS for maintenance work order processing, status reporting of maintenance actions performed, and recording of diagnostic data during ground maintenance.

The Software Factory

In the culture of our aircraft manufacturing facility, software is a part on the aircraft, tracked just like the engines, pneumatic systems, and radar systems. The C-130J Software Integrated Product Team operates a software factory that produces the air vehicle and ground-based data system software parts and approves the software parts for all computerized devices on the aircraft. The air vehicle software parts are written in Ada (250,000 lines of code), and the ground-based data system software parts are written in C++ and a fourth generation language (400,000 lines of code total) for each aircraft. Each software part has a part number, a set of associated drawings, and an assembly (such as a removable memory module). The drawings associated with each software part include the following:
- Software Item Drawings assign a unique part number to each computer software configuration item that is 1) installed on the aircraft, 2) used to create or prepare a part for aircraft installation, or 3) used to install or transfer a software item into an aircraft part. The notes on each Software Item Drawing describe 1) the host hardware part number, 2) the image file names and software version identities or a reference to the document containing specific software configuration information (i.e. version description document), and 3) the software-to-software compatibility dependencies.
- Software Assembly Drawings are produced for each software assembly (integrated collection of software items). A Software Assembly Drawing describes 1) a software assembly used in the production of a deliverable part, or 2) a software assembly delivered to a customer. Software Assembly Drawings assign a unique part number to each release of each software assembly. The parts list in the Software Assembly Drawing describes the software items (by part number and location code) contained on the assembly and the specific media (i.e., 3.5-inch diskette, 4mm tape, etc.) of which the assembly is made. The notes on the Software Assembly Drawing describe

1) the configuration of any vendor-supplied software items (i.e., reference to Vendor's Version Description Document), 2) the specific software assembly instructions used to create the software assembly, and 3) the contents of the label placed on the completed software assembly.

- Software Assembly Instruction Drawings are produced for each deliverable software assembly. The Software Assembly Instruction Drawing describes the required hardware equipment, software environment, personnel, access privileges, and detailed procedures necessary to produce the software assembly.
- Software Installation Instruction Drawings are produced for each software item installed into a deliverable part. The Software Installation Instruction Drawing describes the required hardware equipment, software environment, personnel, access privileges, and detailed procedures necessary to install the software item(s) into the host hardware part.
- Software Index Drawings facilitate the identification of customer deliverable software on each aircraft model, thus allowing the software design organization to control interim software releases to production aircraft without changing the master index for production software releases that are not delivered to a customer.
- Software Control Drawings are produced for each C-130J customer. The Software Control Drawing details the software and hardware combinations delivered to each customer. The body of the Software Control Drawing contains the following information for each deliverable software item: 1) find number, 2) software description, 3) identification of the software manufacturer, 4) software part number, 5) software version identity, 6) the aircraft model, version, serialization usage of the software/hardware combination, 7) note references, 8) hardware description, 9) identification of the hardware manufacturer, and 10) the host hardware part number. Notes in the Software Control Drawing describe: 1) which software items are loadable in the field and 2) any software compatibility/usage limitations.

The people who work in the C-130J Software Factory are collectively called knowledge workers, and they serve in many distinct roles such as software product managers, software requirements engineers, software development engineers, software test engineers, software process engineers, software quality assurance specialists, and documentation specialists. These knowledge workers are tied together through a digital nervous system (DNS), a term coined by Bill Gates of Microsoft [2]:

> "A DNS comprises the digital processes that closely link every aspect of a company's thoughts and actions. Basic operations such as finance and production, plus feedback from customers, are electronically accessible to a company's knowledge workers, who use digital tools to quickly adapt and respond. The immediate availability of accurate information changes strategic thinking from a separate, stand-alone activity to an ongoing process integrated with regular business activities."

Reuse

Software reuse has been at the heart of the C-130J Software Factory since development of the C-130J aircraft began in 1992. The program started with domain analysis and engineering, looking at what could be reused from other programs, defining the domain of the C-130J, and creating reusable assets that have been exploited throughout the program. The cost of developing air vehicle and ground-based data system software is the primary reason for Lockheed's aggressive efforts to achieve real, effective reuse. Reuse has significantly lowered the life-cycle cost and program risk.

Many products of the C-130J Software Factory were designed from the beginning to be reusable:

- Template-Based Design: Six domain-specific design patterns were originally created to serve as class definitions for all device interfaces to the MC OFP and the BIU OFP. Since 1992, three more design patterns were created to address new technology transition, bringing the total to nine design patterns. Courseware was prepared to document these design patterns and teach newcomers how to use the patterns. The productivity gains, improved reliability, and reduced testing overhead provided by applying template-based design were observed throughout the development of the software.
- Source Code: For many device interfaces, source code used for other device interfaces could be reused with very minor modification. In addition, source code from previous blocks could be reused extensively on later blocks (note the reuse figures between Blocks 2 and 3 and Blocks 3 and 4, see page 19).
- Test Scripts: Due to the definition of the classes of device interfaces, test scripts could also be reused. Requirements-based testing also helped by supporting automated generation of test cases directly from the requirements specifications.
- Documentation: Delivered and internal documentation was designed to be reusable, facilitating its production from one software build to the next.
- Software Development Domain Specific Kits (DSKs): Commercially-available DSKs, such as Microsoft Visual Studio .NET and Microsoft Visual Basic for Applications, greatly enhance productivity. We also employ homegrown DSKs, such as our Data Collection System Version 3, which is a DSK designed to build data collection applications.
- Common Software Development Tools: Our Environment and Tools Working Group establishes a set of common software development tools, such as Rational APEX and Cadre Teamwork for use on several Lockheed Martin programs. We save cost in terms of both purchase price and training, and we gain by having more readily interchangeable personnel. Reuse is also enhanced in that tool-specific conversions are reduced or eliminated should an asset produced by one program be adopted by another.
- Domain Knowledge: Knowledge captured during the early domain analysis and engineering activities was stored in courseware, reusable as a teaching instrument throughout the life of the program.

