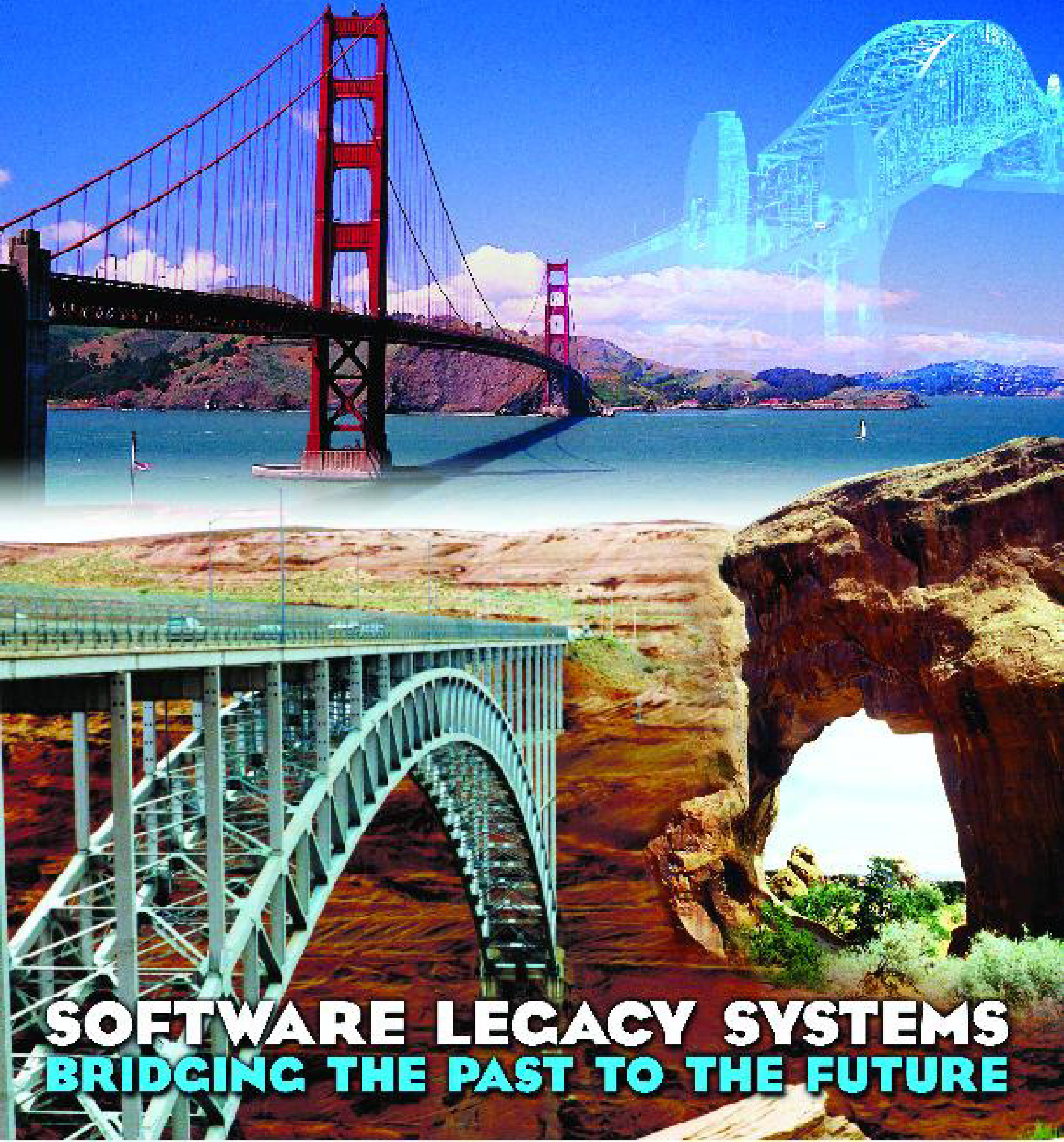
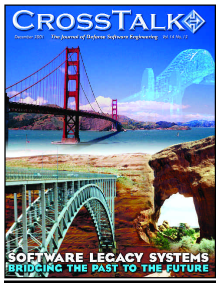
CrossTalk

December 2001 The Journal of Defense Software Engineering Vol. 14 No. 12



SOFTWARE LEGACY SYSTEMS
BRIDGING THE PAST TO THE FUTURE



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

CROSSTALK

SPONSOR	Lt. Col. Glenn A. Palmer
PUBLISHER	Tracy Stauder
ASSOCIATE PUBLISHER	Elizabeth Starrett
MANAGING EDITOR	Pam Bowers
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. 22.

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.

Stronger Than Ever

On behalf of the CROSSTALK staff, welcome to our final issue of 2001. As CROSSTALK concludes another publication year with its fourteenth volume, we remain committed to our mission of providing the defense software community with informative articles on software engineering best practices, proven methods, and lessons learned. We head into 2002 aiming to provide managers, practitioners, and users with the information they need to buy and build quality software – on time and within budget.

As we reflect back on this past year, it is the Sept. 11 attack on our nation that seems to cloud the year's events. We wish to express our sincere sympathy to those families and friends of the many victims. May our American pride remain strong and your loved ones never forgotten.

At CROSSTALK, 2001 was an exceptional year. As our annual index shows on pages 29-30, we published articles on a wide variety of software engineering topics that included Avionics Modernization, Distributed Development, Open and Common Systems, Testing and Configuration Management, and more. All of our 2001 issues are accessible from our Web site at <www.stsc.hill.af.mil>.

In this last issue of the year, we bring you information on upgrading and maintaining legacy system software. This topic is very important to the many organizations that regularly address how to migrate irreplaceable legacy software to modern platform-independent computing environments. Our issue begins with *Reengineering: An Affordable Approach for Embedded Software Upgrade* by Kenneth Littlejohn, Michael V. DelPrincipe, Jonathan D. Preston, and Dr. Ben A. Calloni. In this article, the Air Force Research Lab Information Directorate (AFRL/ID) and Lockheed Martin Aeronautics share what they have learned from the Embedded Information System Reengineering project that resulted in an automation-assisted JOVIAL-to-C reengineering capability.

Next, *The IULS Approach to Software Wrapper Technology for Upgrading Legacy Systems* is brought to us by Dr. David Corman, The Boeing Company, where the Incremental Upgrade of the Legacy Systems program has found that software wrappers play a major technological role for modernizing legacy systems. *A COTS-Based Replacement Strategy for Aging Avionics Computers* describes another AFRL/ID-sponsored effort that demonstrates a TRW-developed generic commercial off-the-shelf (COTS)-based software technology. Here Douglas G. Haldeman, William J. Cannon, and Jahn A. Luke explain how this scalable technology allows for the execution of legacy binary code on the latest generation of COTS microprocessors.

Our feature section concludes with *Automated Transformation of Legacy Systems.* This article by Philip Newcomb and Randy A. Doblar is a look at how artificial intelligence technology tools can be used for reengineering legacy computer languages into modern environments. Rounding out this issue, our supporting article this month is *Balancing Discipline and Flexibility With the Spiral Model and MBASE* by Dr. Barry Boehm and Dr. Daniel Port. See how this model and its recent extension can be used to tailor a project's balance of discipline and flexibility through risk considerations.

Finally, as we wrap up the year, we wish to provide a special thanks to all of our 2001 authors for contributing such informative articles. We also thank our many readers for their feedback and continued interest in our journal. We are excited about our upcoming January 2002 issue in which we will announce the Top 5 Government Software Projects. We had almost 100 entries, and what a tough choice it was to narrow this list down to five. Also, coming up in 2002, we will be featuring themes such as Capability Maturity Model®-Integrated℠, Requirements Risk, Software Estimation, and Information Assurance to name a few.

As CROSSTALK remains stronger than ever, may our nation also remain strong and prosper. Best wishes to you for a happy and healthy New Year.

Tracy L. Stauder

Tracy L. Stauder
Publisher

Reengineering: An Affordable Approach for Embedded Software Upgrade

Kenneth Littlejohn
Air Force Research Laboratory

Michael V. DelPrincipe, Jonathan D. Preston, and Dr. Ben A. Calloni
Lockheed Martin Aeronautics Company

Within the Department of Defense, embedded information systems found in aging aircraft are facing readiness, supportability, and upgrade challenges due to diminished manufacturing sources (DMS), which impacts both embedded processing hardware and software development tools. In many cases, however, the embedded software functionality within these systems is still highly viable. Under the Embedded Information System Reengineering project, the Air Force Research Laboratory Information Directorate and Lockheed Martin Aeronautics Company have matured a legacy software reengineering capability that eliminates software tooling DMS, permitting affordable application support and upgrade. A successful flight demonstration aboard a U.S. Air Force F-117 stealth fighter aircraft was recently conducted to verify correct performance of reengineered weapon system software components generated using high degrees of automation assistance.

Maintaining the viability of embedded information systems is a key technical and economic problem facing operators of aging aircraft. Within the Department of Defense (DoD), many currently fielded embedded information systems face readiness challenges imposed by evolving missions and extended service life spans. For example, emerging requirements for global situational awareness and rapid strike capabilities necessitate increased information processing and information exchange between command and control (C2) and weapon system platforms. However, the ability to overcome these challenges is constrained by such factors as shrinking budgets, limited computational capacity reserves, and the effect of diminished manufacturing sources (DMS).

Wholesale redevelopment is often cost prohibitive, particularly since large portions of embedded applications continue to fulfill mission requirements. In fact, most present-day upgrade programs involve incremental changes to an established design baseline, the majority of which is reused as is. Even when major functional overhauls are performed, budget and schedule realities usually dictate a phased approach. These realities underscore the need for an efficient means to carry forward, modernize, and exploit usable functionality within legacy software.

Solutions must maximize the recapture of prior design investments, provide efficient pathways for continued technology refresh, and accommodate changing technologies and economies of scale over decades-long service life spans. Purposeful migration toward insertion of commercial components mandates changes to existing business practices. The challenge, then, is to offer developers affordable methods of leveraging existing embedded information system applications to provide a foundation on which to base future systems. The Embedded Information System Reengineering (EISR) solution assumes that the end user is actively migrating to commercial hardware and operating systems.

The Air Force Research Laboratory Information Directorate (AFRL/ID) and Lockheed Martin Aeronautics Company have matured an integrated set of technologies that facilitate affordably maintaining and upgrading legacy systems and software. The EISR project has developed an automation-assisted JOVIAL-to-C reengineering capability that permits simultaneous modernization of both the structure and source language of legacy embedded applications.

The EISR environment has several key features: support for detailed analysis of legacy software, visualization of critical execution sequences and complex data dependencies, rapid source conversion, and a high-percentage source construct conversion rate. Using this capability, developers can rapidly characterize the overall legacy software architecture, perform incremental or wholesale source language conversion, and upgrade selected components and structures. Engineers can apply the proven labor-saving visualization and analysis features provided in modern commercial Computer Aided Software Engineering (CASE) tools to legacy JOVIAL applications. Following conversion, legacy applications can then be imported into other mainstream commercial graphical CASE environments that allow visual reconstruction and automatic source code generation.

To summarize, today's system developers face many general and high-level obstacles impeding evolution and modernization of these systems:

- Greatly extended platform service life spans.
- Rapidly changing mission scenarios, system roles, and threats.
- Increased information-processing requirements.
- Desire for cross-platform commonality of capability and architecture.
- Shrinking budgets.
- DMS affecting both application and software engineering environment hardware and software elements.

In addition to these obstacles and challenges, there are additional detailed design-level issues that must be dealt with in order to derive maximum benefit from large-scale reuse of legacy software.

In this paper, we assume upgrade scenarios where developers will migrate from military specific programming languages and development environments toward mainstream commercial replacements. Successful migration requires dealing with specific aspects of legacy applications and their development, which are outlined below.

Outdated Methods

Legacy systems commonly contain hierarchical, functionally decomposed, time-slice scheduled software architectures targeted to uniprocessor platforms. These designs are often highly coupled through global data pools as opposed to modern data-encapsulated object-oriented analysis/object-oriented design (OOA/OOD) forms. Such coupling hinders selective isolation and capture of proven functionality.

Lack of Modern Integrated Analysis Capability

Legacy development environments often consist of a patchwork of standalone textual/command line-based tools. Although there is support for symbol and dependency tracing, this capability is not comprehensive or integrated and is generally cumbersome, involving the use of several separate tools. As a result, it is difficult to determine the scope of a design change. This can result in increased risk and reduced confidence, thereby eroding the motivation for refreshing the structure of an existing embedded application.

Platform Coupling

Frequently there is no clear separation of operating system and pure application functionality. Legacy code often contains interspersed code sequences that perform input/output (I/O) device control, data formatting, and interrupt handling.

Hardware performance (temporal knowledge) is often encoded into legacy algorithms in the form of time constants. This was done to achieve and maintain performance and computational accuracy as the system matured. Thus changes or updates to the processing hardware have unintended detrimental effects on *unchanged* software. Successful migration and reuse of applications containing such characteristics depends on up-front investigation, identification, and understanding of low-level platform coupling to fully understand top-level design constraints and performance impact aspects.

Structural Degradation

Many currently fielded embedded systems are extensions of custom designs that have evolved during an extended upgrade and maintenance lifetime. Engineering decisions made in the development of these early systems were often specific to the *task at hand*, resulting in system architectures that were not designed for direct reuse. As the product is maintained and upgraded over time, the original architectural design often degrades due to multiple sets of small modifications. Local optimizations are made without adherence to the overall architectural policies that drove the initial design.

As a result, many legacy designs are costly to update because of non-uniformity and brittleness. The structural degradation combined with hidden platform coupling often creates a significant perceived risk in reusing an application, eliminating the consideration of this as an option. The application is perceived to be *spaghetti code* that is overwhelmingly complex and unsuitable for reuse.

Resource Constraints

The limited memory, I/O, and processing capacities of legacy militarized electronics units often drove developers to make design decisions favoring efficiency over quality of design. Often mechanisms peculiar to the inherent programming language were used to provide more efficient but not necessarily extensible designs. As a result of design tradeoffs due to the resource constraints and the extended maintenance/upgrade cycle, the applications often evolve characteristics such as a large number of complex threads of control that cross processing segment boundaries and result in complex segment coupling.

Implicit data dependencies shared across processing segments result in data-driven segment coupling and degradation of individual segment cohesion. This combination of tightly coupled segments of code designed using specific programming language constructs can result in a complex system that is difficult to dissect into reusable and migrational segments. Often such an undertaking requires extensive manual analysis and redesign, thereby increasing update costs.

COTS Exploitation

In contrast to past decades, the defense industry is no longer the driving force behind the development and production of computing electronics and software engineering environments. Although the selection of purely commercial off-the-shelf (COTS)-based architectural approaches has yet to overcome many of the inherent problems facing airborne embedded systems, the industry must now rely on the commercial marketplace for large-scale procurement of select processing architecture elements.

The incorporation of commercially available processing elements promises to provide increased throughput, memory reserves, and I/O bandwidth. However, this advance in technology brings with it potentially greater DMS concerns, as commercial components typically have a two-year refresh cycle, forcing application developers to plan for much shorter hardware lifetimes. In addition, legacy design and development tools are often not available for commercial systems, further stressing the development organization.

Available Knowledge and Experience Base

The knowledge and experience level of a development team is critical to the ability to maintain and upgrade existing embedded systems. The original design criteria for an older system may be missing or inadequately documented. Taking into account that the current development team may have no contact with the original designers, capturing the in-depth knowledge encapsulated in the design becomes critical.

Without this information, developers are often unable to overcome the impacts of structural degradation and evaluate the impact of resource constraints and hardware/software coupling. Migrating entire or selected portions of legacy applications to new platforms or architectures may require so much time to understand the impacts that this approach becomes no longer cost-effective.

Despite these problems, the reuse of legacy software functionality in both fielded and future systems is programmatically attractive. Therefore, we must develop a strategy to overcome the barriers of limited budget, increased integration and production costs, shortened cycle time, COTS insertion, and DMS. The failure to do so may adversely affect the DoD's ability to provide and sustain quality products within available funding profiles and scheduled need dates.

Following is a brief description of the EISR project, its accomplishments, and results from the F-117 flight demonstration.

Technical Approach

The EISR program focused on maturation and integration of two capabilities: structural visualization and construct transformation. Combined, these capabilities provide maximum utility for users interested in wholesale or incremental upgrade. The visualization system provides graphic depictions of data dependency, control flow, and program component interaction. Transformation capability focuses on complete and accurate construct coverage, with certain caveats. For example, it was known in advance that certain JOVIAL constructs had no equivalents in C, and that 100 percent construct coverage was not achievable. However, conversion of even 95 percent of JOVIAL constructs was deemed highly successful as this minimizes the amount of manual reengineering work required to obtain an operational converted application.

The EISR program was aware of several past reengineering environment development efforts that attempted to provide automated restructuring and design aids. One of the problems encountered with earlier efforts was accommodating a variety of target design styles. Different embedded information systems often employ different architectural styles as maintainability requirements vary from application to application. Programming a design environment with expert knowledge of each

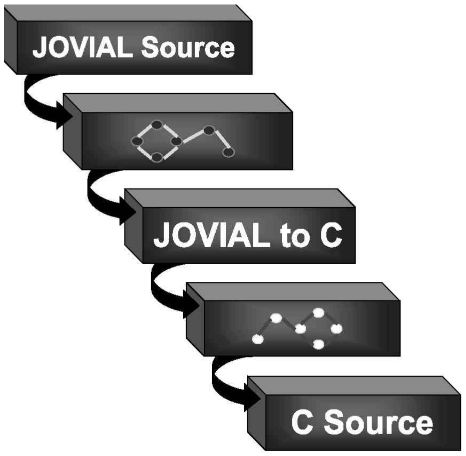


Figure 1: *Transformation Process*

desired target design style is a highly challenging problem. After careful consideration, it was decided to defer investment in automated restructuring and design aids as the combination of visualization and transformation promised the greatest initial return on investment.

Project Results

The EISR technical product is a desktop software-reengineering environment. This system operates on the JOVIAL source files making up an application, parsing and converting them into language-neutral graph-based representations of their operation (see Figure 1). The use of this internal representation yields several benefits. First, since it is language-neutral, this form provides common basic semantics to facilitate integration of *back-end* code generators for a variety of specific target source languages. Second, the form abstracts away language syntax specifics and other peculiarities. Procedural and data elements can then be represented in a common *fundamental* graph-based form that captures inherent interdependencies (see Figure 2).

Figure 2 provides a realistic example of complex data and component dependencies found in typical legacy software artifacts. High legacy software maintenance costs are due in part to the inability of developers to rapidly trace the effects of desired software changes. The EISR graphical analysis suite remedies this by replacing past manual and text-based dependency tracing methods with modern visual tools and search engines. The developers can thus manipulate this graphical representation of the program structure directly, and apply filters and navigational tools to assist in rapid interpretation and restructuring.

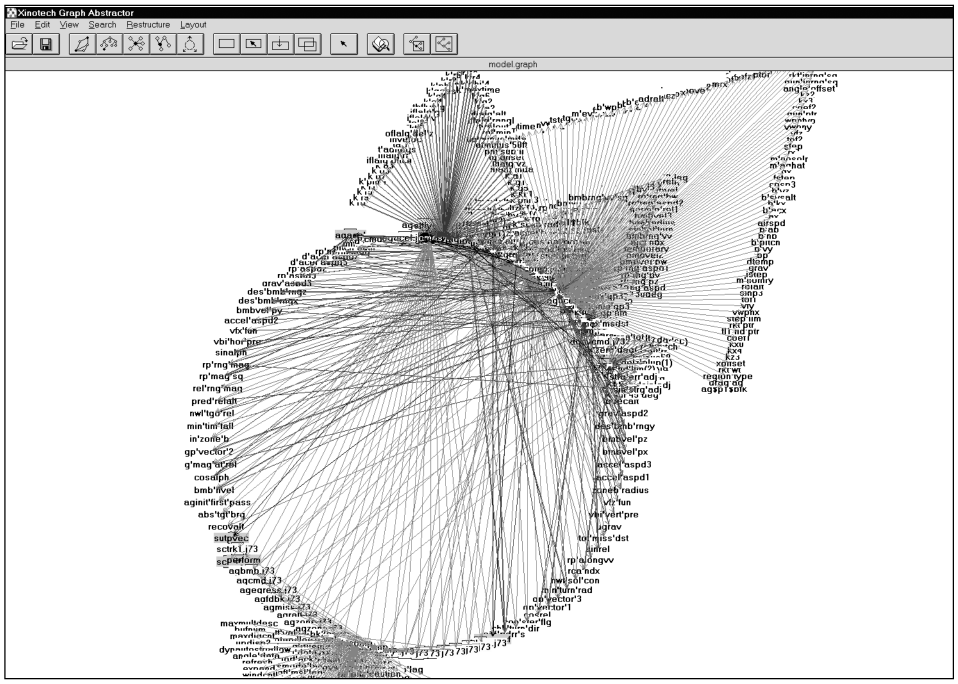
This combination of EISR features gives developers the ability to visualize and trace couplings and structural features of the legacy code. As a result, software engineers now have a capability for understanding and visualizing legacy software artifacts that is on a par with modern graphical CASE tools. A feature of the EISR tool-set is extensibility. The environment is designed around an intermediate representation form that provides a common framework for integrating new *front end* parsers and *back end* target source code generators.

Base Experiments Results

A set of base experiments was conducted under EISR to evaluate the semantic performance and conversion speed of the EISR tool-set using actual DoD application software. The following list summarizes the base experiment results:

- EISR matured capability results in a source code conversion rate of 10K source line of code (SLOC)/minute (PC/NT based).
- Comparative manual conversion rates varied from 20 SLOC/hour to 67 SLOC/hour depending upon experience level of developers and legacy application complexity.
- An initial experiment involving a 4,000 SLOC application required less than 24 hours of *clean-up* touch labor.
- Within the EISR tool-set on initially selected test programs, 100 percent JOVIAL construct coverage was achieved. The following caveats applied:
  - Constant tables became full-fledged structure variables in C.
  - Highly convoluted variable overlays were flagged for manual reengineering since this would be the best overall solution in light of improvements in computing resource availability.
  - Compiler directives were excluded from this statistic since they are not part of the military standard.
- Initial JOVIAL functional structure was retained in the resulting C code with no loss of design partitions or structural understanding.
- Detailed *before and after* code inspections validated that the process of statement-to-statement transformation was l*ossless*.

Figure 2: *EISR Graphical Analysis View (model illustration only)*


Xinotech Graph Abstractor
File Edit View Search Restructure Layout
model.graph

Extended Experimentation

The results of the EISR project provided the opportunity for further experimentation and evaluation using the converted artifacts from the original experiments. Engineers were curious to study the ease with which converted JOVIAL artifacts could be migrated to modern commercial object-oriented CASE environments.

In these experiments, a legacy application with a functional structure was converted to C using both manual and automated processes. The resulting code was imported into an object-oriented, Unified Modeling Language (UML)-based commercial CASE tool. The code then underwent a mild restructuring to migrate the original functionally decomposed design to a medium-grained object-oriented structure. The CASE tool was then used to auto-generate C++ source code from the reengineered representation. The results are summarized below:

- Test Case 1: Transforming Functional Decomposition to object-oriented

design (OOD). Consisted of manual JOVIAL-to-C code conversion and manual transformation from functional decomposition into C++ OOD.
- Test Case 2: Transforming from Functional Decomposition to OOD. Consisted of automated JOVIAL to C code conversion using the EISR technology and manual transformation into C++ OOD.

The result of each test case was a legacy JOVIAL-based application transformed into an object-oriented C++ base application (defined using UML notation) targeted for a PC-based processing platform. In comparing Test Case 1 and Test Case 2, we found that the use of the EISR tool-set to capture and transform the JOVIAL to C reduced the level of effort for the overall process (JOVIAL to UML/C++) by approximately 75 percent.

This experimental set was of particular significance because it examined likely and desired future migration goals. We believe that developers will want to move legacy artifacts into modern graphical CASE environments in order to take advantage of automation-assisted testing features, improved software understanding, commercial standard notation benefits, and cost savings due to economies of scale. EISR technology is thus a *key enabler* for full COTS exploitation, as it forms a bridge to modern commercial practices and tool-sets.

Flight Demonstrations in Summer 2001

A *first ever* flight demonstration of reengineered avionics application code took place on July 12, 2001 over Edwards AFB aboard a USAF F-117 Nighthawk stealth fighter. This significant flight demonstration successfully verified correct performance of reengineered components generated by using high degrees of automation assistance. In this demonstration, a small component of the aircraft navigation application was converted from JOVIAL to C++ using the EISR suite and installed in a Power PC-based mission computer prototype.

This application component ran accurately and continuously through a 1.5-hour flight, providing critical data processing in support of the aircraft system navigation solution. This functionality is part of the F-117 precision navigation suite that the pilot relies on extensively throughout the entire mission. Subsequent tests involving delivery of practice and precision munitions were accomplished during the latter part of July 2001.

This demonstration provided a key confidence point, proving operational viability of reengineered application code. This work illustrated an affordable upgrade strategy for the F-117 mission computer using reengineering and computer emulation technology developed at AFRL/IF. EISR thus allows the DoD to affordably recapture previous investments in proven legacy software artifacts and create a migration pathway for exploitation of COTS economies of scale.

EISR Significance

- EISR technology is estimated to significantly reduce design time-span. The technology is mature. Resulting applications have been operationally proven in several flight demonstrations performing realistic missions.
- EISR capability is currently available *off the shelf*.
- The completeness and robustness of the EISR tool-set resulted in a low risk of losing design content (key algorithms, behavior) during the reengineering process.
- EISR technology reduced the effort associated with the coding phase by an order of magnitude when compared to manual transformation. This, in turn, resulted in a 20 percent cost reduction when considering the *overall* software development process (i.e., design, code, test, etc.).
- EISR technology has enabled a paradigm shift and new programmatic process for legacy information system upgrade (see Figure 3).
- Robust, automation-assisted reengineering capabilities enable affordable structural refresh.
- EISR technologies provide a low-risk bridge for migration to modern commercial CASE tools – which in turn enables even greater cost savings potentials.

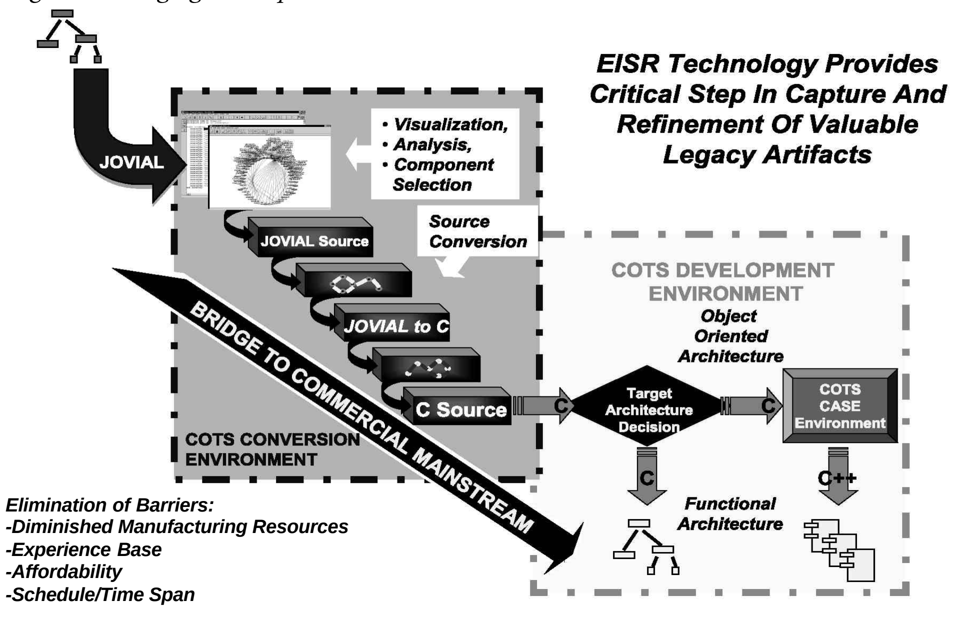
Summary

The EISR program has been successful in putting legacy DoD application software on a convergent path with mainstream software engineering tools and practices. The resulting EISR tool-set has been proven to reliably and wholly capture and transform legacy application content. Legacy software transformation is performed quickly and affordably, drastically improving design cycle times when compared to manual methods. In addition, the resulting artifact forms allow application of a variety of COTS tool-sets for continued development and maintenance. The following list summarizes EISR benefits:

- Modern CASE visualization and analysis capability for legacy designs.
- Minimal introduction of human errors during the process of manual transformation.
- Order of magnitude improvement in SLOC conversion per labor hour expended rate.
- Bridge to mainstream commercial products and practices.
- Maintainability and supportability improvements.
- Affordability increases by leveraging economies of scale.
- Higher availability of skilled developers.

The EISR technology has successfully completed engineering proof-of-concept evaluations. AFRL/IFTA, on behalf of the Computer Resources Support Improvement Program Office (CRSIP), has acquired

Figure 3: *Bridging the Gap*


EISR Technology Provides Critical Step In Capture And Refinement Of Valuable Legacy Artifacts
JOVIAL
• Visualization,
• Analysis,
• Component Selection
Source Conversion
JOVIAL Source
JOVIAL to C
C Source
BRIDGE TO COMMERCIAL MAINSTREAM
COTS CONVERSION ENVIRONMENT
COTS DEVELOPMENT ENVIRONMENT
Object Oriented Architecture
C
Target Architecture Decision
C
COTS CASE Environment
C
C++
Functional Architecture
Elimination of Barriers:
-Diminished Manufacturing Resources
-Experience Base
-Affordability
-Schedule/Time Span

ownership of the EISR technology, including the source code. In October 2001, the EISR technology was transitioned to the CRSIP Program Office that will provide long-term support and maintenance and will facilitate distribution of the technology. Source code for the EISR tool-set is currently owned by the Air Force. The CRSIP Program Office is located at Ogden Air Logistics Center, Hill Air Force Base, UT. Parties interested in licensing use of the tool can contact Gerald L. White at (801) 775-6713, or via e-mail at <gerald.white@hill.af.mil>.◆

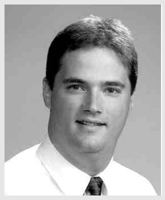
About the Authors

Kenneth Littlejohn is a project engineer in the Embedded Information Systems Engineering Branch, Information Directorate, Air Force Research Laboratory. He has more than 19 years research experience related to affordable design, development, and support of real-time embedded software for Air Force weapon systems. Littlejohn currently serves as the project engineer for the Embedded Information System Reengineering project. He earned a bachelor's degree in electrical engineering in 1987, and a master's degree in computer science in 1994, both from the University of Dayton.

AFRL/IFTA
2241 Avionics Circle
WPAFB, OH 45433-7334
Phone: (937) 255-6548 ext. 3587
Fax: (937) 656-4277
E-mail: kenneth.littlejohn@wpafb.af.mil

Michael V. DelPrincipe has more than 15 years experience in the design, development, and test of embedded real-time avionics fire control, weapons management, and display software applications for airborne weapon systems. He is currently responsible for the technical and programmatic management of multiple Air Force Research Laboratory-sponsored research projects related to legacy embedded information system modernization. DelPrincipe earned a bachelor's of science degree in computer science with honors from the State University of New York, Brockport campus.

Lockheed Martin Aeronautics Company
P.O. Box 746
Fort Worth, TX 76101 MZ 6295
Phone: (817) 777-3667
Fax: (817) 777-3121
E-mail: michael.v.delprincipe@lmco.com



Jonathan D. Preston is a technology program manager within the Advanced Development Programs branch of Lockheed Martin Aeronautics Company. Throughout his career, he has managed several government-funded technology programs that have transferred technologies to major weapon system programs. Preston earned a bachelor's degree in electrical engineering from the Pennsylvania State University and an master's degree in computer science engineering from the University of Texas at Arlington.

Lockheed Martin Aeronautics Company
P.O. Box 746
Fort Worth, TX 76101 MZ 2411
Phone: (817) 763-2740
Fax: (817) 763-2967
E-mail: jonathan.d.preston@lmco.com

Ben A. Calloni, Ph.D., is a research program manager for multiple software research and development efforts at Lockheed Martin Aeronautics Company, Fort Worth, Texas. He is leading the investigation into commercial off-the-shelf solutions for legacy avionics software. Dr. Calloni earned a bachelor's of science degree in industrial engineering from Purdue University, and master's and doctorate degrees in computer science from Texas Tech University. He is a licensed professional software engineer in Texas.

Lockheed Martin Aeronautics Company
P.O. Box 746
Fort Worth, TX 76101 MZ 2859
Phone: (817) 777-4345
Fax: (817) 763-2967
E-mail: ben.a.calloni@lmco.com

COMING EVENTS

January 27-31, 2002
2002 Western MultiConference
San Antonio, TX
www.scs.org

February 4-6, 2002
International Conference on COTS-Based Software Systems (ICCBSS)
Orlando, FL
www.iccbss.org

February 11-15, 2002
Application of Software Measurement (ASM 2002)
Anaheim, CA
www.sqe.com/asm

February 25-27, 2002
15th Conference on Software Engineering Education and Training (CSEE & T)
Covington, KY
www.site.uottawa.ca/cseet2002

April 28–May 2, 2002
Software Technology Conference 2002 "Forging the Future of Defense Through Technology"
Salt Lake City, UT
www.stc-online.org

May 13-17, 2002
Software Testing Analysis and Review (STAREAST 2002)
Orlando, FL
www.sqe.com/stareast

June 4-7, 2002
8th IEEE International Symposium Software Metrics (Metrics 2002)
Ottawa, Ontario, Canada
www.software-metrics.org

The IULS Approach to Software Wrapper Technology for Upgrading Legacy Systems

Dr. David Corman
The Boeing Company

This article describes using software wrappers in Incremental Upgrade of Legacy Systems as a key technology for modernizing legacy systems. It introduces three types of wrappers, describes a process for selecting which upgrade path to utilize, and discusses a tool-set developed by The Boeing Company for the Air Force that automatically generates wrappers. Lastly, it discusses real-world avionics upgrade examples where the tool-set has been applied and its effectiveness.

Avionics upgrades are frequent and occur for many reasons, including warfighting enhancements, countering changing threats, hardware obsolescence, and computer resource under-capacity. A typical production avionics upgrade cycle for military aircraft frequently involves embedded software changes. New versions of mission processor software, the most volatile class of avionics software, are typically released annually and take two years to field from initial definition.

Hardware obsolescence occurs collectively over a longer term as vendors change their business (military/commercial mix) and technology. Software tools and technology also evolve over a longer period but may be driven by short-term events such as the introduction and imposition of Ada. The change cycles are not synchronized so the optimal hardware, software and tool technology, and respective program funding to support an avionics upgrade at a given point in time are often not available.

The problem of cost-effectively upgrading legacy systems can be mitigated through reengineering with the latest generation hardware and architectural concepts, including object-oriented (OO) software design, which inherently contains and isolates change. However, legacy avionics software represents a large investment in development tools, executable code, and ground and flight qualification. Should the upgrade require complete reengineering of this legacy software, much of this investment is lost; many aircraft programs simply cannot afford the up-front costs associated with reengineering and complete re-qualification.

One solution to this dilemma is implementing reengineering incrementally by inserting the latest technology in smaller, affordable steps, thereby reducing risk and deferring or reducing cost. Software wrapper technologies hold particular promise in meeting this challenge.

A wrapper is a software adapter or shell that isolates a software component from other components and its processing environment (its context). The wrapped component becomes a software object. Its operational capability (functions and data) is encapsulated, and it can be integrated through its standard interface with other software objects to form an operational flight program (OFP) on a single or distributed processor host. The wrapper manages the timeliness of all shared and external data, and provides any necessary transformations.

For upgrades, the goal is to develop the new or reengineered applications using the latest software engineering techniques such as OO design and languages (Ada and C++) with minimal concessions to the internal structure of the legacy system. It is developed as if all other applications were resident in the new environment. Because the new software is written within the paradigms of OO design and languages, the wrapper can eventually be removed once all of the application functions have migrated to the new system. At this time, the legacy system can also be removed.

The Incremental Upgrade of Legacy Systems (IULS) program is a research and development effort, whose main objective is to develop, demonstrate, and transition software wrapper technology that will enable cost effective, incremental improvements to fielded weapon system avionics. The products of IULS are: 1) methodology for analyzing software upgrade approach, 2) wrapper technology, 3) tool-set for constructing wrappers for software upgrades, and 4) demonstrations of IULS wrapper technology applied to three significantly challenging problems: F-15E, C-17, and CV-22 avionics.

IULS is funded by the Air Force Research Laboratory, Embedded Information Systems Engineering Branch (AFRL)/(IFTA). Participants in the project include The Boeing Company, Honeywell Technology Center, General Dynamics Information Systems, and TRW-Dayton.

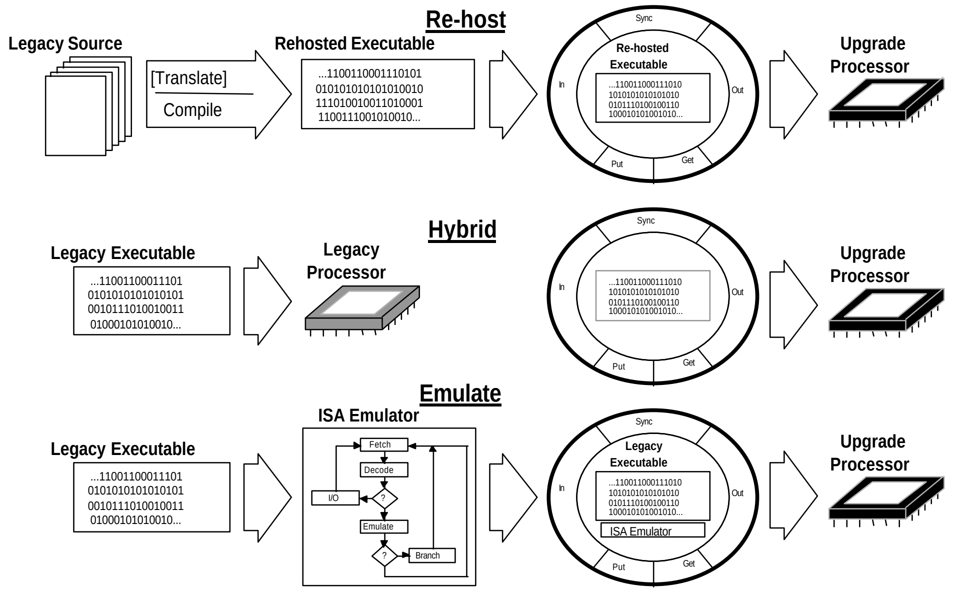
Software Wrapper Technologies

Figure 1 illustrates three hypothetical cases of implementing software changes using wrappers.

Re-Host

In the *re-host* case, the legacy processor is obsolete and/or its resources are insuffi-

Figure 1: *Wrapper Cases*



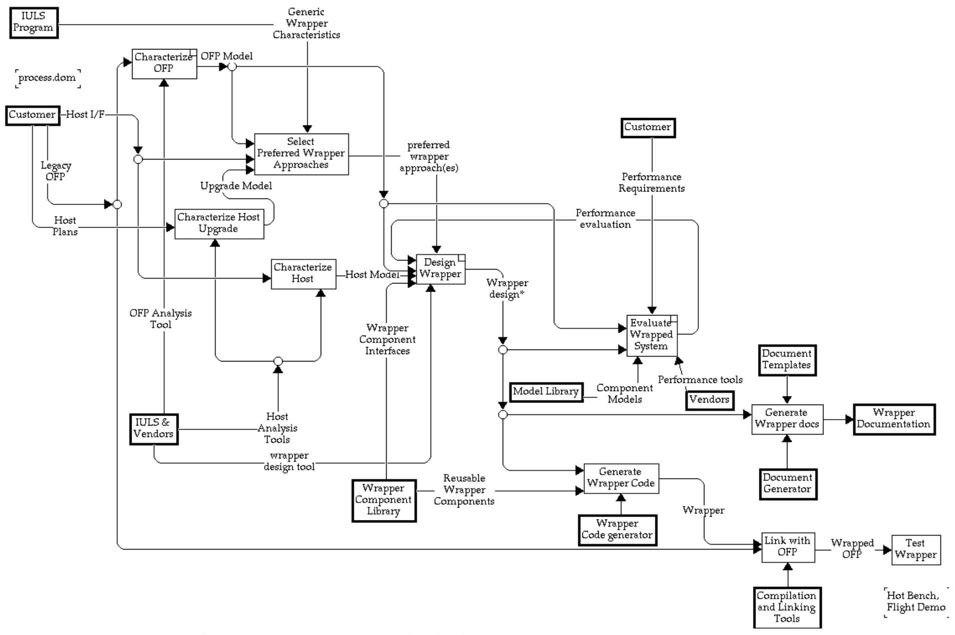


Figure 2: *Nominal Legacy Operational Flight Program Wrapper Process*

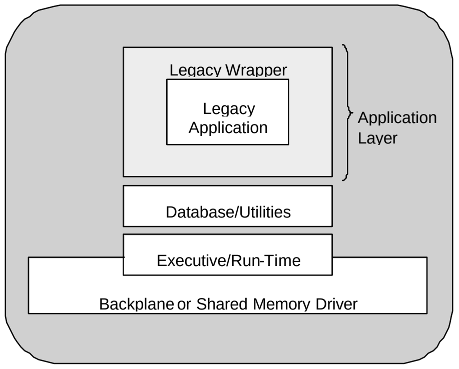
cient to support additional upgrades. The legacy software is re-hosted to a new processor by translating its source code (e.g., Ada 83 to C++) and/or recompiling it for the new target (e.g., Ada 83 to Ada 95). Reengineering the OFP on the new processor could not be justified so wrapper components are added to make it *look like* an object in the OFP. New software features can be added incrementally to the wrapped component, or preferably, designed as new objects in the OFP.