Challenges

The C-130J aircraft denotes a cultural change in a significant part of a major corporation from producing largely mechanical aircraft to producing software intensive aircraft. Such a change takes time for the culture to adapt, and there are many challenges that both the management and technical communities within that culture must face. These are the challenges faced by the C-130J Software Integrated Product Team:

- Building safety critical, high integrity [3] software for an aircraft with corporate funding (the development of the

Statistic Tracked	1998	1999	2000
Number of changes processed	2,430	2,350	2,115
Number of engineering software builds	240	300	330
Number of software qualification tests	79	85	81
Number of pages of documentation produced	472,500	564,200	531,010
Number of software tests executed	700,450	798,683	751,700
Test success percentage	98.27%	98.75%	99.00%

Table 1: *Modern Avionics in the C-130J has Contributed to its Improved Performance*

C-130J was done without funding from external sources, such as the United States government), the corporate investment and risk were high.

- Reducing risk and life-cycle cost for a software intensive system with a 30-year life span by achieving effective software reuse.
- Designing a software intensive system that is adaptable to changing technology during a 30-year life span.
- Meeting the requirements of FAA Type Certification.
- Controlling changes and software versions in light of thousands of requirements against multiple baselines for multiple customers, and creating different builds for different customers concurrently – satisfying the needs of a diverse group of customers, each with their own unique requirements during a 30-year life span.
- Achieving Capability Maturity Model® Level 3 and ISO 9001 certifications and continuing the investment needed to maintain these certifications.

From a broad perspective, the challenges may be grouped into four areas: software reuse, process, certification (for CMM Level 3, ISO 9001, and the FAA), and culture. Within the domain of our company (aircraft development and manufacturing), these challenges were addressed from the point of view of the pre-software intensive culture that was already in place:

- Software reuse was one of the easier challenges to address. The concept of line replaceable units (LRUs) was already in management's minds from a hardware perspective, so adding software parts as LRUs was not a significant leap. Neither was viewing those software parts as complex parts containing smaller component parts. Domain engineering was done at the beginning of the program, at a time when the development laboratories were not yet ready and the systems engineers were engaged in design and simulation. Ideas were also picked up from other existing aircraft programs, adding credibility to our domain engineering effort.
- Introducing a software process orientation was also an easier challenge to address. Management was already aware of manufacturing process concepts, so software development process concepts were not a significant leap in the early stages. A common Software Engineering Process Group was readily established to share ideas and infrastructure between the various software development Integrated Product Teams, such as the C-130J, F-22, C-5 AMP, and C-27J.

  The primary obstacle to our process definition efforts arose when management implemented a lean initiative to reduce waste in both the hardware and software processes. In the efforts to completely document the processes, it became evident how expensive a complete process description would be to produce. In describing our software development processes down to the level of following the trail of paper and electronic data between people's desks, the C-130J Software Integrated Product Team alone ended up with 114 distinct processes in a hierarchy that was three levels deep.

  This collection of process descriptions was a small part of the overall detailed process description for the development and manufacturing of the entire aircraft, which is currently incomplete and estimated to be between 3,000 and 5,000 distinct processes. The effort to create the detailed process description for the hardware side is continuing as we are moving to CMMI adoption.
- Certification activities were more challenging than software reuse and process. Our lean effort described in the previous bullet was a significant aid in our CMM Level 3 certification activities, and applying web technologies to describe our processes allowed us to present this information from the point of view of a CMM assessor, organized by Key Process Area and Key Practice. The introduction of automated data collection during the last three years has made it much easier to produce the evidence demanded by the CMM assessors, but gathering more than 300 artifacts for a CMM assessment is still a daunting task. The challenge of FAA Type Certification was similar to CMM Level 3 certification, and the ISO 9001 certification challenges fell nicely into place as our CMM Level 3 certification challenges were addressed.
- The cultural shift required by management to understand the issues and culture of the software engineers was our greatest challenge. Management expectations were originally high that software engineers could possess the same domain knowledge as systems engineers, and this was simply not the case. The mindset of someone with a master's degree in mechanical or electrical engineering, especially if that degree was granted more than 10 years ago, is fundamentally different from the mindset of a contemporary software engineer.

  Attempts were made to have systems engineers perform software engineering work – the success of these attempts was mixed. Over time, systems engineers and software engineers gradually came to understand each other's mindsets, but occasional personnel turnover disrupted this understanding; we found a continual need to reeducate engineers on both sides.

  Likewise, management's acceptance of software engineering concepts has been gradual, again requiring reeducation with personnel turnovers. After a decade, the three groups – management, systems engineering, and software engineering – still do not completely accept each other's mindsets. We expect this cultural difference to continue for some time to come.

The following statistics are noted in the more than 5 million source lines of code delivered to date: The C-130J software has been built for a 30-year life span. A lot can change in terms of the demands placed on the C-130J aircraft and its mission during these many years. Incorporation of a Global Air Traffic Management system and a comprehensive software maintenance plan are two of the efforts currently underway, and software production is continuing with a projection of more than 9 million lines of code delivered by the end of 2001. New missions,

different requirements from new customers, changing requirements from existing customers, and the introduction of even newer technology to the aircraft are the key factors causing this software growth. Continual process improvement, particularly through the C-130J Digital Nervous System, is underway, and increasing levels of capability maturity, through CMM Level 4 to Level 5, are planned.

Lessons Learned

Many lessons were learned during the last decade of the C-130J software development. Here are some key lessons:

- Objectives and requirements must be nailed down specifically from the beginning. It is never possible to get the requirements right the first time if the problem is of any significant degree of complexity. Requirements traceability and requirements grading are required. Conduct software product evaluations on requirements as intensely as you would review the code.
- You can never have too many simulations or laboratory resources.
- Software engineering capability maturity is not enough by itself to improve the quality of an integrated system like an aircraft. Systems engineering and management capability maturity are also required.
- Driving a product by schedule is unavoidable. Be prepared to deal with it and be prepared to adapt when the schedule slips. Define all your processes and measure their performance. Remember that the last process in the sequence is not necessarily the source of the problem when a schedule slips.
- Automate testing as much as possible. Always plan on running a test again. Always base test cases on requirements, trace test cases to those requirements, and employ automated tools to build your test cases from your requirements specifications when possible.
- Successful reuse requires a significant up-front cost and an effective, compelling producer/consumer model that makes it economically viable. Management must see reuse values and accept the costs as well as the benefits.
- Measurement comes with capability maturity, but no measurements can replace the in-depth, detailed knowledge of the people on the development line. Management must journey to the (software) factory floor before they can really understand the issues.◆

References

1. Lockheed Martin. C-130J Hercules Web site, <www.lmasc.com/c-130j/index.htm>.
2. Bill Gates. Business @ The Speed of Thought – Using a Digital Nervous System, Warner Books, 1999, <www.speed-of-thought.com>.
3. Sutton, James (Lockheed Martin), and Carre, B.A. (Praxis Critical Systems). Achieving High Integrity at Low Cost: A Constructive Approach, ERA 1995 Conference, London, United Kingdom.

About the Authors

Richard L. Conn has more than 20 years experience in software engineering and project management. Conn is currently the software process engineer for the C-130J Airlifter at Lockheed Martin Aeronautics Company. He graduated with bachelor's and master's degrees in computer science from Rose-Hulman Institute of Technology in 1976 and the University of Illinois in 1978, respectively. Conn was an Army officer from 1978-82 at the Army's Satellite Communications Agency and the Air Force Institute of Technology, where he taught computer science. Conn was a member of the Federal Advisory Board for Ada and a distinguished reviewer of the Department of Defense's Software Reuse Technology Road Map.

Lockheed Martin Aeronautics Company
86 South Cobb Drive
Dept. 70-D6, Mail Zone 0674
Marietta, GA 30063-0674
Phone: (770) 494-1670
Fax: (770) 494-1345
E-mail: richard.l.conn@lmco.com

Stephen M. Traub has more than 20 years experience in software engineering and project management. Traub is currently the software designated engineering representative at Lockheed Martin Aeronautics Company on behalf of the Federal Aviation Administration. Graduating from Elon University in North Carolina in 1984, Traub worked for Unisys from 1980-1984 as the principal software engineer for Weapons Assignment tasks for several Navy shipboard systems. He has been at Lockheed Martin since 1984, first working on the C-5B aircraft, and then working on the C-130J in the roles of Mission Computer Software Development lead, software product manager, and Software Integrated Product Team lead.

Lockheed Martin Aeronautics Company
86 South Cobb Drive
Dept. 70-D6, Mail Zone 0674
Marietta, GA 30063-0674
Phone: (770) 494-1670
Fax: (770) 494-1345
E-mail: stephen.m.traub@lmco.com

Steven J. Chung has 18 years of experience in software engineering and project management. Chung is currently the Software Integrated Product Team lead for the C-130J Airlifter at Lockheed Martin Aeronautics Company. Graduating from the University of South Florida in 1983, he worked for Honeywell Space Systems as a software engineer on the Space Shuttle and the Advanced Space Communications Technology programs and E-Systems on a real-time communications network. Chung came to the C-130J program at Lockheed in 1996 as a staff engineer and was promoted to Software Integrated Product Team lead in 2001.

Lockheed Martin Aeronautics Company
86 South Cobb Drive
Dept. 70-D6, Mail Zone 0674
Marietta, GA 30063-0674
Phone: (770) 494-1670
Fax: (770) 494-1345
E-mail: steven.j.chung@lmco.com

Software Technology Support Center

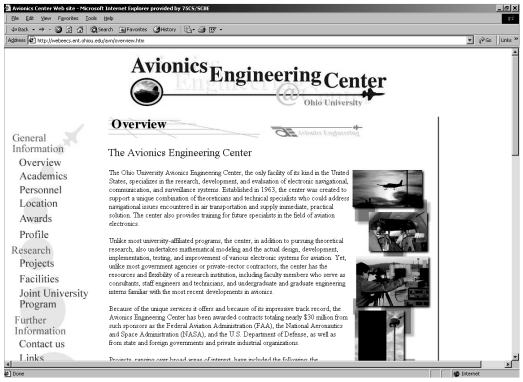
www.stsc.hill.af.mil

The Software Technology Support Center is the command focus for proactive application of software technology in weapon, command and control, intelligence and mission-critical systems. It helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability.

The Defense Advanced Research Projects Agency

www.arpa.mil

The Defense Advanced Research Projects Agency (DARPA) is the central research and development organization for the Department of Defense (DoD). It manages and directs selected basic and applied research and development projects for DoD, and pursues research and technology where risk and payoff are both very high, and success may provide dramatic advances for traditional military roles and missions.



Avionics Engineering Center

webeecs.ent.ohiou.edu/avn/overview.htm

The Avionics Engineering Center (AEC) at Ohio University is a unique research organization specializing in aviation research. For more than 37 years, AEC has been active in communications, navigation and landing systems, and surveillance research for the Federal Aviation Administration, NASA, and Department of Defense. The center specializes in the research, development, and evaluation of electronic navigational, communication, and surveillance systems. It also undertakes mathematical modeling and the actual design, development, implementation, testing, and improvement of various electronic systems for aviation.