Hybrid

In the *hybrid* case, the legacy processor and its OFP are retained for various reasons (high reengineering or logistics costs, etc.), but its resources are insufficient to support additional upgrades. Also, there is an opportunity to satisfy upgrade requirements with reuse library components that are developed with more modern languages (such as Ada 95 or C++) and tools.

New features can be added incrementally to the upgrade OFP as objects on a new processor. The objects will be bridged to the legacy OFP and processor with wrapper components. As components in the legacy OFP require changes, they can be reengineered and moved to the new processor. At some point in the migration, the remaining legacy components are re-hosted, the legacy processor is upgraded or discarded, and the wrapper components in the new OFP, associated with the legacy OFP interfaces, can be removed.

Figure 3: *Wrapper Software Structure*



Emulate

Obsolete or underpowered hardware is also addressed in the *emulate* case. The legacy software is judged to be very costly to reengineer and/or re-qualify. The object code is executed on the new processor by an emulation of the legacy processor's instruction set architecture.

Changes can still be made to the legacy executable using the legacy compiler and software engineering environment. The emulator and other wrapper components make the legacy executable component (binary) *look like* an object. Other feature upgrades could be added as new objects on the new processor.

Wrapping Process

As with any other software development activity, wrapper creation follows a process, shown in the IDEF0 diagram in Figure 2, and is automated with tools. In an IDEF0 diagram, consumed inputs (e.g., data files) go in the left side of an activity box; generated outputs (e.g., completed design objects) emerge from the right side; constraints (e.g., requirements, schedules) go in the top; and mechanisms (e.g., tool support) go in the bottom. The following subsections describe tool mechanisms that support the wrapper design process and the data that flows between them.

Wrapper Implementation Considerations

Wrappers are generally applied at the application domain level. They act as clients and servers to the encapsulated component. Figure 3 illustrates a general wrapper structure for an OFP on a single processor. The legacy application interfaces with other applications and other layers only through the wrapper. The wrapper architecture is tailored to the specific legacy OFP environment.

The method selected to implement the upgrade of a legacy system is to an extent an economic decision. The emulation option will tend to favor a context of a stable application, infrequent OFP modification, and obsolete hardware. These cases will generally be lower in performance, being older systems. Therefore, the context analysis would focus on issues of throughput, OFP stability, parts obsolescence, OFP utility, etc.

Selecting a hybrid or re-host approach would generally be appropriate for a more dynamic and/or higher performance system. Typical context would be OFPs that are subject to periodic update and version release. In addition, the higher the throughput needed, the more likely the upgrade path will not include emulation. The reason for this analysis factor is that while *software emulation* provides a growth path for the legacy upgrade, there is a penalty paid for using processing resource overhead. A hardware emulator approach will, with time, become a technology dead-end (as would be the case for the legacy system) and require more near term upgrade effort.

Any selected legacy upgrade technique will, of course, be subject to obsolescence and eventual upgrading. Therefore, the methods described for the upgrade process and wrapper generation emphasize as flexible an approach as possible. Also, the methods need to be portable between hardware platforms since these components will change rapidly (on the order of months vs. years). Therefore, the initial system context analysis would focus on OFP issues such as throughput, stability, etc.

Using the IULS Tool-Set in Upgrading

As part of the IULS program, Boeing and its IULS teammates developed a set of guidelines, processes, and tools to help the avionics engineer determine and implement a *best* wrapper upgrade strategy. The

IULS tool-set that was developed in this effort has played a major role in automating the upgrade approach in the rehost/hybrid domains. The tool-set is used to iteratively develop high-level and detailed models of the OFP model, the host model, and the upgrade model. Figure 4 displays some of the basic elements of the IULS graphical tool-set.

Briefly, the tool provides a graphical capability to model the legacy and upgraded system, including data interfaces and constraints. It includes a re-use component library that can be used to meet requirements common to avionics applications. It provides code and documentation generation capability to construct and document the software wrapper used to bridge the legacy and upgraded system. Also, the tool provides system-modeling capabilities that can be applied to validate system performance against scheduling constraints common to hard real-time avionics systems.

Customer Inputs to the Process

The *select preferred wrapper approaches activity* consists of selecting from among the three basic wrapper approaches. The host plans, legacy OFP, and host interface (I/F) are information supplied by the customer. The information specifies the OFP re-hosting problem in a manner that is sufficient to trigger the next activities. The host plans identify the need to re-host the OFP onto a new target platform and are used to select one of the three wrapper approaches, or possibly narrow the selection down to two of the three approaches that will be considered during the wrapper design process.

Characterizing the Problem

The IULS tool-set is used to characterize and model the legacy system OFP. The resulting *OFP model* is a description of the legacy code, it's source language, the available documentation and compilers, and the description of the I/O required by the OFP, including timing. The model describes each interface modality that the OFP has with other OFPs and with the legacy host hardware. The tool-set is also used to develop the *host model*, which provides a description of the target host computer environment (not the legacy host). It includes a description of the machine code, compilers available, event capabilities, I/O capabilities, and kernel operating system (OS) interfaces.

The *upgrade model* is a description of the plans to upgrade the target host computer in the future. The plans for future upgrades influence the choice of wrapper approaches.

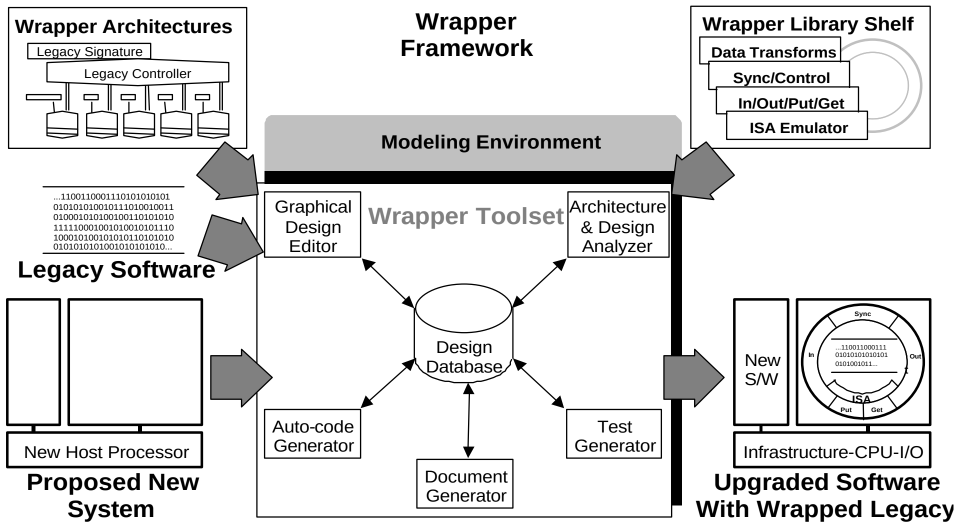


Figure 4: *Graphical Tool-Set*

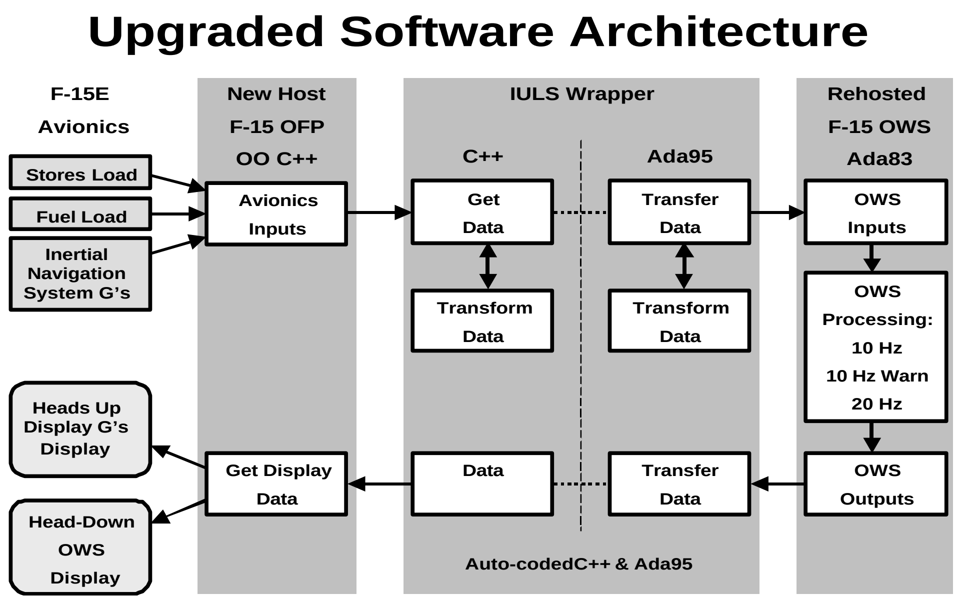
The characterization step provides high-level information that can be used to perform coarse trade-off analyses that contribute toward a final selection of wrapper approach as well as some of the basic decisions about the wrapper design. Some of this high-level model will identify wrapper components able to be tailored from a reuse library.

Wrapper Design

As shown in Figure 4, the wrapper is designed using the OFP model, host model, and upgrade model, as well as the *reusable wrapper components*. The IULS tool-set includes a set of reusable wrapper components that are placed *on the shelf* and can be either quite general in nature (e.g., format converters from one process to another) or domain specific (data access methods for an aircraft). The reusable wrapper components are linked to requirements and test cases that are also reused. The process of wrapper design includes specifying the following within the tool-set:

- Invocation interface: This is the interface for entering into and returning from the OFP code. It includes the description of error handling interfaces.
- Semaphores and interrupt handlers: This is the interface that defines synchronous process behavior, both hardware supported and software only.
- Data accessors: This interface defines accessors for data objects that the OFP shares with other software modules.
- Emulator configuration: This describes the emulator that may be interfaced to the wrapper, depending on the wrapper approach used.
- Object adapter: This is software that makes the wrapped OFP look like one

Figure 5: *OFP Wrapper for F-15 Demonstration*



Component	Software Lines of Codes (Not Comment/Blank)	Total Source Lines
Total OFP (C++ and Ada)	119,363	534,054
OWS Application (Ada Including PIMs)	7,195	23,738
Ada Wrapper	482	880
C++ Wrapper	408	811

Table 1: *Wrapper Component Sizes*

or more objects to an underlying object request broker (ORB). It defines the OO interface classes and methods comprised in the wrapper.

- Legacy port design: If the wrapper approach calls for re-hosting the legacy OFP, then a description of the tools, process, and wrapper interface objects needed to support that approach are specified.

Evaluation of the Wrapped OFP

After a candidate wrapper is designed and a baseline determined, it is evaluated. The evaluation is performed using *component models* selected from a library. The component models are the representations of the target hardware system. These are used in running simulations of the target system to determine performability. System modeling and evaluation tools such as Cosmos and Foresight can be used in concert with the IULS tool-set to simulate and evaluate the wrapper design.

Wrapper Code Generation

Once the wrapper design has successfully passed evaluation tests, the IULS tool-set provides the capability to automatically generate the *wrapper* using a code generator. It identifies the versions of the components, how those components were tailored, and how those components are to link together with the OFP. The wrapper is then linked with the OFP to create the *wrapped OFP*.

OFP Integration, Test, and Documentation

The wrapped components and host components are compiled, linked, and integrated into the target processor system. System and software testing is performed to a level appropriate to the avionics application. By using the same specification for wrapper design, evaluation, and implementation, traceability is greatly simplified and facilitated. Model objects can be cross-referenced to requirements, implementation components, evaluation models, tests, and test results. The IULS tool-set includes a document generator that compiles software documentation from the design database using customizable templates.

Demonstrations

The IULS program included major demonstrations of two of the wrapper approaches: modified re-host and emulation. The demonstrations provided an opportunity to *test and tune* the tool-set in a real-world avionics upgrade environment. The following subsections describe the results of the application of the IULS tool-set.

F-15 Demonstration

For the F-15 demonstration, the IULS problem was to port a legacy F-15 Ada 83 OFP component, the overload warning system (OWS), into a F-15 OFP written in C++ running on a new commercial off-the-shelf (COTS) PowerPC processor. A wrapper was designed and auto-generated using the IULS tool-set. This was the first application of the IULS tool-set and provided us a framework for testing and tuning of the tool.

Figure 5 (see page 11) shows elements of the wrapper design. In particular it shows how the wrapper supports transfer of data from the OO C++ domain into process interface messages understood by the legacy Ada software. The wrapper also performs necessary data transforms and includes a bridge from the Ada 83 software to the C++ and display processes.

The F-15 IULS activity culminated in a successful live flight demonstration conducted on Dec. 1, 1999. The demonstration flight plan called for execution of six test points. These corresponded to combinations of three different weapon loads with two different fuel configurations. The pilot, weapon systems officer, and flight test engineer reported successful test results.

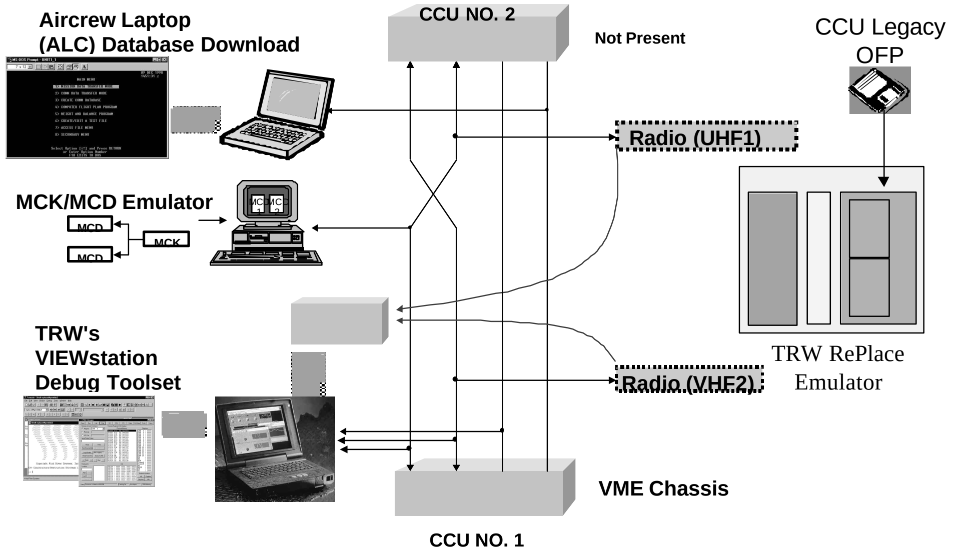
The relative sizes of the components (in source lines of code) for the final demonstration and flight test OFP are shown in Table 1.

We collected metrics on the wrapped system to measure wrapper overhead. It was indicated that wrapped system added an average of 0.36 msec and 0.16 msec to the OWS application execution timelines for the 20 Hz and 10 Hz tasks, respectively, for operation on a PowerPC 603E. The computations were all performed within the time-frame requirements for the OFP.

The F-15 demonstration thoroughly validated the IULS re-host process and tool-set. Operationally, the demonstration received enthusiastic endorsement from the flight crew who referred to it as a *home run* in the post flight debrief. The in-flight performance was 100 percent in agreement with the a priori estimates matching all six test points, exactly. The WrapidH tool proved to be extremely valuable in developing the wrapper design, and the automated code generator worked as expected in both the Ada and C++ domains.

As predicted, considerable domain expertise was required to develop the wrapper. However, IULS engineers who initially had no familiarity with the heritage code performed the bulk of this work. These engineers were able to readily understand the legacy Ada and Common Operational Flight Program (COFP) C++ to the extent required to support wrapper design and system de-bug. Wrapper testing confirmed the prediction that wrapped code integrity would be intact – no problems were detected in which wrapped code operation was an issue. Wrapper overhead was not sub-

Figure 6: *C-17 Demonstration for Emulation Wrapping*



stantial and confirmed system modeling conducted during phase one of the IULS program, which had predicted system throughput was more than adequate for the demonstration requirements.

C-17 Emulation Demonstration

The IULS emulator approach was demonstrated by *wrapping* a legacy C-17 radio control function (RCF) executable (JOVIAL source, MIL-STD-1750A object) with a 1750A ISA emulator running on a PowerPC processor that represented an upgraded communications control unit. The emulator interface and processor context wrappers were generated with TRW's RePLACE tool-set. Figure 6 shows the demonstration concept.

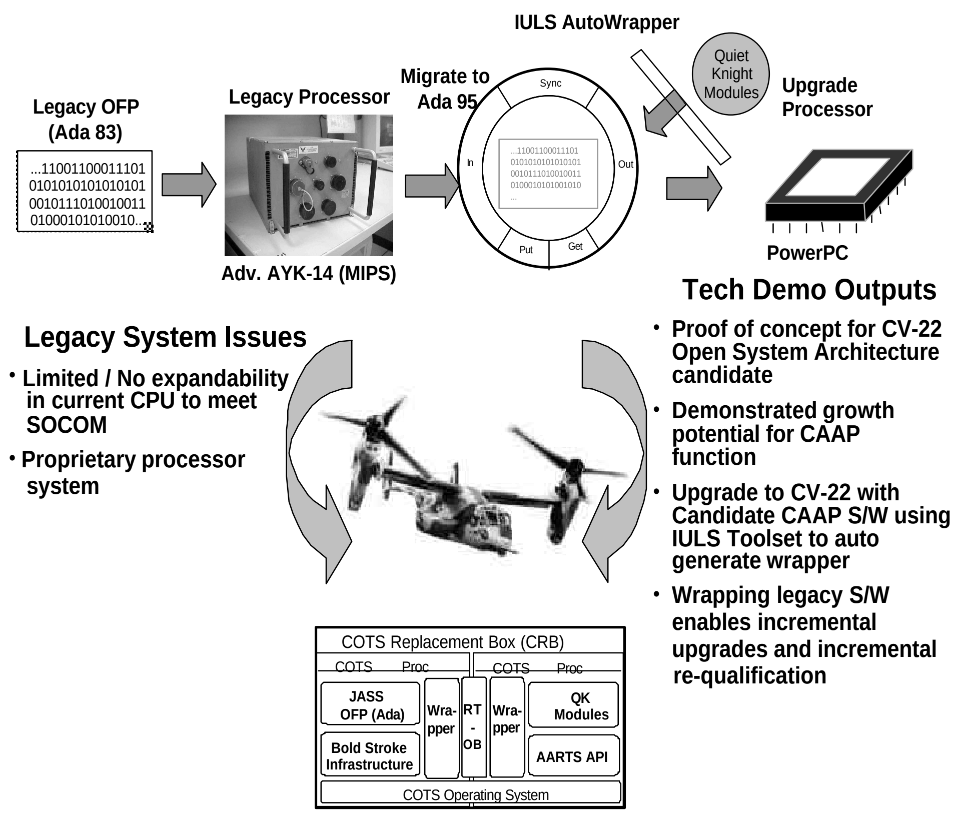
A COTS replacement box (CRB) was constructed, including COTS processor (PowerPC). The emulation engine and RCF OFP were loaded onto the CRB. The CRB operated in concert with the 2nd communications control unit (CCU). This provided a timing challenge since handshaking, normally performed by two 1750A processors across the 1553 interface, was now being performed by the CRB and one CCU. The system was demonstrated in the C-17 avionics laboratory with production test cases and avionics system hardware. The demonstration showed an approximately 90 percent growth capacity for the CRB

The emulation tool-set is being transitioned to the C-17 as part of the on-going communications open systems architecture engineering and manufacturing development program. In this application, emulation is being extended as part of a larger open system upgrade. In particular, a new C++ native language executive is being developed that will then make *calls* to selected (emulated) legacy components. In effect, the emulated legacy software functions as a library of callable functions. This is in contrast to the standard emulation wrapper in which what is simply desired is to execute the legacy software on a more modern processor.

The lesson in transiting the emulation wrapper is that quite frequently programs will apply tools in a different manner than were originally planned, and the tool-set needs to provide inherent flexibility.