Federation of American Scientists

www.fas.org

The Federation of American Scientists conducts analysis and advocacy on science, technology, and public policy, including national security, nuclear weapons, arms sales, biological hazards, secrecy, education technology, information technology, energy, and the environment. FAS is a privately-funded nonprofit policy organization whose Board of Sponsors includes 58 of America's Nobel laureates in the sciences.

National Aeronautics and Space Administration's Aerospace Technology Enterprise

www.aero-space.nasa.gov

National Aeronautics and Space Administration's Aerospace Technology Enterprise Web site outlines its goals and objectives to providing direction for its Enterprise programs. This includes new technologies, systems, and models for air and space transportation operations. The agency's goals are to revolutionize aviation, advance space transportation, and pioneer technology innovation and commercial technology.

American Institute of Aeronautics and Astronautics

www.aiaa.org

Today, with more than 31,000 members, the American Institute of Aeronautics and Astronautics (AIAA) is the world's largest professional society devoted to the progress of engineering and science in aviation, space, and defense. The Institute continues to be the principal voice, information resource, and publisher for aerospace engineers, scientists, managers, policy makers, students and educators.

Air Force Technology

www.airforce-technology.com

The Air Force Technology Web site provides international coverage of bombers, fighters, surveillance and patrol aircraft, training aircraft, attack, support, and naval helicopters. It maintains a listing of Military Aerospace Products and Services, an alphabetical listing of Military Aerospace Contractors and Suppliers, and Defense Industry Exhibitions, Conferences, and Events.

Practical Software Measurement, Performance-Based Earned Value

Paul Solomon
Northrop Grumman Corporation

Successful software project management can be achieved by focusing on requirements, selecting the most effective software metrics, and using Earned Value Management. Best practices and lessons learned by the Northrop Grumman team in developing weapons system software for the B-2 Stealth Bomber are discussed.

This article discusses a set of integrated, performance measurement techniques that increased Northrop Grumman Corporation's software success. These techniques can enable excellent project management in the following ways:

- Defining effective metrics for sizing the project and measuring progress.
- Using Earned Value Management (EVM) as the key, integrating tool for control.
- Defining quality goals in terms of project milestones and metrics.
- Planning for incremental releases and rework.
- Revising the plan for deferred functionality and requirements volatility.
- Focusing on requirements, not defects, during rework.
- Using testable requirements as an overarching progress indicator.

These techniques are based on the following industry and professional standards:

- CMU/SEI-92-TR-19, Software Measurement of Department of Defense Systems.
- "A Guide to the Project Management Body of Knowledge (PMBOK)," Project Management Institute, December 2000.
- Practical Software and Systems Measurement: A Foundation for Objective Project Management (PSM) [1].
- (ANSI/EIA)-748-98, EVM Systems Standard (Standard), American National Standards Institute/ Electronics Industry Association.

The techniques evolved from lessons learned and continuous process improvement during development of embedded weapons system software for the U.S. Air Force B-2 Stealth Bomber and other programs at Northrop Grumman Corporation's Air Combat Systems (ACS), a business area of the company's Integrated Systems Sector. The ACS software organization achieved Level 4 using the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM) in 1998 and has a goal of achieving Level 5 in 2001.

The essence of EVM, per the Standard, is that at some level of detail appropriate for the degree of technical, schedule, and cost risk or uncertainty associated with the program, a target value (i.e., budget) is established for each scheduled element of work. As these elements of work are completed, their target values are earned. As such, work progress is quantified and the earned value becomes a metric against which to measure both what was spent to perform the work and what was scheduled to have been accomplished. The combination of advance planning, baseline maintenance, and earned value analysis yields earlier and better visibility into program performance than is provided by non-integrated methods of planning and control.

> "The essence of EVM ... is that at some level of detail appropriate for the degree of technical, schedule, and cost risk or uncertainty associated with the program, a target value ... is established."

Improvements in Opportunities

Despite being at SEI CMM Level 3 since 1995 and having a validated EVM system, the software development organization was not consistently achieving its objectives and customer expectations. More importantly, the management control system was failing to accurately report project performance. These issues were identified and disclosed in the quality audits of the EVM organization. In 1996, an audit defined the following issues and goals:

> "The process for managing software projects with regard to baseline planning, determination of schedule milestones, and earned value could be improved to provide better milestones and metrics for interim performance measurement during development, testing, and rework … actual progress against the total technical requirements is not displayed on a schedule in relation to a plan, a projected or actual software release on a schedule may not reflect completion of all effort originally planned, and earned value does not necessarily represent the percentage of completion of the total statement of work.
>
> It is recommended that the Software Engineering Process Group (SEPG) be empowered to develop a better process for measuring and reporting progress on software projects. It is recommended that the following topics be addressed:
>
> - Criteria for determining which planned requirements are significant for tracking progress.
> - Criteria for milestone definitions.
> - Earned value and internal replanning for deferred requirements or functionality.
> - Earned value and internal replanning for revised requirements.
> - Planning and measuring progress during rework phase."

Existing Measure Shortcomings

A team was formed to review existing measures and processes. It found that, although they adhered to company policies and relevant quality standards, including the SEI core set of software measures (size, effort, schedule, and qual-

ity), the measures used were not effective for technical progress.

The first finding was that during the initial coding phase, the most common sizing measure was source lines of code (SLOC). SLOC was utilized as a sizing measure as the basis for budgets and for earned value using a percent of completion method. However, the analysis concluded that there is usually a significant error in estimating SLOC. Consequently, any progress metric based on SLOC, including EV, was highly volatile. For example, all projects reviewed had software components that experienced multiple, significant increases in estimated SLOC. When the new estimate was first used as a denominator for percent complete, then negative progress and earned value were reported.

Second, while the schedule metrics and procedure discussed completion milestones, the milestone definitions and completion criteria lacked quantifiable objectives. Normally, an incremental build is released despite not incorporating all the planned functional requirements. It had been practice to display a completed milestone on the schedule and to take all of the earned value that was budgeted for that milestone without disclosing that not all the base-lined requirements or functionality had been achieved.

Third, the process review disclosed a deficiency regarding product quality measures such as defects found and closed. A manager normally uses a burndown curve of defects or trouble reports and tends to focus on eliminating defects rather than attaining requirements. Earned value had been based on the burndown plan. However, because the presence of defects indicates the failure to meet requirements, measures of defects are not the best measures of progress. Also, any measure of progress based on defects is unstable because the number of defects discovered during reviews and testing is always different than planned, and removal of one defect often results in detection of new ones. There were no metrics to track progress toward meeting all system requirements. As a result, progress measured as the ratio of defects removed to total estimated defects was a more volatile measure than the percent of SLOC completed. Also, there was no budget to enable earned value for the remaining work.

Fourth, the schedule and performance measurement baseline had been predicated on a discrete number of software builds but completion of the project often required many additional builds. However, earned value was taken as originally budgeted when builds were completed. As a result, there was no remaining budget for the additional builds and the cost performance reports overstated schedule status.

Practical Software and Systems Measurement

A good source of metrics is the Practical Software and Systems Measurement (PSM) guide. PSM provides project and technical managers with the quantitative information required to make informed decisions that impact project cost, schedule, and technical performance objectives. PSM is applicable to the overall planning, requirement analysis, design, implementation, and integration of systems and software activities. It provides a process to collect and analyze data at a level of detail sufficient to identify and isolate problems. This data includes estimates, plans, changes to plans, and counts of actual activities, products, and expenditures. The unit level (as defined by the product component structure or system architecture) is the most commonly used level of detail.

Resultant Process Improvements

The set of process improvements had five components:

- Developing the Performance Measurement Baseline (PMB).
- Requirements decomposition and traceability.
- Planning for defects and rework.
- Selection and use of software metrics.
- Performance-Based Earned Value.
- Revisions to plan for deferred functionality.

Performance Measurement Baseline

EVM begins with defining the project's product and management objectives. These technical, schedule, and cost objectives are transformed into a PMB schedule and budget baseline (also commonly called a Work Breakdown Structure). The team developed standard templates for the PMB. The templates ensured knowledge transfer and inclusion of common project components such as key schedule constraints, subcontractor control milestones, and systems engineering activities.

Requirements

During the requirements phase, high-level requirements are defined and decomposed to the levels needed to govern the design, implementation and integration, and test phases. Establishing a time-phased requirements baseline against which progress can be consistently measured is the most important EVM step. It drives the project sizing, the resource forecast (budget), and the schedule. The technical requirements also establish the criteria for completing tasks. The output of the requirements phase defines the criteria or attributes for completing significant milestones, or taking earned value in all subsequent levels and stages of development. Of equal importance are a disciplined requirements traceability process and a requirements traceability data base.

To ensure the acceptance of the end product and enable consistent performance measurement, allocated requirements should be testable and traced to detailed specifications, software components, and test specifications. Per PSM, some requirements may not be testable until late in the testing process, others are not directly testable, and some may be verified by inspection.

Dr. Peter Wilson, of Mosaic, Inc., discusses the utility of testable requirements [2]. Per Dr. Wilson, a testable requirement is one that is precisely and unambiguously defined, and one for which someone must be able to write a test case that would validate whether or not the requirement has or has not been implemented correctly. The number of testable requirements may be very different from the number of test cases. There are a number of reasons for this:

- A testable requirement may require more than one test case to validate it.
- Some test cases may be designed to validate more than one testable requirement.
- Testable requirements appear to have the granularity and flexibility to make earned value a practical tool for software developers.

To redirect management focus on meeting requirements, a Systems Engineering (SE) process improvement team rewrote the SE procedures. The new procedures mandated requirements traceability and the use of technical performance measures (TPM). Per the procedure, "TPMs are used to plan and track key technical parameters throughout a development program," and "to the maximum extent practical, earned value, both planned budget and earned value taken, should be based on those TPMs that best indicate progress towards meeting the system requirements." The procedure also requires verification of the testability of

requirements.

The time-phased plan for each project phase and each build should include milestones that objectively define the functional content to be achieved at that point. The milestones should be defined in terms of incremental functionality, both the number of testable requirements and the functional capabilities to be achieved. An incremental milestone is normally defined as 100 percent of the requirements needed to achieve a functional capability. However, for earned value purposes, it can also be targeted as a lesser percentage. The functionality targets should be documented as part of the criteria for completing the milestone and taking objective earned value. The percentage target is normally related to the targeted quality, as measured by defects.

Planning for Defects and Rework

In planning for incremental builds, the Statement of Work for all builds subsequent to the first should include an estimate for rework of requirements or code to fix defects that were found in previous builds but will be fixed in subsequent builds. To ensure adequate budget and period of performance, the planning assumptions should include a planned rate or number of defects to be found in each build, and a plan to fix these defects within the rework Statement Of Work of each build. Furthermore, rework should be planned in separate work packages. Failure to establish a baseline plan for rework and to accurately measure rework progress caused many projects to get out of control.

The team's remedy was to change the EVM procedure. The procedure requires that rework is planned in separate work packages from the initial development effort and that objective metrics for rework are used for earned value.