CV-22 Demonstration

We are applying the *re-host* wrapper approach with a twist during an on-going demonstration with the CV-22 program. Figure 7 shows the basic elements of the demonstration. The challenges are twofold: First, migrate Ada JASS-JVX Avionics System Software mission software from proprietary processor and system architecture to a COTS PowerPC processor and VME architecture residing under the Boeing Bold Stroke software infrastructure. Second wrap a new application (Quiet Knight – Ada 95) to be executed on a second COTS processor using the tool-set and object request broker (ORB) for distribution.



IULS AutoWrapper
Legacy OFP (Ada 83)
...11001100011101
0101010101010101
0010111010010011
01000101010010...
Legacy Processor
Adv. AYK-14 (MIPS)
Migrate to Ada 95
Sync
In
Out
Put
Get
Quiet Knight Modules
Upgrade Processor
PowerPC
Tech Demo Outputs
Legacy System Issues
• Limited / No expandability in current CPU to meet SOCOM
• Proprietary processor system
• Proof of concept for CV-22 Open System Architecture candidate
• Demonstrated growth potential for CAAP function
• Upgrade to CV-22 with Candidate CAAP S/W using IULS Toolset to auto generate wrapper
• Wrapping legacy S/W enables incremental upgrades and incremental re-qualification
COTS Replacement Box (CRB)
COTS Proc
COTS Proc
JASS OFP (Ada)
Bold Stroke Infrastructure
Wra-pper
RT-OB
Wra-pper
QK Modules
AARTS API
COTS Operating System

Figure 7: *IULS CV-22 Wrapper Demonstration*

The CV-22 demonstration is a work in progress. At the time this article was written, the demonstration is planned for December 2001. The wrapper tool-set is being used to develop the ORB interface definition language to construct the wrapper. Initial results have been very promising. The mission OFP has largely (99 percent) been ported to the COTS processor, and the IULS tool-set has been used to generate the wrapper software for the ORB.

Summary

This article has described several important applications of software wrappers to legacy software modernization. Under the AFRL-sponsored IULS program, Boeing developed a tool-set that can be made available to qualified requestors from AFRL/IFTA to support avionics upgrade opportunities. Wrappers are not a panacea for every modernization need. However, experience to date indicates that they can be an important element within the upgrade process.◆

About the Author

David Corman, Ph.D., is a technical fellow of The Boeing Corporation. Dr. Corman is currently working on a variety of projects that focus on upgrading of legacy systems. He was the lead engineer on the Incremental Upgrade of Legacy Systems (IULS) program that resulted in the development of a wrapper tool-set that has been demonstrated in real-time applications including the F-15, C-17, and CV-22. Dr. Corman has more than 20 years experience in software development, including real time, mission planning, C4I systems, and avionics domains, and holds bachelor's and master's degrees in systems science and mathematics, and applied mathematics from Washington University in St. Louis. He earned a doctorate in electrical engineering from the University of Maryland, College Park.

The Boeing Company
P.O. Box 516
St. Louis, MO 63166
Phone: (314) 234-3725
E-mail: david.e.corman@boeing.com

A COTS-Based Replacement Strategy for Aging Avionics Computers

Jahn A. Luke
Air Force Research Laboratory/Information Directorate

Douglas G. Haldeman and William J. Cannon
TRW Avionics System Division/Avionics Engineering Centers

This article describes a commercial off-the-shelf (COTS)-based form, fit, function, and interface replacement strategy for legacy avionics computers and embedded information systems that can reuse existing software code as is while providing a flexible framework for incremental upgrades and managed change. It is based on a real-time embedded software technology that executes legacy binary code on the latest generation COTS microprocessors. This scaleable technology, developed by TRW and sponsored in part by the Air Force Research Laboratory, demonstrates performance improvements of five to 20 times that of the legacy avionics computer that it replaces. It also promises a four-fold decrease in cost and schedule over rewriting the code, and provides a known, good starting point for incremental upgrades of the embedded flight software. Code revalidation cost and risk are minimized since the structure of the embedded code is not changed, allowing the replacement computer to be retested at the black-box level using existing qualification tests.

Military aircraft are active years longer than originally anticipated. The avionics computers on these aging aircraft are expensive to maintain due to parts obsolescence, and require additional processing and memory to handle changing requirements. As a result, old computer hardware needs to be replaced with newer, more capable microprocessor technology. However, incompatibility issues require that embedded software be rewritten. It is estimated that billions of upgrade-dollars could be saved if new computers could execute the old embedded code along with any new code to be added.

TRW has developed a generic commercial off-the-shelf (COTS)-based software technology called Reconfigurable Processor for Legacy Avionics Code Execution (RePLACE) that capitalizes on technology advances to eliminate costs associated with rewriting legacy software to execute on new hardware. This software allows a user to 1) replace obsolete computers with commercial processor-based hardware, 2) continue to use existing operational flight plan (OFP) without modification, and 3) incrementally upgrade hardware and software to support new and modified capabilities. RePLACE COTS software allows Department of Defense (DoD) dollars to be spent on solving supportability and performance problems and adding new needed capabilities, rather than recapturing current capabilities in new hardware.

Figure 1 depicts using the COTS software to migrate from a current legacy avionics line-replaceable unit (LRU) to a new COTS-based replacement box (CRB). Typically, today's legacy avionics and embedded information systems are based on custom hardware and proprietary back-planes with an obsolete 16-bit instruction set. Little or no "modern" higher order language (HOL) support is available to support software for this equipment. In addition, these legacy systems are often limited in terms of throughput and memory.

A COTS-based, open-systems replacement strategy provides a low cost of entry strategy for COTS microprocessor technology insertion. Both the legacy instruction set architecture (ISA) as well as the new native ISA can be executed. Native code execution is supported by advanced HOLs such as Ada95 and C++. In addition, legacy code can execute much faster in the new hardware and utilize more memory than is available for needed upgrades.

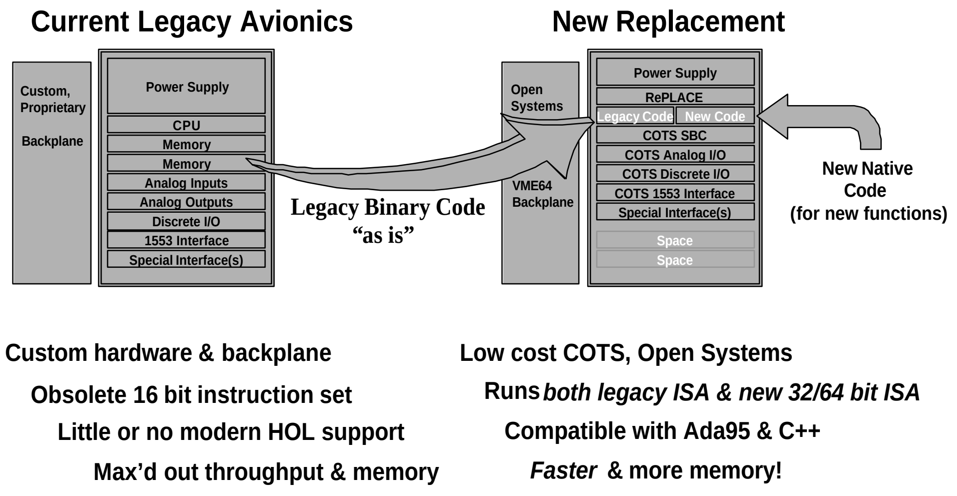
A software perspective of RePLACE is illustrated in Figure 2. Sitting on top of the COTS microprocessor and Portable Operating System Interface (POSIX)-compliant real-time operation system (RTOS) are two virtual machines: the legacy virtual machine and the native virtual machine. Within the legacy virtual machine space are four key software components: the legacy instruction set engine, the legacy OFP code binary image, the input/output (I/O) mapping software, and the virtual component environment (VCE).

The legacy instruction set engine contains unique cache-optimized code developed by TRW to execute the legacy binary OFP code. The I/O mapping software maps the new COTS interface devices to the legacy interface devices. The VCE allows for the efficient transition between the native and legacy virtual environments allowing the transfer of data and concurrent execution of both legacy and new native code.

The key features may be summarized as follows:

- Relies on using state-of-the-art COTS microprocessors and open system standards that improve both the legacy system's producibliity and supportability.
- Runs legacy code from five to 20 times faster than the original legacy system by using a unique cache-optimized approach. This provides extra comput-

Figure 1: *Current Legacy Avionics to a New COTS-Based by Utilizing COTS Software*



ing power for new functions. In addition, the performance of the legacy virtual machine is linearly scaleable with the performance of the COTS microprocessor technology that is being used. That is, as the internal clock speed of the microprocessor chip goes up, the legacy instruction/execution speed increases linearly.

- Makes the instruction set engine adaptable to diverse instruction set architectures, including, for example, MIL-STD-1750A, Z-8002, and AN/AYK-14A. This makes possible the replacement of multiple diverse legacy avionics LRUs with a common set of avionics hardware modules on a given platform or across different platforms.
- Provides I/O mapping software that exactly matches the new I/O COTS devices to the legacy I/O devices, providing a *drop-in* environment for the unmodified legacy OFP with little or no knowledge required of the original code.
- Executes both legacy and new native software concurrently in separate virtual spaces. This promotes incremental addition of new capabilities and gradual transition to new code.
- Provides instruction execution speed matching that can be initiated in critical sections of the OFP to provide legacy OFP timing adjustments for timing sensitive code.
- Allows practical real-time non-intrusive (RTNI) monitoring of legacy software built into the replacement avionics, dramatically enhancing the observability and testability of the embedded legacy software.

In order to illustrate the concurrent execution of legacy and native code in a RePLACE dual instruction set computer (DISC) environment, Figure 3 depicts the legacy code remaining in control while new software enhancements are introduced. On the far left are two time-lines showing the various rate groups processing tasks running in the legacy machine and in the CRB.

In the case of the CRB, the original rate groups are executed in a much shorter time frame within any given minor cycle. This leaves additional processor throughput at the end of each minor cycle to add new software running in the native ISA. Through the VCE context switch mechanism, referred to as a *thunk*, new native code can be introduced to replace existing legacy code or added to the existing legacy code.

Alternately, event flags can be set to augment the legacy code thread as illustrated. Note that legacy instruction execution speed matching can also be introduced for timing-sensitive code. In addition, the technology includes the capability to disable legacy code outputs without legacy OFP cognizance, providing a convenient mechanism to switch off functions in the legacy code without being intimately familiar with the legacy code structure. In all cases, the original legacy binary OFP code remains intact with *no* changes.

Another key component of the COTS software strategy is the availability of a source level, symbolic user console for system developers. A tool has been developed to facilitate the use of RePLACE in modern microprocessor technologies and is referred to as the virtual integration environment workstation, or VIEWstation. It is tightly integrated with the COTS native code integrated development environment and commercial tools that are selected to support new native code development.

VIEWstation is loosely integrated with the legacy code tools. It provides a source level, symbolic configurator tool to assist the software developers in mixing and matching legacy and native code; it also provides in-target debugging and RTNI monitoring of the legacy code. VIEWstation incorporates COTS graphical data analysis tools and an interactive symbol browser/editor. Examples of VIEWstation use include downloading and disassembling software binaries, displaying real-time debug and monitoring data, performing and displaying timing

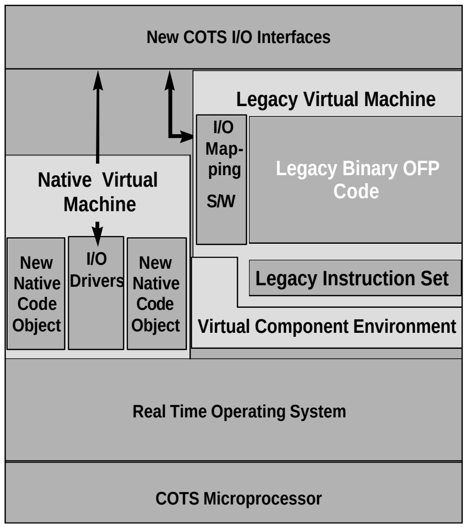


Figure 2: *A Software Perspective of RePlace*

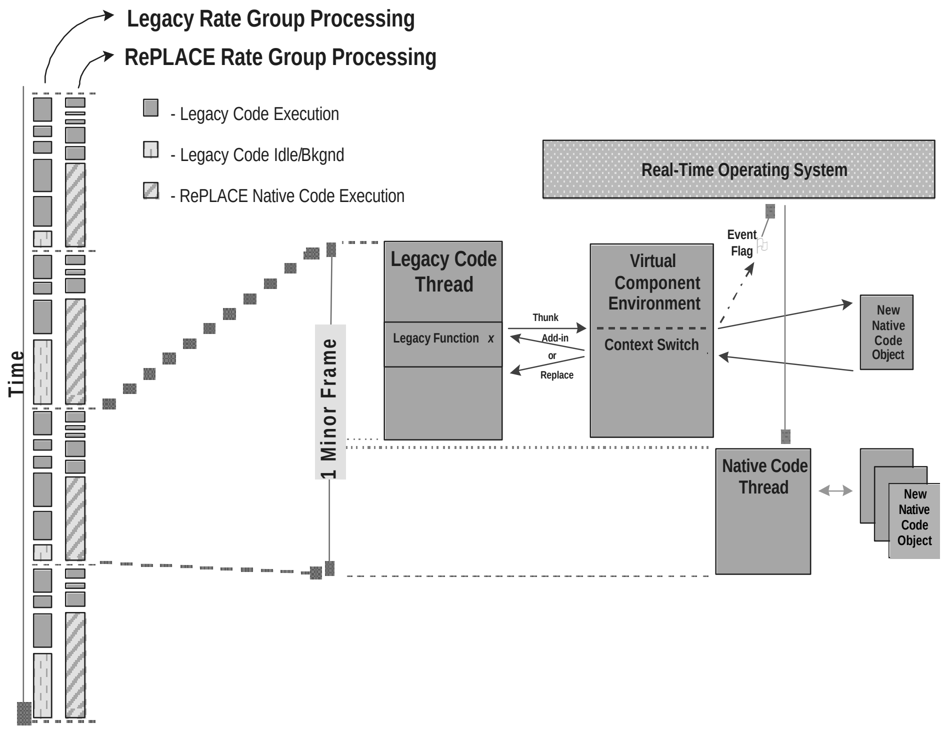
characteristics, creating and deleting events, and developing *add-in* code that is executed in response to specific user-specified events in the legacy code.

Results

Demonstrations of RePLACE executing legacy binary OFPs have been conducted in conjunction with AFRL for the following DoD aircraft subsystems:

- F-16 Heads-Up Display.
- F-16 Advanced Central Interface Unit (stores management processor).
- F-16 General Avionics Computer (fire control computer).
- AC-130H Gunship Mission Computer (SKC-3007A).
- C-17 Communication Control Unit.

Figure 3: *New Software Enhancements Introduced to Legacy Code*



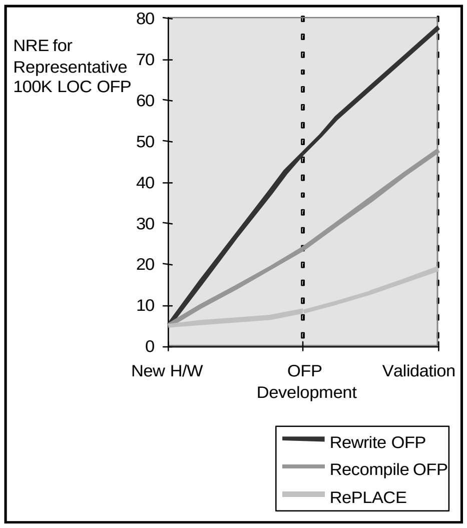


Figure 4: *Comparison of Alternative Software Adaptions*

- MH-60K Mission Computer (AP-102A).
- F-117 Mission Computer (AP-102A).

In all of these systems, legacy code performance improved from five to 20 times that of its legacy hardware environment. The cost and time to target a particular subsystem were a fraction of other approaches, including rewriting the OFP and custom chip replacements (hardware based emulation). The MIL-STD-1750A DISC has completed validation using the official acceptance test procedures and verification software originally developed and supported by the Systems Engineering Avionics Facility.

Comparisons of various alternatives to software adaptation are shown in Figure 4. The non-recurring costs for OFP development are, as would be expected, much higher for the case of an OFP rewrite. To a lesser extent, but still significant are the costs associated with an OFP rehost. This is because a rehost activity must still address new machine dependencies and the immaturity of the associated software tools that are targeted for the new hardware. In addition, under the OFP rehost strategy, incremental software upgrades are difficult to implement. As the bottom curve illustrates, the cost for OFP development with a RePLACE approach is extremely small because the existing binary OFP code is used as is – unmodified. The costs included in the figure are representative of the time to tailor the I/O mapping software to support the new hardware and to incorporate legacy software tools into VIEWstation.

Code revalidation costs are significant for all three approaches. However, these costs are minimized with RePLACE since the structure of the embedded code is not changed, allowing the replacement computer to be retested at the black-box level using the existing functional qualification tests. TRW has developed a set of analytical tools to support this type of tradeoff for different user scenarios.

Upgrade Strategies

Potential upgrade strategies using the COTS software cover a wide spectrum of upgrade possibilities. These include the introduction of a new ruggedized COTS replacement box for an existing legacy avionics LRU. It also includes the introduction of a new microprocessor replacement module for insertion into an existing avionics LRU. Finally, included is a tool to mitigate the inherent risks associated with introducing both new hardware and software into a platform at the same time.

Figure 5 illustrates determining the baseline of new COTS-based hardware with the legacy OFP as a first step in the process of adding new functionality into an existing platform. Tailoring the RePLACE DISC Legacy Instruction Set Architecture (ISA) is required to match the unique features and characteristics of the legacy machine; tailoring the VIEWstation is required to integrate with the legacy software tool-set.

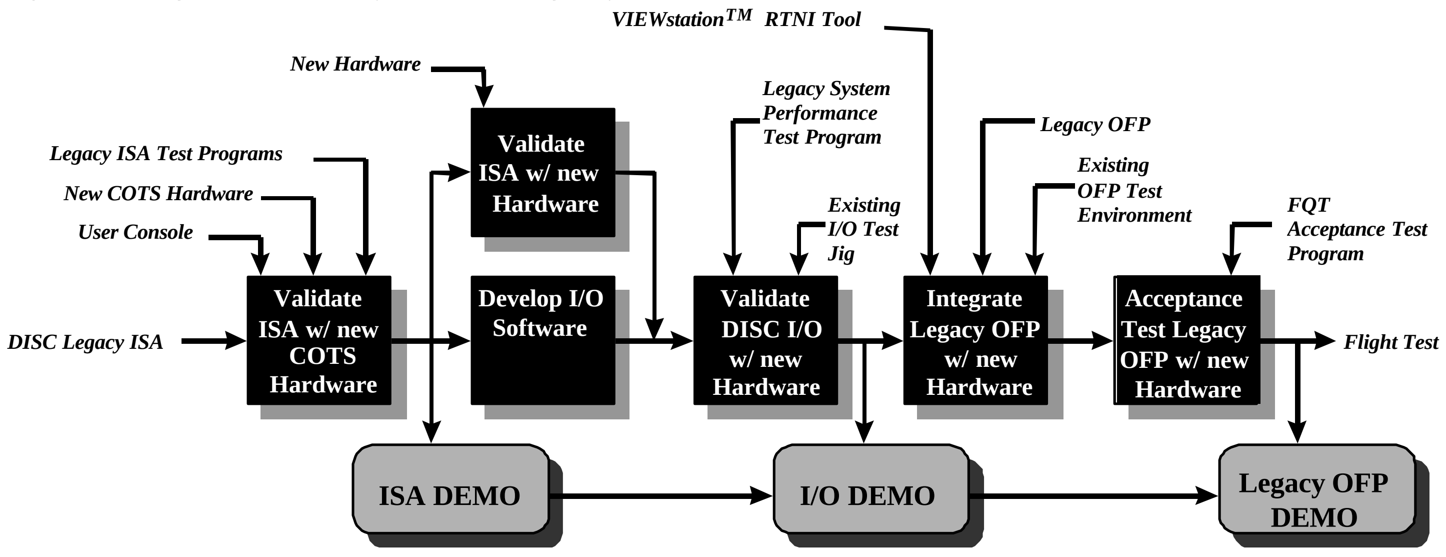
The validation steps include validating the execution of the ISA within the new microprocessor, validating the I/O characteristics using the new microprocessor, and integrating and running acceptance tests of the legacy OFP with the new hardware.

Conclusion

RePLACE technology offers the DoD customer significant supportability, producibility, and performance improvements through the introduction of COTS-based state-of-the-art hardware; it maintains backward compatibility with existing OFPs while providing the means to affordably migrate to state-of-the-art hardware and software technology. The result is significant cost savings, substantial risk reduction, and incremental upgrade opportunities for future enhancements. It also requires no custom hardware – it is an embedded software technology that leverages off the latest in commercial processor technology.