Selection and Use of Software Metrics

For tracking progress against a plan using EVM, the most effective measures are those that address the issues, product size and stability, and schedule and progress. Three measurement categories are mapped to these issues: functional size and stability, work unit progress, and incremental capability.

The specific measures to be discussed are requirements, requirements status, component status, test status, increment content-components, and increment content-functions.

Issue: Product Size and Stability. (Category: Functional Size and Stability) The requirements measure counts the number of requirements in the system or product specification. It also counts the number of requirements that are added, modified, or deleted and provides information about the total number of requirements and the risk due to growth and/or volatility in requirements.

When incremental builds are planned, this measure is also the basic component of the measure, increment content-function, as discussed below.

Issue: Schedule and Progress. (Category: Work Unit Progress.) The recommended measures for Work Unit Progress are requirements status, component status, test status, increment content-components, and increment content-functions.

- *Requirements Status:* The requirements status measure counts the number of requirements that have been defined and allocated to software components, allocated to test cases, and successfully tested. When used to measure test status, the measure is used to evaluate whether the required functionality has been successfully demonstrated against the specified requirements. Some requirements may not be testable until late in the testing process. Others are not directly testable or may be verified by inspection.

  This measure is ideal for EVM because it is objective. The budget allocated to requirements may be equally distributed or weighted according to the estimated effort for those requirements. Consequently, requirements-based EVM, as the integrating tool for technical, schedule, and cost objectives, provides Northrop Grumman's best measure of project status, progress, and remaining effort. Since implementing requirements-based EVM, program progress has never been significantly overstated and the management control system has provided more reliable data and earlier warning of program problems (see Table 1, page 28).

  Tables 1 through 5 (see page 28) are abstracts of measures that are fully described in PSM. Per PSM, there are three aggregation structures to accumulate measurement data. Tables 1 and 2 are component-based and functional-based aggregation structures.

  Component-based aggregation structures are derived from the relationship of the system components within a particular architecture or design. For projects that implement an incremental development approach, lower-level components (such as units and configuration items) are usually mapped to the incremental delivery products as part of the aggregation structure.

  Functional-based aggregation structures define the functional decomposition of system requirements. They are often mapped to the system design components. If they are mapped to design components, then measures of the requirements (such as the number of requirements tested) can be aggregated and evaluated for a particular function.

  The data collection level describes the lowest level at which data is collected to allow problems to be isolated and understood. It can then be rolled up using the aggregation structure.
- *Component Status:* The component status measure counts the number of software components that complete a specific activity. An increase in the planned number of components may indicate unplanned growth and cost impacts. However, the number of components, although not constant, is the perpetual denominator for measuring percent complete.

  In the initial design phase for EVM, a unit of measurement should be selected based on the design standards and practices employed for each build. This may be modules, packages, pages, or another appropriate component.

  Completion of components during the design and implementation phases should be based on component reviews, inspections, walkthroughs or specified tests, as appropriate (see Table 2 page 28).
- *Test Status:* The test status measures count the number of tests attempted, executed to completion, or completed successfully. It can be applied for each unique test sequence, such as component, integration, system, and regression test and is a good basis for earned value (see Table 3, page 28).

Issue: Schedule and Progress. (Category: Incremental Capability.) Incremental capability measures count the cumulative functions or product components with a product at a given time. An increment is a predefined group of work units, functions, or product components delivered to the next phase of development. These measures determine whether the capability is being developed as scheduled or delayed to future deliveries. There are two measures of increment content.

- *Increment Content-Components:* The increment content-components measure identifies the components that are included and assembled into increments. Increment content is often deferred to preserve the scheduled delivery date. When this occurs, it is essential to quantify the deferred content in terms of earned value and to annotate the schedule to indicate that the true status and the expected completion date of the base-lined work (see Table 4).
- *Increment Content-Functions:* The increment content-functions measure is preferred for schedule progress and for earned value because it directly maps to the number of functional requirements. It requires a formal, detailed list of functions and requirements by increment, as documented in the Requirements Traceability database (see Table 5).

The completion criteria for both increment measures are successful integration and successful testing, as described in Tables 4 and 5.

Tables 1-5: *Abstracts of Measures Fully Described in PSM*

Issue: SCHEDULE AND PROGRESS **Table 1**
Aggregation Structure: FUNCTION
Category: WORK UNIT PROGRESS
Measure: REQUIREMENTS STATUS
Typically Collected for Each: REQUIREMENTS SPECIFICATION

DATA ITEM	COMPLETION CRITERIA
# Requirements Traced to: • Detailed Specifications • Software Components • Test Specifications • Tested Successfully	• Completion of Specification Review • Baselining of Specifications • Baselining Requirements Traceability Matrix • Successful Completion of all Tests, in Appropriate Test Sequence

Issue: SCHEDULE AND PROGRESS **Table 2**
Aggregation Structure: COMPONENT
Category: WORK UNIT PROGRESS
Measure: COMPONENT STATUS
Typically Collected for Each: CONFIGURATION ITEM (CI) OR EQUIVALENT

DATA ITEM	COMPLETION CRITERIA

• Total # of Components
• # of Components Completed Successfully by Activity:
  • Preliminary Design
  • Detailed Design
  • Implementation
  • Component Test
  • CI Test
	• Component Reviews, Inspections, Walkthroughs
	• Successful Completion of Specified Test
	• Released to Configuration Management
	• Resolution of Action Items

Issue: SCHEDULE AND PROGRESS **Table 3**
Aggregation Structure: SOFTWARE ACTIVITY
Category: WORK UNIT PROGRESS
Measure: TEST STATUS
Typically Collected for Each: CONFIGURATION ITEM