The engine can be supplied with a turnkey lab-quality or ruggedized box, or with a set of open system COTS board-level products. Also, PC-based configuration, control, and monitoring software is available to provide setup, maintenance, and user interaction with the engine. Although conceived for on-board avionics computer replacement strategies, it is equally effective to other embedded computer applications such as command and control systems, automated test equipment, weapon system trainers, and integrated support environments.◆

Figure 5: *Adding New Functionality into an Existing Platform*


VIEWstation™ RTNI Tool
New Hardware
Legacy System Performance Test Program
Legacy OFP
Legacy ISA Test Programs
New COTS Hardware
User Console
Existing OFP Test Environment
Existing I/O Test Jig
FQT Acceptance Test Program
Validate ISA w/ new Hardware
DISC Legacy ISA
Validate ISA w/ new COTS Hardware
Develop I/O Software
Validate DISC I/O w/ new Hardware
Integrate Legacy OFP w/ new Hardware
Acceptance Test Legacy OFP w/ new Hardware
Flight Test
ISA DEMO
I/O DEMO
Legacy OFP DEMO

About the Authors

Jahn A. Luke is a senior program manager at the Embedded Information Systems Engineering Branch, Information Directorate, Air Force Research Laboratory (AFRL) where he is research and development task manager for the Computer Resources Support Improvement Program. He has more than 25 years of hardware and software experience at AFRL in the development of technologies for real-time simulation systems and embedded computer software. He is currently managing projects addressing aging avionics and legacy embedded information systems within the Department of Defense.

AFRL/IFTA
2241 Avionics Cir., Bldg. 620 Rm. S3-Y57
Wright Patterson AFB, OH 45433
Phone: (937) 785-6548 ext.3585
E-mail: jahn.luke@wpafb.af.mil

Douglas G. Haldeman is currently the project manager for TRW's Group II Mission Computer Replacement Program – a mission computer upgrade for the Navy's E-2C aircraft that uses RePLACE technology. Haldeman has a bachelor's of science degree in electrical engineering from Rose-Hulman Institute of Technology and a master's of science in electrical engineering from Purdue University. He has more than 25 years experience with TRW and has been working for the last 18 years in advanced avionics development and legacy avionics upgrades.

1900 Founders Drive Suite 202
Dayton, OH 45420
Phone: (937) 259-4970
Fax: (937) 259-4900
E-mail: doug.haldeman@trw.com

William J. Cannon is the manager of Legacy Processor Technologies for the TRW Dayton Avionics Engineering Center. He has over 26 years of experience in advanced embedded computer technology. Cannon's responsibilities include the assessment of legacy embedded computer systems upgrade opportunities and the technical oversight and direction of the development and application of the Reconfigurable Processor for Legacy Applications Code Execution (RePLACE) technology that is the subject of this paper. Cannon was the principle inventor of this technology.

1900 Founders Drive Suite 202
Dayton, OH 45420
Phone: (937) 259-4965
Fax: (937) 259-4900
E-mail: bill.cannon@trw.com

Web Sites

Computer.org

http://computer.org

This site is the Institute of Electrical and Electronics Engineers (IEEE) Computer Society's technical and career resources Web site. The site features a search engine for the organization's 12 applications-oriented publications that foster active communication between practitioners and the research community, and its nine Transactions, research-oriented publications that document the state of the art in computer science. A search of "software legacy systems" turned up thousands of hits. The IEEE Computer Society is the world's leading organization of computer professionals with more than 100,000 members. The Computer Society is the largest of the 36 societies of IEEE. Its vision is to be the leading provider of technical information and services to the world's computing professionals. The society promotes an active exchange of information, ideas, and technological innovation among its members.

FCW.com

www.wargaming.net/Programming/69/Reengineering_index.htm

This is the Web site for Federal Computer Week magazine. Federal Computer Week is produced with emphasis on desktop, client/server, and enterprise-wide computing. A site search for Federal Computer Week turned up 226 results for legacy systems and legacy software articles.

GCN.com

www.gnc.com

This is the site for Government Computer News that reaches almost 90,000 government IT managers, delivering timely and insightful news coverage of the people and events that shape government technology.

Reengineering

www.wargaming.net/Programming/69/Reengineering_index.htm

This Web site features a list of reengineering books along with links for additional information on the publication. Topics include workflow reengineering, analyzing outsourcing, building enterprise information architectures, systems information reengineering, reengineering software, and more.

The RENAISSANCE Web

www.comp.lancs.ac.uk/projects/RenaissanceWeb/index.html

This site is intended as a resource for the entire software reengineering community. It also covers topics such as software maintenance, software evolution, reverse engineering, and software understanding. The site contains original content, taken from the ESPRIT RENAISSANCE software reengineering project, as well as a comprehensive list of other software reengineering resources on the Internet. The RENAISSANCE project is an ESPRIT funded research project into software reengineering and software evolution. One project result is a set of RENAISSANCE consultancy reports that covers architectural modeling, evolution risks economics, reverse engineering of system families, and the recycling of reusable code components.

Automated Transformation of Legacy Systems

Philip Newcomb and Randy A. Doblar
The Software Revolution, Inc.

The transformation of system applications code and database at automation levels exceeding 99 percent is now a viable approach to legacy information system modernization. The benefit of the approach is migrating the legacy system to a modern computing environment while preserving the repository of business knowledge and processes imbedded in the legacy system.

During the last 50 years, information processing systems have become the intellectual repositories for most business and government organizations. Today these organizations face the complex and costly problem of how best to restructure the installed base of outdated information processing resources while maintaining their legacy intellectual property. This legacy intellectual property continues to provide value as organizations are forced to innovate to survive in the fast-paced age of e-business, e-communication, e-organizations, and in the case of the military, e-warfare.

The need to modernize legacy systems is primarily driven by three factors: expansion of the system's functionality; improved maintainability of the system using modern tools and techniques; and reduction of operational costs and improved reliability by replacing obsolete hardware suites with high-speed, open-architecture systems. Alternative solutions for modernization of the system include developing a new system, system replacement with a commercial off-the-shelf (COTS) solution, or manual rewriting of the legacy applications software and databases to operate within a modern computing environment.

Developing a totally new system or replacing legacy systems by manually rewriting the system's software with the support of semi-automated tools is extremely costly and time consuming. Replacing the system using COTS technologies, while less costly and timelier, usually requires extensive and expensive customization to provide functionality not provided by the COTS product. In addition, the Gartner Group has shown that the success rate for information system modernization projects using these traditional solutions has thus far been approximately 7 percent; it is a success rate that has not bred confidence within the information technology (IT) community.

Manual approaches have been prone to failure due to inconsistency, cost overruns, and schedule delays. COTS solutions have fallen short of expectations because of their inability to provide the customer with the functionality needed to meet its specific organizational goals. And semi-automated tool-based solutions, while relatively promising, have not provided a sufficient level of automation to overcome the drawbacks associated with manual intervention required to address untransformed code.

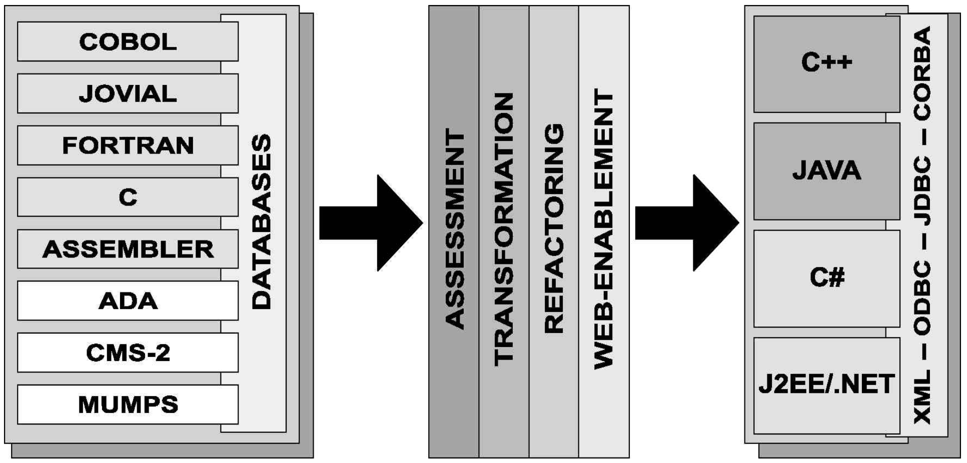
Looking more closely at the automated transformation approach, it becomes evident that using available tools capable of transforming 60 percent or less of a system's legacy code automatically results in extensive amounts of untransformed code that must be addressed manually. Using a one million-line information system as an example, 400,000 lines of code would remain to be addressed manually.

The Gartner Group, an industry-recognized source of business and technology intelligence, states that on average a well-trained programmer can transform 160 lines of code per day. Continuing to use a one million-line information system and a transformation tool capable of only 60 percent automation as our example, the resultant 400,000 lines of untransformed code would require 2,500 man-days of manual intervention. If however, a transformation tool was available that provided a 99 percent level of automation, that same one million-line system would only have 10,000 lines of untransformed code to be addressed in 62.5 man-days of manual intervention – a significant qualitative and quantitative improvement. Increasing the automation level therefore, as it relates to the quality and degree of transformation completeness is highly desirable and is a significant factor in selecting the optimal automated transformation solution.

Through the application of a suite of artificial intelligence (AI) technology tools, it is now possible to achieve levels of automation often exceeding 99 percent to assess, transform, re-factor or re-engineer, and if desired, web-enable a wide variety of legacy computer programming languages, along with system databases, into modern, platform-independent object-oriented software environments (Figure 1).

Much of the work to develop such a highly automated legacy system modernization technology originated from the Air Force-funded Knowledge-Based Software Assistant (KBSA) transformation research program in the late 1980s and early 1990s. The focus of that program was research to develop highly automated processes for program specification and synthesis. Program transformation technologies were adapted to achieve a highly extensible and adaptable tool suite and technology base to support the recreation of legacy system code in a new target language with minimal manual intervention. One of the author's work in support of KBSA at

Figure 1: *Legacy System Transformation Solution Space*



Boeing's Research and Technology Laboratory laid the foundation for the currently available automated technology. The automation levels currently being achieved for the vast majority of transformation tasks have exceeded 99 percent.

Employing this highly automated approach, legacy systems can be modernized in a fraction of the time and cost needed by the competing alternatives discussed, thus dramatically reducing the time associated with return on investment. The application of the AI technology also reduces the technical and schedule risks associated with the modernization process, while simultaneously reducing the flow time of the project. The process provides a fully documented, functional system that is in a state to be maintained and upgraded using modern tools and software workbenches.

Examples of performance metrics for projects recently addressed using this technology include a 40-to-1 reduction in functional testing time for a JOVIAL transformation task of 250,000 lines that enjoyed an automation level of 99.98 percent and a functional test of 560,000 lines of COBOL transformed at 99.99 percent that only identified 400 *bugs* during functional testing, many of which were resident in the original COBOL system. These same automated tools have also recently been applied to the transformation of C, FORTRAN, and BAL Assembler systems with levels of automation that have been demonstrated to exceed 99.99 percent for large systems.

The newly generated software also has the benefit of consistent quality and uniformity because an automated tool created it. Systems comprised of large quantities of code, if addressed manually, will require many programmers. Programmers, though writing code in the same languages, have different styles and skills. Those differences can create major difficulties during the system integration and testing phase of a transformation project. A highly automated approach requires negligible manual intervention, offers a solution that facilitates the uniformity of the code and thus, compresses the integration and testing schedule for the project.

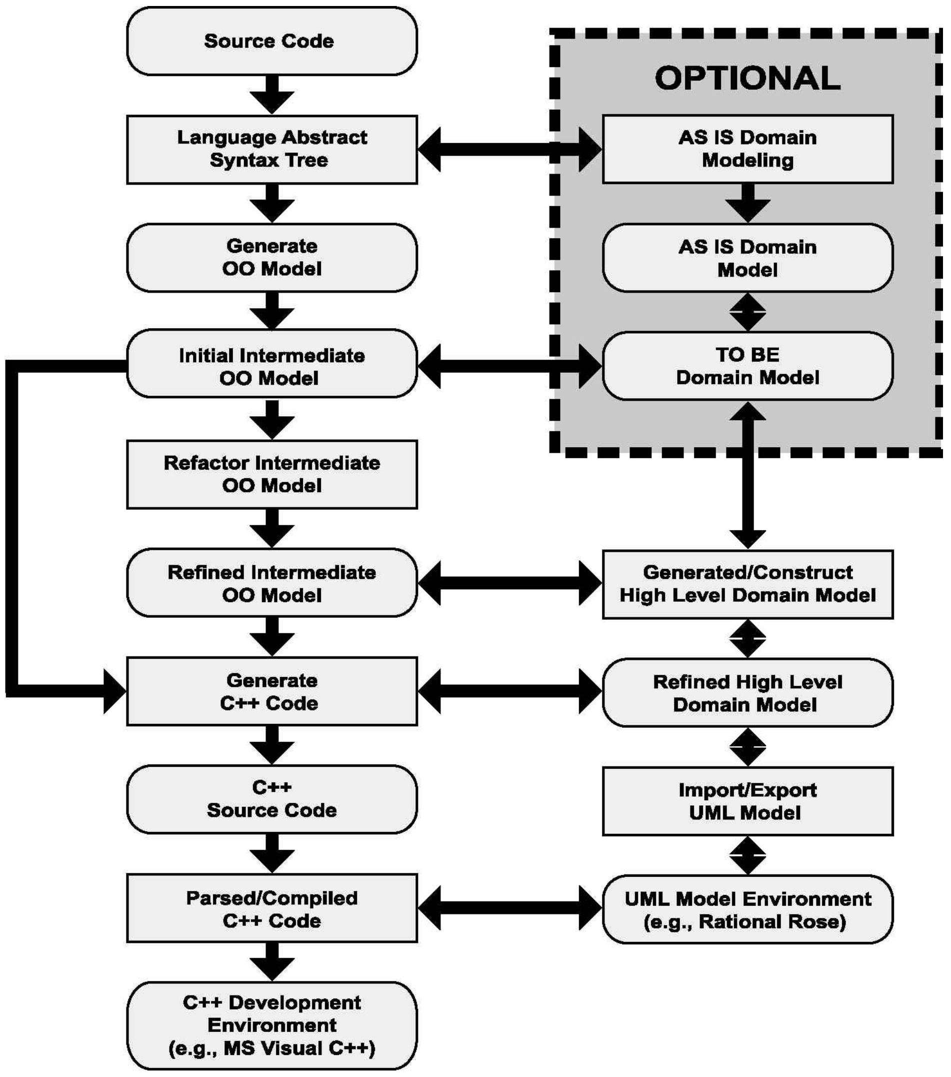

Source Code
Language Abstract Syntax Tree
Generate OO Model
Initial Intermediate OO Model
Refactor Intermediate OO Model
Refined Intermediate OO Model
Generate C++ Code
C++ Source Code
Parsed/Compiled C++ Code
C++ Development Environment (e.g., MS Visual C++)
OPTIONAL
AS IS Domain Modeling
AS IS Domain Model
TO BE Domain Model
Generated/Construct High Level Domain Model
Refined High Level Domain Model
Import/Export UML Model
UML Model Environment (e.g., Rational Rose)

Figure 2: *Automated Modernization of Legacy System into C++*

Technology Description

By employing AI-based automated program transformation technologies, the legacy application and database modernization process can be addressed in four disciplined steps:

- Assessment: Captures the legacy system's *As Is* state by extracting properties of the existing system's design, and simultaneously generating detailed documentation of the system.
- Transformation: Provides transformed software that is compiler-ready and testable at the unit level, and fully documented.
- Refactoring: Reengineers the new system to improve system architecture and performance. The refactoring process provides a disciplined approach to design improvement that minimizes the chances of introducing new flaws.
- Web-enablement: Facilitates migration of the new system to the Web environment by transforming the legacy application to Java that runs on a Java Virtual Machine.

Figure 2 illustrates the process for automated legacy system modernization through the assessment, transformation, and refactoring processes. A discussion of these three building blocks along with associated Web-enablement of a modernized system follows.

Assessment

The code is parsed to build an in-memory knowledge-based abstract syntax tree (KBAST) model of the entire system. An inventory is developed, using an iterative process, against the KBAST model to determine if any components of the application system are missing, detect multiple versions of code, and identify linkage problems. Deviations of the dialects from standard are typically not well documented; this makes it necessary to develop a series of modifications to the parser before the technology addresses all constructs of the applications code.

After development of the KBAST model, a preliminary transformation of the source code into the target *To Be* model is performed. The purpose of this effort is to assess and compare the *As Is* and *To Be* system models to determine what modifications are to be made to the transformation process to achieve a highly automated transformation into the target language.

A *dry run* of the transformation

process is performed by creating an intermediate object-oriented (IOO) model to develop the transformation metrics, including identification of the percentage of redundant and re-usable code, current and predicted code properties, and potential code and data size reductions possible in the refactoring process. The code is then transformed into the IOO formalism that allows for detailed identification and assessment of the properties of the target system. Of key significance for reuse and maintenance is 1) extraction, parameterization, and merging of derived methods associated with derived classes; and 2) measurement of the amount of decoupling and degree of cohesion and coherence of the resultant system.

The final step of the assessment process involves domain analysis of a system, which is a process that systematically creates a common framework for describing program elements and situations within the code. This descriptive framework facilitates recognition of unique and common roles and relationships among one or more systems. The framework has two tightly related dimensions of analysis that address both the classifications of identifiers (data and structures) as well as the situations that involve their usage.

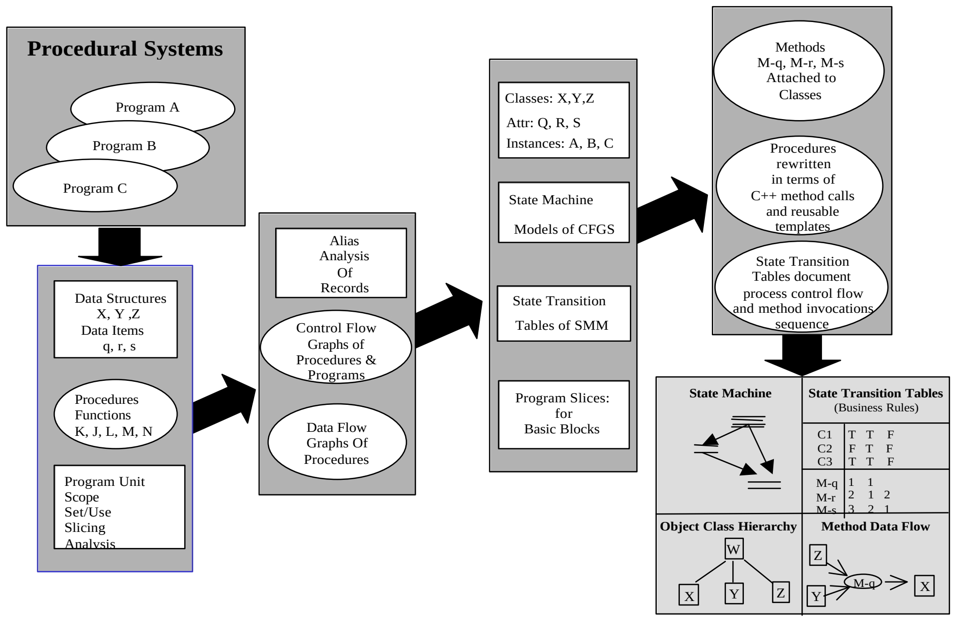
The construction of the first dimension of analysis entails describing the individual names that occur within a system as elements of common terms within a domain dictionary.

The second dimension describes the more complex relationships among elements in the form of interpretations. The interpretations are denotations for complex situations in the code. The AI-based technology automatically constructs interpretations by rewriting code directly from the structures represented by the code's abstract syntax in the KBAST. Interpretations resemble the sentential and syntactic form of the code except that domain dictionary terms have been substituted for the identifier names in the program code. A single interpretation will therefore match many syntactically similar but terminologically different specific situations. These situations may occur in the code and serve to identify commonality among complex relationships that occur within a single system or span multiple systems.

Interpretations are stored within annotation libraries and used within the technology for documenting decisions about the situations in the code. Interpretations are automatically generated for individual program statements, data structure definitions, basic code blocks, functions, or entire programs. These interpretations range in specificity from generic to domain-specific interpretations. The degree of specificity or generality in the interpretation depends upon the relative generality or specificity of the type denotation substitutions. The size of the interpretation depends upon the user-directed or system-directed choice of context of interest.

Domain analysis assists in identifying issues and opportunities in the transformation and refactoring process. It only requires manual intervention when it is necessary to establish a taxonomic system to support classification tasks. An example would be the identification or comparison of sets of code-level statement functions and data structures that are of specific interest such as the data-related usages in an entire application.

Figure 3: *Architecture of the Avionics Simulation Station*



Procedural Systems
Program A
Program B
Program C
Data Structures X, Y ,Z Data Items q, r, s
Procedures Functions K, J, L, M, N
Program Unit Scope Set/Use Slicing Analysis
Alias Analysis Of Records
Control Flow Graphs of Procedures & Programs
Data Flow Graphs Of Procedures
Classes: X,Y,Z
Attr: Q, R, S
Instances: A, B, C
State Machine Models of CFGS
State Transition Tables of SMM
Program Slices: for Basic Blocks
Methods M-q, M-r, M-s Attached to Classes
Procedures rewritten in terms of C++ method calls and reusable templates
State Transition Tables document process control flow and method invocations sequence
State Machine
State Transition Tables (Business Rules)
C1 T T F
C2 F T F
C3 T T F
M-q 1 1
M-r 2 1 2
M-s 3 2 1
Object Class Hierarchy
W
X
Y
Z
Method Data Flow
Z
Y
M-q
X

Transformation

The modernization process begins with the automatic identification of candidate classes and objects for output into an IOO model. This is a relatively complete transformation of the input source code into an IOO model that is consistent with the structure of object-oriented (OO) C++. This transformation into the IOO form locates redundant, duplicate, and similar data and processes, and abstracts those detected items into classes and methods. The classes, relationships, attributes, and operations of the derived IOO model conform to Universal Modeling Language (UML) standards.

The overall process for transforming from a procedural to an OO application starts with input of legacy application programs and produces as output a completely integrated and modernized system. The output system consists of object classes and their instances that are complete with regard to data typing, methods, and IOO processes (executable mission-oriented C++ functions, which refer to class member element and member functions or methods). These constructs possess a derived architecture and control structure consistent with the OO programming paradigm. The IOO application contains calls to derived methods associated with desired classes. Figure 3 provides a more specific depiction of the IOO model generation process.

The design documentation extracted from the IOO model is a hybrid between conventional OO modeling languages and event-driven programming models. The mapping from procedural code into OO code is functionally faithful to the original procedural system. However, this IOO model follows the semantic and syntactic rules of the OO languages C++ and Java.

It should be emphasized that variations in the transformation process do not come for free. The effort of adapting the transformation technology base to customer-specific objectives typically requires tailoring the transformation pathways to customer-specific objectives. This is the principle effort performed by a transformation service provider in a highly automated system modernization process.

Support for the system transformation processes comprises a wide range of tasks. Some of the components include physical transformation of the application software, user interface, databases, and platform adaptation. Table 1 illustrates the overall set of activities in modernizing an information system. The principal roles and responsibilities of the transformation technology provider, major system integrator, and customer are indicated in the column headings.

The principal project phases are identified in the table cells in Table 1. Column one task components include application transformation, user interface transformation, database transformation, and platform adaptation, and can be addressed with a very high level of automation. The second column involves functional testing of all major components to assure functional integrity of the resultant system in the new target environment. The final column encompasses deployment and retraining of the user base.

In a typical engagement, the user interface is automatically transformed into a functionally equivalent user interface in the target environment. The database is converted using information derived from the system database descriptions that drive database conversion programs. Platform and operating system calls are transformed into equivalent target environment services.

Generally, we have found it possible to automate virtually all of the application, database, user-interface, and platform adaptation tasks. Automation has not been introduced where manual processes are already adequate such as the specification of index fields in a relational database using the interface provided by the database vendor, testing the system user-interface, or identification and reporting of system bugs and enhancements.

Throughout the development of an automated process the largest unknowns have been resolving configuration discrepancies that surfaced because the original source code was either incomplete or could not be demonstrated to build the original system, and determining the functionality of undocumented legacy system calls. The most significant areas of effort have been in the replacement of legacy platform functionality that did not exist in the target environment, and enhancing the dialect-specific language constructs.

Refactoring

Refactoring is the process of changing a software system to improve the software's structure and performance without altering its functional behavior. Refactoring is used to eliminate, replace, or rewrite code to improve its efficiency and understandability or to transform applications to use a suitable set of modern infrastructure support functions. Refactoring extracts and parameterizes methods; merges and consolidates similar methods; reduces the set of methods associated with a class to the minimal set of well-understood operations; improves the coupling, cohesion and comprehensibility of the overall application; and reduces overall code duplication and code redundancy.

Transformation Technology Provider	Major Systems Integrator	Integrator and Customer
Platform Adaptation	Platform Adaptation Function Test	Platform Adaptation Deployment
User Interface Transformation	User Interface Transformation Reqt's Function Test	User Interface Transformation Reqt's Deployment
Data Base Transformation	Data Transformation Function Test	Data Transformation Deployment
Application Transformation	Application Functional Test	Application Deployment

Table 1: *Legacy System Modernization Services and Responsibilities*

Legacy applications often have many dependencies on the legacy software infrastructure. Consequently, it is often not directly portable to another software environment. Modernization of applications often requires isolating the application-specific code from the code associated with various environment-specific utility/support functions known as infrastructure code.

Often the legacy infrastructure functionality does not exist in the new environment, or exists in a different form. Thus, this layer of support services must be discontinued, redeveloped, or suitably replaced. The definition or introduction of an appropriate interface to the facilities/services layer into the newly derived application requires the development of new code or an application program interface (API) layer to comparable support services in the target platform. This is also required to test both the interface and the service layers accessed through the facilities/services layer interface.

Adaptation of the application to interface with a target environment not previously encountered is the most manually intensive effort in a transformation project. In our experience, this adaptation has been the single largest source of schedule risk, though it is not in general, a significant technical risk. Generally lessons learned from these kinds of adaptation tasks are transferable between projects. Separation of these infrastructure layers from the application layers via a suitable interface layer expedites resolution of errors that arise from variations in alternative target operational environments.

Web-Enablement

Web-enablement entails the transformation of an application into a networked, distributed application that makes use of the following:

- Browser user-interfaces (BUI).
- Web-based languages.
- Run-time environments such as Java and the Java Virtual Machine (JVM).
- Web-based data transmission and manipulation protocols such as the Extended Markup Language for the interchange of data.

Hybrid Web applications exhibit some, but not necessarily all of these features. A Web-application may be written in C++ and have a BUI front-end that supports interface with users, or a back-end database with connectivity via Microsoft's Object Data Base Connectivity (ODBC). Java applications typically employ Java Data Base Connectivity to connect to various vendor database products with similar functionality as ODBC.

The Common Object Request Broker Architecture (CORBA) has a common interface definition language (IDL) that supports both Java and C++. CORBA is commonly used to provide distributed component services for a smaller number of users with high-performance requirements. CORBA's IDL facilitates the creation of networked distributed applications by simplifying the definition of interfaces that allow components to call one another that reside anywhere in the network, and may be implemented in any language that supports IDL. A component architecture such as J2EE provides standardized, extensible server-side and client-side components to provide multi-tier distributed services. Java more typically uses remote method invocation than CORBA IDL for transferring data between distributed components.

Support for both CORBA and Enterprise Java Beans for distributed component management services can coexist in large applications. For instance, CORBA with C++ can be used for highly optimized transaction-oriented database applications, while Java Enterprise Java Beans and Java Applets are often used for highly interactive distributed applications.

Given the multiple business processes performed by the applications within large confederated legacy systems and the tradeoffs between several possible alternative distributed component Web-based architectures, the definition of the most appropriate transformation approach is a complex decision to be evaluated. The decision requires an in-depth analysis of the customer's applications, analysis of the implications of alternative solutions, and possibly some amount of iteration to define an appropriate transformation pathway. This decision process is driven by the current system architecture; the target architecture objectives; the technical infrastructure, including overall tool-set capabilities; and personnel resources available to support this transformation process.

Conclusion

While transforming legacy source code to C++ at an automation level of 99.99 percent is achievable using the approach described here, experience has also shown that it is unwise to take too many steps along the modernization pathway at one time. Hence, one should regard the initial transformation into C++ as an easily achievable goal that provides the staging point for subsequent phases, including system refactoring, confederated system consolidation, and Web-enablement.

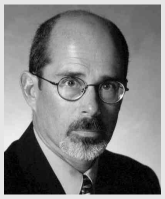
It should be emphasized that while high levels of automation are achievable for transformation tasks, a modernization project also involves development and adaptation tasks that are manually intensive such as infrastructure API layer development for unfamiliar legacy or target platforms. In addition, there are a number of tasks that by their very nature require human guidance or description such as certain forms of domain analysis and refactoring tasks that require specific domain expertise. We have however, been successful at minimizing the effort associated with these tasks by providing high levels of machine support for them as well.

Nevertheless, as today's organizations address the critical structural, cultural, and financial issues surrounding the migration of their often irreplaceable legacy software applications and databases to modern platform-independent computing environments, it is essential that they understand that a new automated low-risk approach is available. Exceedingly advanced tool-sets and processes for rapidly reengineering legacy system software into modern computing environments provides organizations with a valuable new alternative that is faster, lower cost, lower risk, and higher quality than other methods currently available.◆

About the Authors

Philip Newcomb is an internationally recognized expert in the application of artificial intelligence (AI) and formal methods of software engineering, and has published numerous papers in his field. He has done groundbreaking research in applying AI, software engineering, and automatic programming. He formulated the conceptual product framework and developed the software transformation technology and products offered by The Software Revolution, Inc. He has graduate work and degrees from Carnegie Mellon University, the University of Washington, Ball State University and Indiana University.

The Software Revolution, Inc.
3330 Monte Villa Pkwy.
Bothell, WA 98021
Phone: (425) 354-6464
Fax: (425) 354-6465
E-mail: newcomb@softwarerevolution.com



Randy A. Doblar, vice president, Sales and Marketing, for The Software Revolution, Inc., has more than 28 years experience in program management and business development in the defense and commercial marketplace. He has published papers in the scientific disciplines of acoustics, geophysics, and oceanography, and has been instrumental in leading The Software Revolution, Inc.'s effort to establish the eVolution 2000TM technology as a new industry standard for legacy system modernization.

The Software Revolution, Inc.
3330 Monte Villa Pkwy.
Bothell, WA 98021
Phone: (425) 354-6480
Fax: (425) 354-6465
E-mail: rdoblar@softwarerevolution.com

Balancing Discipline and Flexibility With the Spiral Model and MBASE

Dr. Barry Boehm and Dr. Daniel Port
University of Southern California, Center for Software Engineering

This article details how the spiral model and its recent extension, Model-Based Architecting and Software Engineering (MBASE) can be used to tailor a project's balance of discipline and flexibility via risk considerations. It also describes and rationalizes the major MBASE extensions to the spiral model – model clash avoidance, stakeholder win-win – and elaborates on the use of these extensions and risk considerations in the anchor-point milestones used in MBASE and the spiral model.

In his keynote address at the 1996 International Conference on Software Engineering, Tom De Marco summarized the work of the great military analyst Karl Von Clausewitz on the interplay of armor and mobility in military conflict. De Marco said Clausewitz proposed that at times, armor would dominate mobility, as with heavily armed medieval knights dominating lightly armed peasantry. But if over-optimized, one strategy will lose to advances in the other, as the ponderous French knights found in their inability to dominate the lightly armed and mobile English longbowmen in their watershed loss to the English at Crecy in 1346.

De Marco then drew a parallel between *armor-intensive* software strategies such as the software Capability Maturity Model® (CMM®) and the *mobility-intensive* lightweight processes that were emerging at the time. He was inferring that the software CMM was too ponderous to cope with the need for rapid development and rapid change characteristic of such sectors as electronic commerce and Web-based systems. In the ensuing discussion, software CMM advocates cited the high mortality rates of lightweight process organizations, and their frequent inability to cope with success when they need to scale up their process and architectures to deal with more complex services and heavier workloads.

Underlying this point/counterpoint is a key software-engineering question: How much discipline is enough, and how much flexibility is enough?

In *Understanding the Spiral Model as a Tool for Evolutionary Acquisition* [1], we showed that the risk exposure considerations used as spiral model decision criteria could be used to address *how-much-is-enough* questions. There, we showed how a *how-much-testing-is-enough* question could be addressed by balancing the risks of doing too little testing (alienating your users) and the risks of doing too much testing (unavailable combat capability, missed market windows).

In this article, we show how the spiral model and its recent extension, Model-Based (system) Architecting and Software Engineering (MBASE), can be used to tailor a project's balance of discipline and flexibility via risk considerations. We also describe and rationalize the major MBASE extensions to the spiral model (model clash avoidance, stakeholder win-win), and elaborate on the use of these extensions and risk considerations in the anchor-point milestones used in MBASE and the spiral model.

In subsequent articles, we will present an attractive special case of MBASE, the Schedule as an Independent Variable (SAIV) process model, and present an integration of the MBASE project approach with the University of Maryland's Experience Factory approach, which facilitates an organization's transition to the Capability Maturity Model®-Integrated℠ (CMMI℠).

MBASE and Model Clash Avoidance

Particularly for an invisible product such as software, projects make use of various process, product, property, and success models to guide its progress. *Process models* can include the waterfall model (sequential determination of the system's requirements, design, and code); evolutionary development (development of an initial core capability, with full definition of future increments deferred); incremental development; spiral development; rapid application development; adaptive development; and many others.

Product models can include various ways of specifying operational concepts, requirements, architectures, designs, and code, along with their interrelationships; for example, the object-oriented design models specified within the Rational Unified Process, e.g., class and sequence diagrams, use-cases, etc.

Property models can include models of desired or acceptable cost, schedule, performance, reliability, security, portability, evolvability, reusability, etc., and their tradeoffs, e.g., Constructive Cost Model II (COCOMO II).

Success Models can include correctness (satisfying the specified requirements), organization and project goals, stakeholder win-win, business-case, Goal-Question-Metric (GQM), or others such as IKIWISI (I'll know it when I see it: a frequent response when users are asked to specify user-interface requirements).

Software architects and product developers are generally familiar with the concept that trying to integrate two or more arbitrarily selected products or product models can lead to serious conflicts and disasters. Some examples are mixing functional and object-oriented components, or the architectural style clashes you can encounter when integrating commercial off-the-shelf (COTS) products (see [2] for a well-described case study involving factors of four and five overruns in schedule and effort). Software process people are similarly familiar with the serious effects of trying to coordinate organizations with clashing process models such as top-down/bottom-up or CMM Level 1/Level 5.

However, relatively few people are aware of the extent to which projects can run into trouble because they choose incompatible combinations of software process, product, property, and success models. Such clashes are not only frequent and severe, but they are also hard to diagnose because they derive from different sources and involve mismatched assumptions lying deep below the project's written

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.
℠ Capability Maturity Model-Integrated and CMMI are service marks of Carnegie Mellon University.

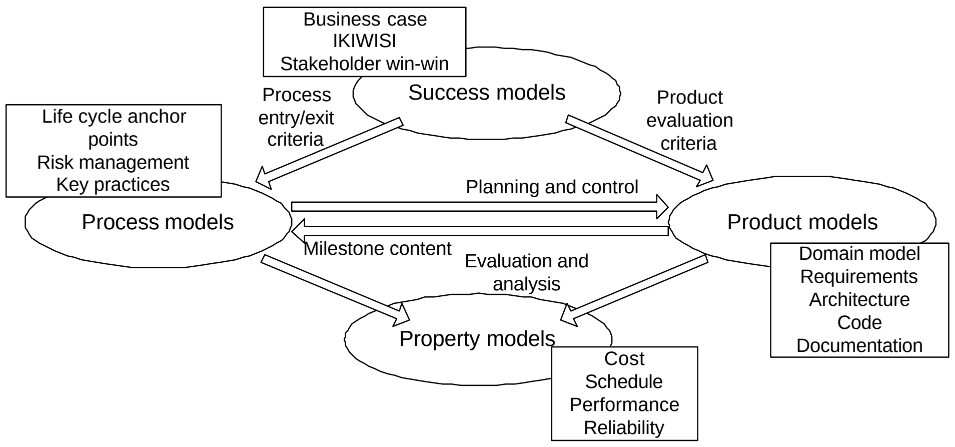


Figure 1: *Model-Based Architecting and Software Engineering Model Integration Framework*

plans and specifications.

A good Department of Defense (DoD) example was a deeply troubled project encountered during the National Research Council *Ada and Beyond* study [3]. This project inherited a commitment to a waterfall process model via its organization's commitment to DoD-STD-2167. It also inherited a commitment to COTS-based product model through the need to comply with a secretary of defense mandate.

The waterfall model assumes that the requirements determine the capabilities, and the project had contracted for a two-second response time requirement. However, the contractor found that none of the available COTS capabilities could process the system workload with a two-second response time, and was proceeding to develop an expensive custom software system to meet the two-second requirement. The customer wanted the COTS-directive product model to take precedence, and to have the available COTS capabilities determine the performance requirements.

Besides this difficult model clash, the project had also inherited an Ada-based product model via the DoD Ada mandate. This clashed with the COTS model, since some attractive COTS solutions did not have adequate Ada bindings. Further, none of the approaches were compatible with the project's success model of producing an initial operational capability in 36 months; or with an additional property model, which was being adopted by the organization. This model was *Cost As Independent Variable*, which would have been difficult to satisfy when the project already had at least four existing independent variables.

This example covers only a small subset of the model clashes a project can encounter. Further discussions, examples, and case studies can be found in [4] and [5]. We have been able to use MBASE to help other large government and commercial projects to diagnose and avoid serious model clashes, and have worked with smaller companies to develop lightweight versions of MBASE to balance discipline and flexibility on rapid-development projects. After describing MBASE in the next four sections, we will summarize how its use could have avoided the DoD project model clash dilemmas noted, followed by a summary of MBASE usage to date.

MBASE Model Integration Framework and Process Framework

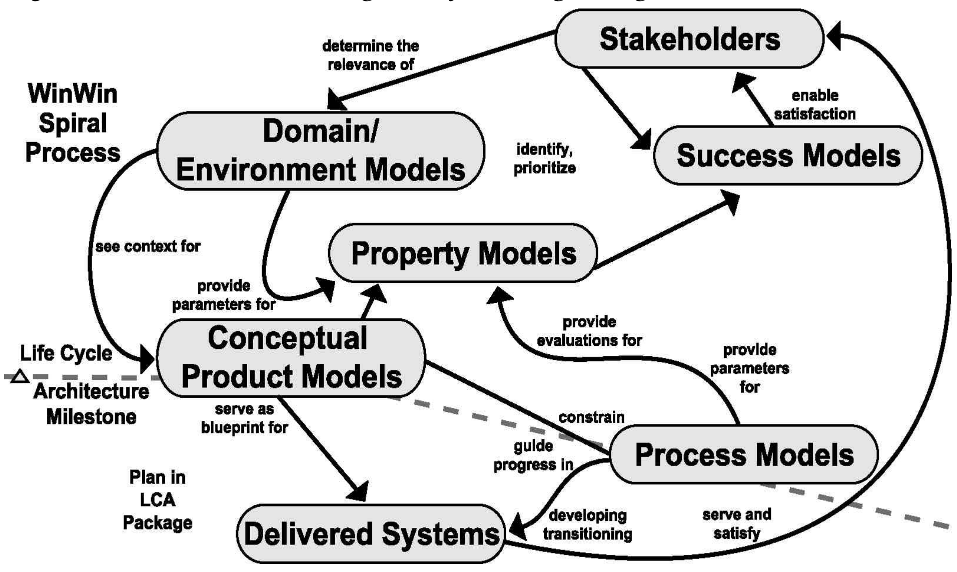
Figure 1 summarizes the overall model integration framework used in the MBASE approach to ensure that a project's success, product, process, and property models are consistent and well integrated. At the top of Figure 1 are success models, illustrated with several examples, whose priorities and consistency should be considered first as they tend to drive the selection and use of other models (including other success models).

Thus, if the overriding top-priority success model is to "demonstrate a competitive agent-based electronic commerce system on the floor of the COMDEX trade show in nine months," this constrains the ambition level of other success models. It would be a major schedule risk to insist on provably correct code or a fully documented system. The nine-month schedule constraint is most critical because the system will lose most of its value if it is not available to compete for early market share at COMDEX.

The risk schedule overrun also determines many aspects of the product model (architecture designed to easily shed lower-priority features if necessary to meet schedule), the process model (SAIV), and various property models (portable and reliable enough to achieve a successful demonstration). The achievability of the success model needs to be evaluated with respect to the other models. Figure 1 shows that the choices of process and product models need to be evaluated by having the success model provide evaluation criteria for the product milestone artifacts, and provide preconditions and postconditions (entry and exit criteria) for the process milestones.