DATA ITEM	COMPLETION CRITERIA
• Total # Test Cases • # of Test Cases Attempted • # of Test Cases Passed	• Successful Completion of Each Test Case in Appropriate Sequence

Issue: SCHEDULE AND PROGRESS **Table 4**
Aggregation Structure: COMPONENT
Category: INCREMENTAL CAPABILITY
Measure: INCREMENT CONTENT – COMPONENTS
Typically Collected for Each: CONFIGURATION ITEM (CI) OR EQUIVALENT

DATA ITEM	COMPLETION CRITERIA
• # of Components • # of Components Successfully Integrated	• Successful Integration • Successful Testing

Issue: SCHEDULE AND PROGRESS **Table 5**
Aggregation Structure: FUNCTION
Category: INCREMENTAL CAPABILITY
Measure: INCREMENT CONTENT – FUNCTIONS
Typically Collected for Each: FUNCTION OR EQUIVALENT

DATA ITEM	COMPLETION CRITERIA
• # of Functional Requirements • # of Functional Requirements Successfully Implemented	• Successful Integration • Successful Testing

Performance–Based Earned Value

The recommended software metrics for schedule and progress are also the basis for performance-based earned value (PBEV). PBEV is a lean, cost-effective means of implementing EVM to minimize administrative costs and to focus on the big picture. It results in less work packages to track, more emphasis on objective measures of technical performance related to achieving requirements, and less emphasis on tracking support activities. PBEV has the following characteristics:

- Emphasize key performance metrics and project progress relative to plan (schedule and budget), system requirements, and TPMs that support requirements.
- Maximize budget to key technical activities.
- Measure products and product components, not tasks and inch-stones.
- Use no EV for reviews, meetings, and recurring reports.
- Manage costs, not schedule of support tasks.
- Budget for support, from the level of effort tasks can be allocated, to discrete tasks to maximize focus on technical progress.

Plan for Deferred Functionality

To prevent the overstatement of progress and the premature consumption of budget, it is recommended that the increment content-functions measure is the primary basis for earned value during the design and integration and test phases.

To illustrate how deferred functionality should be quantified at the work package level, assume that a work package for implementation of code has release of a build as its completion milestone with a budget of 500 hours. Also, assume the build includes 100 testable requirements that are budgeted to require five hours each to implement. If the build was released with 90 requirements integrated,

then earned value would be 450 hours. The event of releasing the build short of its targeted functionality is cause to close the work package and replan the remaining work. In this case, transfer the deferred requirements and the residual budget of 50 hours to the work package for the next planned build. Place the budget in the first month of the receiving work package to preserve the schedule variance. If no planned builds remain, establish them through the normal internal replan process by closing the last work package and opening a new one for the next build with the unused budget.

Process Improvement Ups Customer Satisfaction

These practices have improved our management effectiveness and increased customer satisfaction. The *Air Force Acquisition Newsletter* cited our success as follows:

> "The B-2 Spirit Stealth Bomber Program implemented several innovative process improvements using EVM. These include integrating earned value with systems engineering processes, defining improved software engineering metrics to support EVM, and developing a leaner, more effective methodology called performance-based earned value (PBEV).
>
> These changes paid off during upgrades of the B-2 weapon system. One of those upgrades was the development of the Joint Standoff Weapon/Generic Weapon Interface System (JSOW/GWIS), a software intensive effort. The new metrics helped to make it a very successful program. The PBEV methodology was used to ensure that the warfighter received the most functionality from software development efforts. On JSOW, we provided 85 percent more capability than originally planned, on schedule and under budget [3]."

The most important business objectives of a best practice are increased corporate profit and customer satisfaction. Evidence of achieving these objectives is in the Air Force quarterly assessment report of the B-2 software maintenance contract. We received excellent award fee ratings for the year ended April 2001 in all categories: technical, program management, scheduling, and cost.

Conclusion

Using earned value to plan and manage software projects can prevent expensive failures. Earned value should be based on testable requirements and selected software measures that best underlay the plan and progress to achieve all project objectives. We are now revising our systems engineering process to incorporate lessons learned and improved processes from software development.◆

References

1. Practical Software and Systems Measurement: A Foundation for Objective Project Management. U.S. Department of Defense and U.S. Army. October 2000, Version 4.0b, available at <www.psmsc.com>.
2. P. B. Wilson. "Sizing Software with Testable Requirements." Journal of Systems Development Management. August 2000, reprint available at <www.testablerequirements.com>.
3. "Aerospace Acquisition 2000." Air Force Acquisition Reform Newsletter. Jan./Feb. 2000, Vol. 3, Number 1.

About the Author

Paul J. Solomon is the director, Earned Value Management Service on the B-2 Stealth Bomber program for Northrop Grumman Corporation's Air Combat Systems, a business area of the company's Integrated Systems Sector. He is on the board of the National Defense Industrial Association, Program Management Systems Subcommittee that authored ANSI/EIA-748. He was a member of the team that received the 1998 David Packard Excellence in Acquisition Award. He presented the concepts in this article at the 2001 Software Technology Conference and the 2001 SEPG Conference in India. Solomon holds MBA and BA degrees from Dartmouth College and is a Project Management Professional.

Northrop Grumman Corporation
3520 E. Ave. M, TD21/4B
Palmdale, CA 93550
Phone: (661) 540-0618
E-mail: solompa@mail.northgrum.com

STC 2002

Call for Exhibitors

Software Technology Conference • 28 April - 2 May 2002

"[STC] continues to bring in top quality attendees involved in IT purchasing decisions"

— STC 2001 Exhibitor

Exhibitor Registration open 1 August 2001

Exhibit Space is assigned on a first-come, first-serve basis. Submit your registration form TODAY!