In the nine-month COMDEX demonstration example, a cost-schedule estimation model would relate various product characteristics (sizing of components, reuse, product complexity), process characteristics (staff capabilities and experience, tool support, process maturity), and property characteristics (required reliability, cost constraints) to determine whether the product capabilities achievable in nine months would be sufficiently competitive for the success models. Thus, as shown at the bottom of Figure 1, a cost and schedule property model would be used for the

Figure 2: *Model-Based Architecting and Software Engineering Process Framework*



evaluation and analysis of the consistency of the system's product, process, and success models.

In other cases, the success model would make a process model or a product model the primary driver for model integration. An IKIWISI success model might initially establish a prototyping and evolutionary development process model leaving most of the product features and property levels to be determined by the evolutionary development process. A success model focused on developing a product line of similar products would initially focus on product models (domain models, product line architectures), with process models and property models subsequently explored to perform a business-case analysis of the most appropriate breadth of the product line and the timing for introducing individual products.

Figure 2 provides an overall process framework for the MBASE approach. The primary drivers for any system's (or product line's) characteristics are its key stakeholders. These generally include the system (taken below to mean *system or product-line*) users, customers, developers, and maintainers. Key stakeholders can also include strategic partners, marketers, operators of closely coupled systems, and the general public for such issues as safety, security, privacy, or fairness.

The critical interests of these stakeholders determine the priorities, desired levels, and acceptable levels of various system success criteria. These are reflected in the success models for the system such as stakeholder win-win, business case, organization and project goals, operational effectiveness models, or IKIWISI. These in turn determine which portions of an applications domain and its environment are relevant to consider in specifying the system and its development and evolution process. The particular objective is to determine a system boundary, within which the system is to be developed and evolved; outside of which is the system environment (and context).

For example, in our COMDEX electronic commerce application, the driving success model is the nine-month schedule. The system boundary would be determined by the most cost-effective set of capabilities that could be developed in nine months. Thus, a credit card verification capability might be considered outside the initial system boundary, although it would be needed later.

This latter point illustrates how boundaries might (and will likely) change over time, particularly in the face of evolving success models (e.g., the nature of a *competitive e-commerce system*). For example, if a compatible COTS credit card verification capability became available and easy to integrate, it could be added within the system boundary. Thus the domain scope for the demo system would be very much determined by the available COTS products that could be tailored, integrated, and built upon.

Determining the appropriate combination of COTS products and extensions could take several win-win spiral cycles of experimental prototyping and risk resolution, in concert with cost-schedule modeling to determine how much capability would be feasible to develop in nine months. The appropriate process model would be SAIV, which adds further product model constraints, such as the need to prioritize features and to design the system architecture for ease of adding or dropping marginal-priority features in order to minimize the risk of not meeting the nine-month schedule.

A Different Balance of Discipline and Flexibility: Safe Air Traffic Control

With a different set of stakeholders and success models, the same MBASE process framework in Figure 2 will produce a different balance of discipline and flexibility. In an air traffic control system, for example, the key stakeholders will include the airplane passengers and various regulatory bodies whose success models involve a very high level of system safety.

In this case, the success models will reject high-risk product models, including unreliable COTS products. The process models will include considerably more discipline to eliminate safety risks at the requirements, architecture, design, and code levels. The key property model will focus on safety rather than schedule, although schedule considerations might still affect the timing of various increments of system capability. Thus, the different risk patterns imposed by the stakeholders and their success models will produce different sequences of product capabilities and processes with different balances of discipline and flexibility.

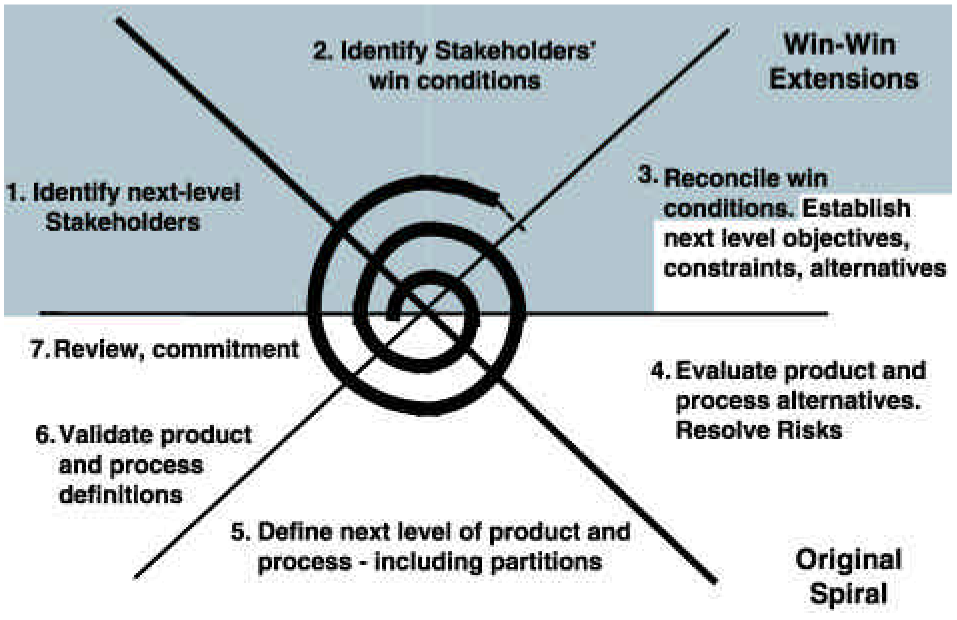
MBASE and Stakeholder Win-Win

A common element in the e-commerce and air traffic control examples is the need to reconcile the key stakeholders' success models. Thus, a stakeholder win-win negotiation process becomes a key step in each spiral cycle of the MBASE approach, as shown in Figure 3.

In the COMDEX application, for example, the initial spiral cycle would focus on evaluating COTS products and scoping the overall system to be buildable in nine months. In a subsequent spiral cycle, the next-level stakeholders would include representative users of the e-commerce system, and the reconciliation of their win conditions would include prototyping of the user interface to eliminate the risk of showing up at COMDEX with an unfriendly user interface. The MBASE tool support includes a groupware system called Easy WinWin, which enables distributed stakeholders to enter their win conditions and to negotiate mutually satisfactory (win-win) agreements with other stakeholders [6].

The win-win spiral model in Figure 3

Figure 3: *The Win-Win Spiral Model*


2. Identify Stakeholders' win conditions
Win-Win Extensions
1. Identify next-level Stakeholders
3. Reconcile win conditions. Establish next level objectives, constraints, alternatives
7. Review, commitment
4. Evaluate product and process alternatives. Resolve Risks
6. Validate product and process definitions
5. Define next level of product and process - including partitions
Original Spiral

Spiral Model Essentials
1. Concurrent determination of key artifacts.
2. Each cycle does objectives, constraints, alternatives, risks, review, and commitment to proceed.
3. Level of effort driven by risk considerations.
4. Degree of detail driven by risk considerations.
5. Use of anchor point milestones: LCO, LCA, IOC.
6. Emphasis on system and life-cycle activities and artifacts.

Table 1: *Essentials of the Spiral Model*

provides another view of how risk considerations are used to reconcile stakeholder success conditions in terms of product, process, and property models. A complementary view was shown in Figure 2 (see page 24), which also identifies the win-win spiral model's role in guiding the early feedback cycles involved in defining and reconciling the system's domain, product, process, and property models.

MBASE and Life-Cycle Anchor Points

MBASE also adopts and extends the six Spiral Model Essentials presented in our May 2001 CROSSTALK article [1] and summarized in Table 1. The *stakeholder commitment to proceed* in Essential 2 is implemented via the win-win spiral model as shown in Figure 3 (see page 25). MBASE adopts Essential 5 by using its life-cycle anchor points as critical review and management decision points. It adopts Essentials 3 and 4 on risk management via continuous risk identification, risk assessment, and risk exposure reduction, and Essentials 1 and 6 via a concurrent engineering approach to both system and software issues.

The life-cycle anchor points are described further in the side bar located on page 27. As shown in Figure 2, one of them is the Life Cycle Architecture milestone. It includes a product definition, a process definition, and a feasibility rationale ensuring the compatibility of the system's product, process, property, and success models. From this base, the project can continue to construct the system by refining the product models into an executing product up through a third milestone, the Initial Operating Capability.

The specific content of the first two anchor point milestones is summarized in the sidebar. It includes increasingly detailed, risk-driven definitions of the system's operational concept, prototypes, requirements, architectures, life-cycle plan, and feasibility rationale. For the feasibility rationale, property models are invoked to help verify that the project's success models, product models, process models, and property levels or models are acceptably consistent.

The first milestone is the Life Cycle Objectives (LCO) milestone, at which management verifies the basis for a business commitment to proceed at least through an architectural stage. This involves verifying that there is at least one system architecture and choice of COTS/reuse components that is shown to be feasible to implement within budget and schedule constraints to satisfy key stakeholder win conditions and to generate a viable investment business case.

The second milestone is the Life Cycle Architecture (LCA) milestone, at which management verifies the basis for a sound commitment to product development and evolution. This is a particular system architecture with specific COTS and reuse commitments that is shown to be feasible with respect to budget, schedule, requirements, operations concept and business case; identification and commitment of all key life-cycle stakeholders; and elimination of all critical risk items. The AT&T/Lucent Architecture Review Board technique [7] is an excellent management review approach involving the LCO and LCA milestones. It is similar to the highly successful recent DoD best practice of software Independent Expert Program Reviews [8].

Table 2: *LCO, LCA, and IOC Pass/Fail Criteria*

LCO	LCA	IOC
For at least one architecture, a system built to that architecture will: • Support the core operational concept. • Satisfy the core requirements. • Be faithful to the prototype(s). • Be buildable within the budgets and schedules in the plan. • Show a viable business case. • Have its key stakeholders committed to support the Elaboration Phase (to LCA).	For a specific detailed architecture, a system built to that architecture will: • Support the elaborated operational concept. • Satisfy the elaborated requirements. • Be faithful to the prototype(s). • Be buildable within the budgets and schedules in the plan. • Have all major risks resolved or covered by a risk management plan. • Have its key stakeholders committed to support the full life cycle.	An implemented architecture, an operational system that has: • Realized the operational concept. • Implemented the initial operational requirements. • Prepared a system operation and support plan. • Prepared the initial site(s) in which the system will be deployed for transition. • Prepared the users, operators, and maintainers to assume their operational roles.

The third anchor point is the system's Initial Operational Capability (IOC), defined further in [9]. The LCO, LCA, and IOC have become the key milestones in the Rational Unified Process [10, 11, 12]. There are many possible minor milestones (adjusted to the particular project as needed) that may lie between LCO and IOC and several important post-deployment milestones beyond IOC. Table 2 summarizes the pass/fail criteria for the LCO, LCA, and IOC anchor points.

The focus of the LCO review is to ensure that at least one architecture choice is viable from a business perspective. The focus of the LCA review is to commit to a single detailed definition of the review artifacts. The project must have either eliminated all significant risks or put in place an acceptable risk-management plan. The focus of the IOC review, also called the Transition Readiness Review, is to ensure that the initial users, operators, and maintainers (generally equivalent to beta-testers) are fully prepared to successfully operate the delivered system. If the pass/fail criteria for any review are not satisfied, the package should be reworked.

We determined these anchor point milestones as common commitment points across commercial, aerospace, and government organizations when searching with our University of Southern California (USC) Center for Software Engineering Affiliates for a set of common milestones for referencing COCOMO II cost and schedule estimates. They work well as common commitment points across a variety of process model variants because they reflect similar commitment points during one's lifetime.

The LCO milestone is the equivalent of getting engaged, and the LCA milestone is the equivalent of getting married. As in life, if you marry your architecture in haste, you and your stakeholders will repent at leisure (if, in Internet time, any leisure time is available). The third anchor point milestone, the IOC, constitutes an even larger commitment: It is the equivalent of having your first child with all the associated commitments of care and feeding of a legacy system.

To return to our DoD 2167/COTS/Ada/deadline/CAIV model clash example on page 24, at the latest it would have failed its LCO milestone review by being unable to demonstrate that a COTS-based architecture could satisfy the two-second response time requirement. Even earlier, though, this model clash would have been picked up by the MBASE process framework in Figure 2, in feeding back to the stakeholders the need to revise their success models to permit a clash-free solution. This would involve additional win-win spiral cycles to determine a mutually satisfactory (win-win) combination of features, budgets, schedules, increments, and COTS choices.

MBASE Usage Experience

For the past five years, USC has used and refined MBASE extensively within its two-semester graduate software-engineering course. The students work on a Web-based electronic services project for a real USC client (frequently a digital library application for the university information services division) from initial system definition through transition, utilizing a specialized form of MBASE. This specialization includes particular tools and models such as Easy WinWin, Rational Rose, MSProject, and elements of the Rational Unified Process. More than 100 real-client projects have used MBASE, and over 90 percent have delivered highly satisfactory products on very short fixed schedules. The annual lessons learned have been organized into an extensive set of usage guidelines and an Electronic Process Guide [13], all accessible at <http://sunset.usc.edu/research/MBASE>. In the spring of 1999, MBASE was used in both the undergraduate and graduate software engineering courses at Columbia University. Although these are single semester courses, MBASE was successfully adapted to help student teams complete a full project life cycle for real clients.

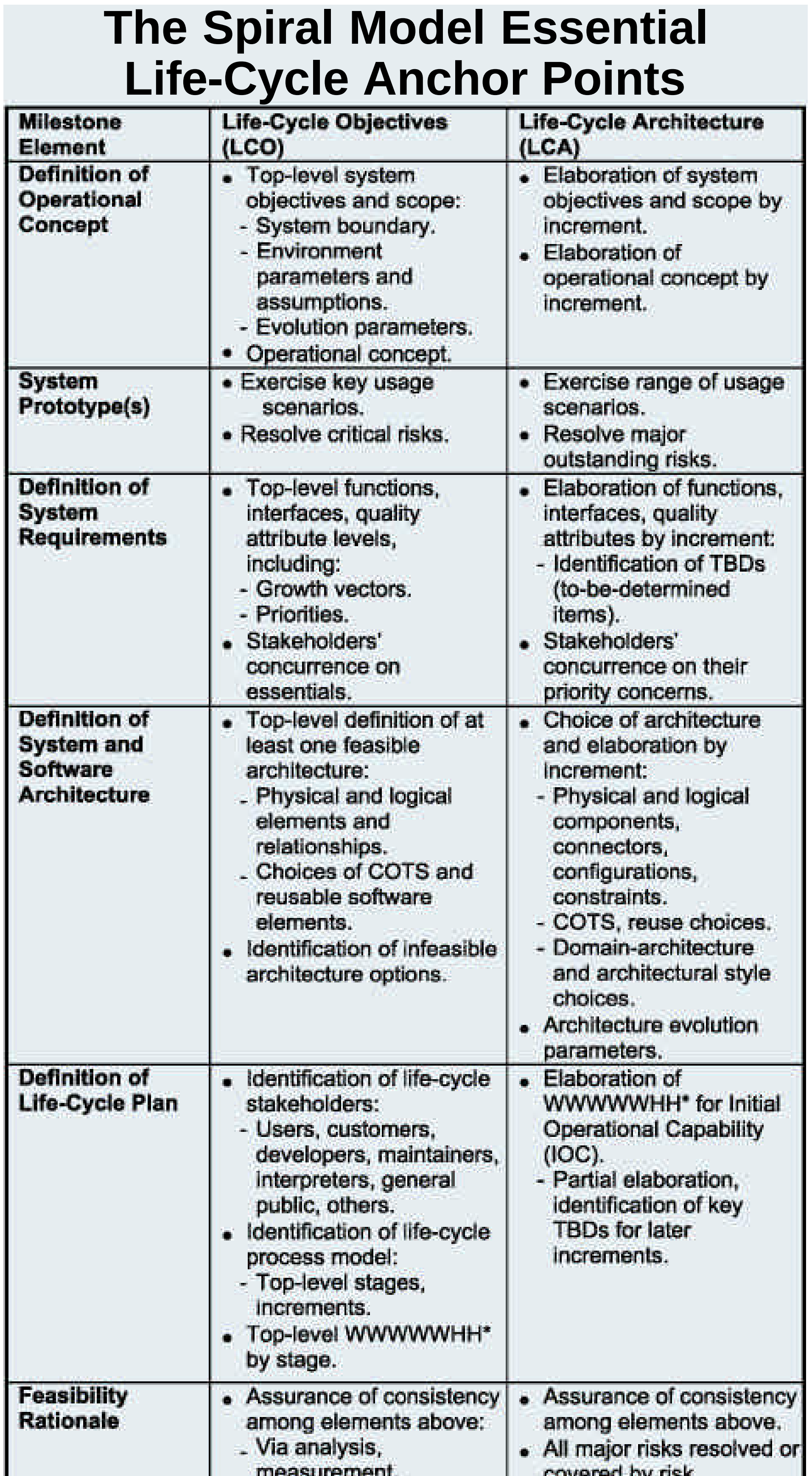
Within industry, Xerox has adopted many elements of MBASE to form its *time-to-market* process, including the use of the LCO and LCA anchor points as synchronization points for the hardware and software portions of their printer product definitions.

As mentioned previously, Rational has adopted the LCO, LCA, and IOC anchor points within their Rational Unified Process while MBASE adopted Rational's Inception-Elaboration-Construction-Transition phase definitions.

C-Bridge has mapped their *define*, *design*, *develop*, *deploy* rapid development methodology for e-commerce systems to the MBASE spiral model.

The Internet startup company Media Connex adopted MBASE and used Easy WinWin to establish win-win relationships among their key stakeholders. Each of these companies converged on different balances of discipline and flexibility to satisfy their stakeholders' success models.

Additionally, there are numerous companies and organizations directly making



The Spiral Model Essential Life-Cycle Anchor Points

Milestone Element	Life-Cycle Objectives (LCO)	Life-Cycle Architecture (LCA)
Definition of Operational Concept	• Top-level system objectives and scope: - System boundary. - Environment parameters and assumptions. - Evolution parameters. • Operational concept.	• Elaboration of system objectives and scope by increment. • Elaboration of operational concept by increment.
System Prototype(s)	• Exercise key usage scenarios. • Resolve critical risks.	• Exercise range of usage scenarios. • Resolve major outstanding risks.
Definition of System Requirements	• Top-level functions, interfaces, quality attribute levels, including: - Growth vectors. - Priorities. • Stakeholders' concurrence on essentials.	• Elaboration of functions, interfaces, quality attributes by increment: - Identification of TBDs (to-be-determined items). • Stakeholders' concurrence on their priority concerns.
Definition of System and Software Architecture	• Top-level definition of at least one feasible architecture: - Physical and logical elements and relationships. - Choices of COTS and reusable software elements. • Identification of infeasible architecture options.	• Choice of architecture and elaboration by increment: - Physical and logical components, connectors, configurations, constraints. - COTS, reuse choices. - Domain-architecture and architectural style choices. • Architecture evolution parameters.
Definition of Life-Cycle Plan	• Identification of life-cycle stakeholders: - Users, customers, developers, maintainers, interpreters, general public, others. • Identification of life-cycle process model: - Top-level stages, increments. • Top-level WWWWWHH* by stage.	• Elaboration of WWWWWHH* for Initial Operational Capability (IOC). - Partial elaboration, identification of key TBDs for later increments.
Feasibility Rationale	• Assurance of consistency among elements above: - Via analysis, measurement, prototyping, simulation, etc. - Business case analysis for requirements, feasible architectures.	• Assurance of consistency among elements above. • All major risks resolved or covered by risk management plan.

* WWWWWHH: Why, What, When, Who, Where, How, How Much

use of MBASE elements within their project development efforts. For example, the U.S. Army Tank and Automotive Command has used Easy WinWin and other MBASE elements to reconcile its software technology organizations' process and product strategies.

Conclusions and Future Directions

The ability to balance discipline and flexibility is critical to developing highly dependable software-intensive systems in a rapidly changing environment. The MBASE integration framework, process framework, and associated guidelines provide a set of risk-driven techniques that elaborate on the spiral model and the Rational Unified Process enabling an organization to achieve an appropriate balance of discipline and flexibility for each of its projects.

However, this requires a large number of guidelines to keep all of a complex software system's process, product, property, and success models well integrated across all of the phases and activities in the software-system life cycle. In some ways, we have been able to reduce this complexity. One way is by providing tools and templates for MBASE artifacts via the MBASE Electronic Process Guide [13].

Another way is to develop special domain-specific models such as for the digital library domain that enables student teams to learn the development principles and successfully develop moderate-sized Web-based applications in 24 weeks [14].

Third is to develop specialized models for particular situations, such as the SAIV process model we will discuss in an upcoming CROSSTALK article.

A future challenge is to extend the project-oriented MBASE approach to address organization-level software and system process issues. In January CROSSTALK, we will present an integration of MBASE with the University of Maryland's Experience Factory approach [15], and show how it can help organizations transition to the CMMI.

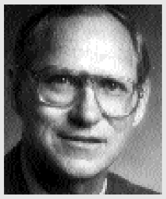
Acknowledgements

We would like to acknowledge the support of the Defense Advanced Research Projects Agency and the National Science Foundation in establishing and refining MBASE, the DoD Software Intensive Systems Directorate in supporting its application to DoD projects and organizations, and the affiliates of the USC Center for Software Engineering for their contributions to MBASE.◆

References

1. Boehm, B., and W. Hansen, eds. "The Spiral Model as a Tool for Evolutionary Acquisition." CROSSTALK May 2001, pp. 4-9.
2. Garlan, D., R. Allen, and J. Ockerbloom, eds. "Architectural Mismatch: Why Reuse Is So Hard." IEEE Software November 1995, pp. 17-26.
3. Boehm, B., et al. "Ada and Beyond: Software Policies for the DoD." National Academy Press, 1997.
4. Boehm, B., and D. Port, eds. "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them." ACM Software Engineering Notes January 1999, pp. 36-48.
5. Boehm B., D. Port, and M. AlSaid, eds. "Avoiding the Software Model Clash Spider Web." IEEE Software November 2000, pp. 120-122.
6. Boehm, B., P. Gruenbacher, and R. Briggs, eds. "Developing Groupware for Requirements Negotiation: Lessons Learned." IEEE Software May/June 2001, pp. 46-55.
7. Marenzano, J., "System Architecture Validation Review Findings," in D. Garlan (ed.). ICSE-17 Architecture Workshop Proceedings. CMU, Pittsburgh, PA, 1995.
8. Report of the Defense Science Board Task Force on Defense Software. Defense Science Board. OUSD (A&T), November 2000.
9. Boehm, B. "Anchoring the Software Process." IEEE Software July 1996, pp. 73-82.
10. Royce, W.E. Software Project Management: A Unified Framework. Addison-Wesley, 1998.
11. Jacobson, I., G. Booch, and J. Rumbaugh, eds. The Unified Software Development Process. Addison-Wesley, 1999.
12. Kruchten, P. The Rational Unified Process (2nd ed.). Addison-Wesley, 2000.
13. MBASE Guidelines and MBASE Electronic Process Guide. USC-CSE. <http://sunset.usc.edu/research/MBASE>.
14. Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, eds. "Using the WinWin Spiral Model: A Case Study." Computer July 1998, M. 33-44
15. Basili, V., G. Caldeira, and H. Rombach, eds. "The Experience Factory," in J. Marciniak (ed.). Encyclopedia of Software Engineering. Wiley, 1994.

About the Authors



Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, the Defense Advanced Research Process Agency, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Dr. Boehm originated the spiral model, the Constructive Cost Model (COCOMO), and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Phone: (213) 740-8163
Fax: (213) 740-4927
E-mail: boehm@sunset.usc.edu

Daniel Port, Ph.D., is a research assistant professor of Computer Science and an associate of the Center for Software Engineering at the University of Southern California. He received a doctorate degree from the Massachusetts Institute of Technology, and a bachelor's degree from the University of California, Los Angeles. His previous positions were assistant professor of Computer Science at Columbia University, director of Technology at the USC Annenburg Center EC2 Technology Incubator, co-founder of Tech Tactics, Inc., and a project lead and technology trainer for NeXT Computers, Inc.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Phone: (213) 740-7275
Fax: (213) 740-4927
E-mail: dport@sunset.usc.edu

CROSSTALK
The Journal of Defense Software Engineering

ARTICLE INDEX VOLUME 14

MONTHLY COLUMNS:

ISSUE		COLUMN TITLE	AUTHOR
Issue 1: January	Publisher's	Virtual Reality Can be a Great Value or Merely a Vice	R. Alder
Modeling and Simulation	BackTalk:	Politics in the Red Zone	G. Petersen
Issue 2: February	Publisher's	Software Measurement is Inherent to Project Success	B. Allgood
Software Measurement	BackTalk:	Feng Shui for Not-Really-Dummies, AKA Engineers	M. Welker
Issue 3: March	Publisher's	Publisher Leaves Process Improvement Legacy	E. Starrett
Process Improvement	BackTalk:	How Process People View Life	D. Cook
Issue 4: April	Publisher's	The Two Sides of the Web We Weave	T. Stauder
The Promise of Web-Based Applications	BackTalk:	A Tale of Two Developers	G. Petersen
Issue 5: May	Publisher's	Start Your Software Odyssey Here	Lt. Col. G. Palmer
2001 Software Odyssey	BackTalk:	Software Odd to See	G. Petersen
Issue 6: June	Publisher's	Times, They are a Changin' for Software Development	K. Richins
Software Development Methodologies	BackTalk:	Have We Lost Our Focus?	D. Cook
Issue 7: July	Publisher's	The Unsung Heroes of Testing and CM	E. Starrett
Testing and CM	BackTalk:	The Weakest Geek	G. Petersen
Issue 8: August	Publisher's	Software: An International Tie to be Grateful For	T. Stauder
Software Around the World	BackTalk:	Who's Driving the Software World Anyway?	T. Henderson
Issue 9: September	Publisher's	Software Community: Ready for the Challenges of Avionics Upgrade?	Lt. Col. G. Palmer
Avionics Modernization	BackTalk:	Aircraft, Software, COTS, GOTS, and a 1967 Chevy BelAir	D. Cook
Issue 10: October	Publisher's	How Wide Open Should 'Open Systems' Be?	Lt. Col. G. Palmer
Open and Common Software Systems	BackTalk:	GCSS on LAN	G. Petersen
Issue 11: November	Publisher's	Which Distributed Development Concept Did You Have in Mind?	E. Starrett
Distributed Software Development	BackTalk:	There's No Shame In Saying, 'I Don't Know!'	D. Cook
Issue 12: December	Publisher's	Stronger Than Ever	T. Stauder
Software Legacy Systems	BackTalk:	The Ronco Pocket Engineer	G. Petersen

ARTICLES:

THEME	ARTICLE TITLE	AUTHOR(S)	ISSUE	PAGE
ACQUISITION	A Spiral Model as a Tool for Evolutionary Acquisition	Dr. B. Boehm, W. J. Hansen	5	4
AVIONICS MODERNIZATION	The Air Force Develops an Initiative to Manage Change in Avionics Systems	P. Bowers	9	4
	Avionics Modernization and the C-130J Software Factory	R. Conn, S. Traub, S. Chung	9	19
	The Challenges of Software Certification	G. Romanski	9	15
	Customizing the Software Process to Support Avionics Systems Enhancements	Dr. P. Donzelli, R. Marozza	9	10
	Integrated Road Maps Route the Migration to Avionics Open Systems	P. Bowers	9	8
CAREERS	A Smart Way to Begin a Civilian Engineering Career in the U.S. Air Force	T. Stauder	7	19
	The Software Engineer: Skills for Change	Dr. S. Cross, Dr. C. Graettinger	6	22
CONFIGURATION MANAGEMENT	Software Configuration Management: A Discipline with Added Value	T. Butler, F. Turner, E. Sullivan, V. Standley	7	4
CMM® and CMMI℠	CMM Level 4 Quantitative Analysis and Defect Prevention	A. Florence		20
	Dispelling the Process Myth: Having a Process Doesn't Mean Sacrificing Agility or Creativity	H. Glazer	11	27
	Lessons Learned in Attempting to Achieve Software CMM Level 4	A. Florence	8	29
	Process Capability Data for the Asking	Lt. Col. R. Lang	7	21
DEVELOPMENT METHODOLOGIES	Ada in the 21st Century	B. M. Brosgol	3	20
	Extending UML to Enable the Definition and Design of Real-Time Embedded Systems	A. Moore	6	4
	Extreme Methodologies for an Extreme World	T. Leishman	6	15
	Getting Software Engineering Into Our Guts	L. Bernstein, D. Klappholz	7	25
	Mission-Based Incremental Development of C2 Systems for More Efficient Business Support	I. Ögren	10	26
	Teaching Intelligent Agents: Software Design Methodology	LTC M. Bowman, Dr. A. Lopez Jr., Maj. J. Donlon, Dr. G. Tecuci	6	10
	The Quality of Requirements in Extreme Programming	R. Duncan	6	19
DISTRIBUTED DEVELOPMENT	A Foundation for Coalition Interoperability Using NATO's C3 Technical Architecture	Dr. F. I. Moxely, L. Simon, E. J. Wells	8	4
	Distributed Development: Insights, Challenges, and Solutions	P. E. McMahon	11	4

THEME	ARTICLE TITLE	AUTHOR(S)	ISSUE	PAGE
DISTRIBUTED SOFTWARE	Factors to Consider When Selecting CORBA Implementations	Dr. T. J. Croak	11	17
	Toward Adaptive and Reflective Middleware for Network-Centric Combat Systems	Dr. D. Schmidt, Dr. R. Schantz, M. Masters, Dr. J. Cross, D. Sharp, L. DiPalma	11	10
INFORMATION ASSURANCE	Intrusion Detection: Implementation and Operational Issues	J. McHugh, A. Christie, J. Allen	1	27
	Cyber Warfare: A New Doctrine and Taxonomy	Lt. Col. L. D. Alford Jr.	4	27
LEGACY SYSTEMS	Automated Transformation of Legacy Systems	P. Newcomb, R. Doblar	12	18
	A COTS-Based Replacement Strategy for Aging Avionics Computers	D. G. Haldeman, W. J. Cannon, J. A. Luke	12	14
	The IULS Approach to Software Wrapper Technology for Upgrading Legacy Systems	D. Corman	12	9
	Reengineering: An Affordable Approach for Embedded Software Upgrade	K. Littlejohn, M. V. DelPrincipe, J. D. Preston, Dr. B. A. Calloni	12	4
LESSONS LEARNED	Lessons Learned From Using COTS Software on Space Systems	R. J. Adams, S. Eslinger	6	25
	Managing the Invisible Aspects of High-Performance Teams	RADM P. Moneymaker, Dr. L. Carter	5	29
	Predicting Staff Sizes to Maintain Networks	Dr. L. D. Gowen	11	22
	Streamlining Brings Oracle Big Savings, Better Service	S. R. Perkins	5	19
MEASUREMENT	Applying Function Point Analysis to Requirements Completeness	C. Dekkers, M. Aguiar	2	8
	The Best Measurement Tool Is Your Telephone	D. S. Lucero, F. Hall	3	15
	Evolving Function Points	L. Fischman	2	24
	Measure Size, Complexity of Algorithms Using Function Points	N. Redgate, C. B. Tichenor	2	12
	The Nine-Step Metrics Program	T. K. Perkins	2	16
	Practical Software Measurement, Performance-Based Earned Value	P. Solomon	9	25
	Software Estimation Model Calibration	Dr. R. Jensen	7	13
	The Software Maintainability Index Revisited	K. D. Welker	8	18
	Software Maintainability Metrics Model: An Improvement in the Coleman-Oman Model	A. Liso	8	15
	Software Measurement Programs and Industry Leadership	C. Jones	2	4
MODELING AND SIMULATION	Computers and M&S – Modeling and Simulation 101	Dr. D. A. Cook	1	8
	Evans and Sutherland: A Model in the Simulation Industry	M. Welker	1	4
	Integrating Process Simulation and Reliability Models	I. Rus, J. Collofello	1	15
	Leading, Integrating, and Leveraging M&S for the Warfighter	Col. W. Forrest Crain	1	11
OPEN AND COMMON SYSTEMS	The DII COE: An Enterprise Framework	Dr. G. Frazier	10	15
	The DII COE: Basic Principles and Future Challenges	D. Gardner	10	10
	DII COE for Real Time: Becoming Reality	Lt. Col. L. Robillard, Dr. H. Callison, M. Goo, J. Maurer	10	19
	Joint Technical Architecture: Impact on Department of Defense Programs	J. Kerner	10	4
PROCESS IMPROVEMENT/ BEST PRACTICES	Balancing Discipline and Flexibilty With the Spiral Model and MBASE	Dr. B. Boehm, Dr. D. Port	12	23
	Can Australia Improve Its Software Process?	A. James	8	12
	Event-Driven Learning as a Process Improvement Strategy	M. J. Hillelsohn	1	22
	The Next Refinement to Software	K. Thurston	5	16
	Perspectives on the Software Engineering Process Group	P. Kimmerly, Maj. J. Ogren, C.	3	4
	Proposal on Library-Centered Software Process Assessment	T. Komiyama, T. Sunazuka, S. Koyama	8	22
	Raytheon Stands Firm on Benefits of Process Improvement	P. Bowers	3	9
	SPMN Director Identifies 16 Critical Software Practices	M. W. Evans	3	27
	The State of Software Development in India	C. Embar	8	9
	Verification and Validation Implementation at NASA	Dr. L. H. Rosenberg	5	12
PROJECT MANAGEMENT	Does Your Internal Management Meet Expectations?	E. Schragenheim, H. W. Dettmer	4	19
	Project Scheduling According to Dr. Goldratt	T. K. Perkins	1	19
TESTING	How Much Code Inspection is Enough	R. T. McCann	7	9
	Maintaining the Quality of Black-Box Testing	P. J. Schroeder, Dr. B. Korel	5	24
	The Problem with Testing	N. Hines	7	27
WEB-BASED APPLICATIONS	The Ideal Collaborative Environment	P.A. Dargan	4	11
	The Potential of Extensible Markup Language	D. A. Hayes	4	16
	The Vulnerabilities of Developing on the Net	R. A. Martin	4	5

The Ronco Pocket Engineer

What constitutes an engineer? What makes one tick? I'm sure there are a myriad of answers but to me there are four basic ingredients to the recipe. Take three parts scientist, add two parts tinker, fold in two parts artist, and sprinkle with four parts guru.

Engineers have the scientific drive to search, explore, and discover how things work. A philomath by nature, engineers continue to learn inside and outside their field. Check out their cubicles overflowing with books, manuals, magazines, and journals.

Unlike their scientific colleagues, who loose themselves in a boundless search for knowledge, engineers are pulled back to reality by a chronic urge to tinker. While most kids are playing with their toys on Christmas day, engineers tear their toys apart to see the motor, controller, or mechanics. Once satisfied, they use the parts to create new toys that resemble those found by Woody and Buzz in Sid's room – who, by the way, is destined to be an engineer. This is the trait that leads an engineer's parents, spouse, or manager to suspect that some part of the engineer's brain is not fully developed.

Like artists, engineers create but their creativity takes a different form. Artists start with a blank canvas and end up with something that is observed and valued by its appeal. Engineers use existing materials to form a bricolage that is used and valued by its utility. The Internet is a prime example, i.e., lashing together the complexities of assorted computers, long-standing phone lines, modern fiber optics, ubiquitous cable, and strings of code to create a new form of communication. NASA engineers were at the zenith of their mission making an air filter out of spare parts for Apollo 13.

The leitmotif of an engineer is the self-confidence of always being right. No matter what the circumstance, engineers feel they know the answer. An engineer considers himself the guru of gadgets, expert of electronics, maharishi of mathematics, professor of programming, and sage of systems. At a company party, I had a graphologist analyze the handwriting of my team. One character trait that appeared in every engineer was the need to be right. The graphologist asked for the group leader. I stepped forward. She said, "I'm so sorry."

While an engineer's self confidence facilitates the drive to explore, tinker, and create, it becomes an Achilles heel for implementation. Implementation requires support and funding that comes from management or venture capitalists. Few engineers are blessed with the talent to influence, persuade, and convince a manager or venture capitalist.

Engineers present somniferous ideas thinking they have the only answer, unaware that a manager is bombarded with ideas daily and struggles to sort out the lot. Engineering managers relate with Charles de Gaulle, who said after a few years as France's president, "It is impossible to govern a country that produces 457 different kinds of cheese." Well President de Gaulle, try governing a software development team that has 457 different requirements, 45.7 methodologies, 4.57 languages, and 457,000 process improvement ideas.

How can you get a dilatory manager to cogitate your ideas and become more credulous to your point of view? How can you bridge the fissure between engineering insight and management vision?

When the clock strikes midnight, get off the chat line, turn on the tube, and turn to a cable channel. There he is – the prince of pitch, the sultan of sales, the prime minister of peddling, the royal highness of hawking – Ron Popeil. Mr. Ronco is himself, the most successful infomercial pitchman in history.

What, you don't know Ron? Give me a break. I'm sure if we cleaned out your closets, cabinets, or garage we would find a Veg-O-Matic, Mr. Microphone, Pocket Fisherman, food dehydrator, or a can of GLH Formula #9 spray-on hair with his name on it.

If not one of Ron's products, surely we would find a Chia Pet, Thighmaster, or a set of Ginsu Knives. Perhaps we would discover a Flowbee – the ingenious hair cutting system that hooks to a vacuum. Is there a Hairdini hidden in your armoire that will twist your hair into a quick bun in seconds?

You can learn from these ridiculous yet amusing infomercials. I know – you are a professional and would never be so bumptious. Well listen Mr. Professional: Ron's pulled in more than $1 billion in retail; what was your latest bonus? Reason and logic have not convinced your pervicacious boss, so maybe you need a little spice in the recipe. Don't let the potboiler products repulse you, look at how they are offered.

First, accentuate the need. Does anyone really need a Ratoto? Not until you see that baby skin a potato in seconds. While you may see the obvious need for your brilliant ideas, many mangers do not. It's your responsibility to help your manager appreciate the need for your brainchild.

Second, your idea has to be demonstrable. My appeal to the Veg-O-Matic was weak at best until I actually witnessed it slicing, dicing, and churning out julienne fries in just minutes! Although your concepts may be more complex than the Inside-the-Shell Egg Scrambler, you need to demonstrate how your idea will work. It's your responsibility to help your manager visualize the use of your idée fixe.

Third is repetition. How many times have you seen the Ginsu Knife demonstration? How about Ron's new product the Showtime Rotisserie? The more you are exposed to upbeat demos your subconscious is inculcated into submission. Don't let your ideas ride on one presentation. Indoctrinate your manager with quotidian concepts of your proposal. If your suggestion has a theme song your manager should be humming it.

Finally, curtail the risk. I forked out $19.95 for Eagle Eye Sunglasses that illuminate fish in 20 feet of water, not because I knew they would work, but because I knew if they did not work, I was only out 20 bucks. It's like riding a bike with training wheels, swimming with water wings, or working with a net. If the worst thing that can happen is palatable, then why not take a gamble? When engineers present their panaceas, managers see slipped deadlines, blown budgets, and lost careers. Package your concept with a set of training wheels. If your manger feels he or she can get on your conceptual bike without serious injury, he or she will be riding in no time.

I can already see you wheedling away. "Boss, you get Object-Oriented (OO) design and the Unified Process (UP) for just $19.95K. But wait! Fund the project this month and we'll throw a free eXtensible Markup Language (XML) workshop and starter kit. You get OO, UP, and XML all for just $19.95K. But wait! Fund the project this week and we'll throw in a cup of Java. You get OO, UP, XML, and Java all for $19.95K, or four monthly payments of $5K each."

Maybe you should start the presentation with the Clapper.

– Gary Petersen, Shim Enterprise, Inc.

PROJECT MANAGEMENT

depend on THE STRENGTHS of your TEAM

STSC

Each stage of your software product life cycle needs to be managed appropriately to meet schedules, maintain costs, and produce quality. Your project management team must be prepared and disciplined. A team without a good plan suffers failures.

If you are struggling to bring your people together to meet customer needs and get a product to the marketplace, where do you turn? Where do you start?

Begin by calling us. The *Software Technology Support Center* can teach you how to effectively manage all phases of your project, from the beginning to the end: identify the weak links, determine the real problems, and tailor solutions to your organizational needs.

We provide training in the methodologies that will make your team self-sufficient.

We have the experience to support all your project management needs. We can help you to identify your problems and develop the remedies. We can teach you to be a disciplined and consistently successful project manager. Call us first – whether your organization is big or small, just beginning a project or embattled in difficulties. We can steer you in the right direction.

OO-ALC/TISE • 7278 4th Street • Hill AFB, UT 84056 • 801 775 5555 • FAX 801 777 8069 • www.stsc.hill.af.mil

MISSION CRITICAL SOFTWARE CRSIP

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CROSSTALK / TISE
7278 4th Street
Bldg. 100
Hill AFB, UT 84056-5205

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

RETURN SERVICE REQUESTED