To become a STC 2002 Exhibitor, visit

www.stc-online.org

- Online exhibit space registration
- Downloadable exhibit space registration form
- Complete trade show rules and regulations
- Current exhibit hall layout & exhibitor listings
- Online housing reservation system
- All other conference information

Questions?

Contact Trade Show Management at:
Phone: 435-797-0047
Fax: 435-797-0861
Email: **stcexhibits@ext.usu.edu**

Marketing and Networking Opportunities

- Endorsed by the Department of Defense as the premier software technology conference
- Interact with over 2,500 leaders and decision makers in the Department of Defense, Government, and Industry
- 60% of Trade Show hours throughout the conference week dedicated to exhibits only
- Informal Social held in exhibit hall
- All conference breaks served in the exhibit hall during open show hours
- 150-word organization description and logo listed on conference Web site
- Web site links
- Exhibit track presentations
- Literature kiosks located by show entrance
- One-day trade show guest passes for your distribution
- Pre-registered and post-conference participant lists
- Two evenings available for hosting hospitality events - prime hotel space available
- Discounted conference registration for badged exhibitors
- Full-service media room on-site

Aircraft, Software, COTS, GOTS, and a 1967 Chevy BelAir

I grew up an Air Force brat living in nine different houses before I turned 18. Some kids might feel that this constant changing of houses and schools was bad – but I loved it! Back in 1966, my dad was wrapping up a 2-½ year tour in Istanbul, Turkey. We had shipped a car over to Turkey with us, but sold it prior to Dad's transferring back to Sheppard Air Force Base. So before leaving, he ordered a car from Chevrolet (nothing online back then – catalogs, phone calls, and telegrams took care of it). We flew back to the United States in late December 1966. On Christmas Eve, Dad went down to the local Chevy dealer and picked up our 1967 Chevrolet BelAir. It was bright white, huge, and ours. I was 11 years old at the time, and I grew up with that car. I named it Cynthia.

I first learned to drive in Dad's Chevy. It had a manual transmission with the stick shift on the column. It also had an overdrive lever under the dash. It had a 283 cubic-inch engine and an oversized clutch. I not only learned to drive in that car, but being a one-car family, it was the car for my early (and very limited) adventures in dating.

Eventually, I moved out, joined the Air Force myself, and after 23 years retired from one career and moved on to another. I now live in Utah.

A few years ago my mom and dad decided it was time to sell the Chevy. They had become a two-car family in the 1980s, but by the mid-1990s didn't need two cars any more. My mom and dad asked if I was interested in owning the Chevy – I thought long and hard about it. I had always loved that car, and every time I visited my parents in Orlando, I would help Dad wash and wax Cynthia, and then take her for a test drive. Every so often, I would cruise in it up to Daytona Beach. I even knew just how to tune-up the engine. (Remember actual tune-ups with tachometers and dwell meters, points and a rotor to replace? Heck, the engine compartment was so big you could stand in it.) The car was indestructible on the outside – it even survived me learning to parallel park! I'm pretty sure that Mom and Dad would have given me a great price on Cynthia (in fact, had I asked, I am pretty sure I could have just talked them out of it). After almost 30-plus years of waiting, I could own a classic '67 Chevy BelAir, complete with antique license plates.

But, I decided to pass. I guess I was both older and wiser, even though the car was a classic – a 30-year old classic with no air conditioning, no power anything (brakes, steering, or windows), rear-wheel drive, and no air bags or shoulder belts. I would have had to drive it from Orlando to Kaysville, Utah – more than 2,300 miles. And once I had gotten it here, well, a rear-wheel drive car with no weight in the back is probably not the best vehicle to own during winter in the state with the "Best Snow on Earth." Not to mention the problems with trying to get parts for a 30 year-old car. So Mom and Dad sold the car to a friend in Alabama, where I am sure that Cynthia is still in action.

What in the world does that have to do with a column for CROSSTALK? This issue is about avionics modernization. Well, we have B-52 airplanes currently flying that first entered the Department of Defense (DoD) inventory back in 1955 – a life span so far of 46 years, and current engineering analyzes show the B-52's life span to extend beyond the year 2045[1]. Now put into perspective that the Wright Brothers first flew in 1903. Out of the 98 years that the world has known powered flight, B-52s have been flying for almost 50 percent of that time! If the B-52 flies until 2045, as projected, it will have been deployed for 90 years, almost 65 percent of flight's history.

We sometimes forget the life span of the hardware that our software drives. If you're working on avionics software now, can you imagine somebody trying to update (and debug) your software in the year 2091?

Face it, it takes a *long* time to write and update the software for avionics applications. In fact, one recent avionics system calculated that they averaged only about 0.4 lines of code per hour. That's one reason commercial off-the-shelf (COTS) and government off-the-shelf (GOTS) software are important nowadays. They are a lot better then purchasing one million lines of pre-written code and then only having to write 100,000 lines of "software glue" to make the COTS/GOTS work. You just saved more than 2 million person hours at 0.4 lines of code per hour. In addition, the advantages of extending the life span of existing aircraft can save the DoD billions of dollars.

So the next time you're driving a four- or five-year-old car and you feel that it's getting old, and you're itching for a new car, remember that the aircraft you supply software for might have a projected life span of 50-plus years. Sort of puts things in perspective, huh?

By the way, what did the 1967 Chevy BelAir have to with software or avionics? Not much – but it sure was fun reminiscing.

– David A. Cook, STSC
david.cook@hill.af.mil

1. http://www.af.mil/news/factsheets/B_52_Stratofortress.html

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
5851 F Ave.
Bldg. 849, Rm B04
Hill AFB, UT 84056-5713

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737