Requirement Modeling for the C-5 Modernization Program©

Steven D. Allen, Mark B. Hall, Verlin Kelly, and Mark D. Mansfield
Lockheed Martin Aeronautics Company

Dr. Mark R. Blackburn
Systems and Software Consortium

This article outlines the approach and benefits of a requirement-based modeling effort for the Lockheed Martin Aeronautics Company C-5 Modernization (C-5M) Program to upgrade the aircraft's engines and improve the overall reliability of the aircraft. Requirement-based modeling resulted in more consistent, complete, and precise requirements and interface information to support the design and implementation process. Systems engineers used simulations to validate requirement models and detected a large number of requirement defects that were corrected well before software implementation.

The Lockheed Martin Aeronautics Company C-5M Program is involved in the upgrade of aircraft engines and associated software systems to improve the overall aircraft reliability. Several C-5M project personnel had successfully used the software cost reduction (SCR) requirement modeling method to develop requirements for the C-130J Avionics System [1]. Requirement-based modeling develops precise behavioral requirements and formalizes interface information earlier in the development life cycle to support the design and implementation process. The C-5M Program implemented several related strategies for improving the specification, design, and implementation of the avionics software. These strategies supported on-schedule releases of major functional blocks with a significant reduction in post-release problem reporting and correction. This article focuses on the requirement specification process improvements realized through the use of requirement modeling and early requirement validation.

A *requirement model* is a formal specification of the required functional behavior of a component specified in terms of the interfaces to the component. The modeling process, supported by automated analysis provided in the modeling tool, helps detect requirement and interface problems. In addition, system engineers use a requirement simulator to validate modeled requirements prior to transfer to the designers and implementers. A *requirement simulator* is a tool that loads requirement models and supports scenario execution against the functional behavior captured in the model through a graphical user interface (GUI). A *scenario* is a sequence of input events that result in a corresponding sequence of internal state and output changes. A scenario can represent high-level system use cases or low-level component interactions. Unexpected state or outputs observed during the scenario simulation are often results of requirement defects.

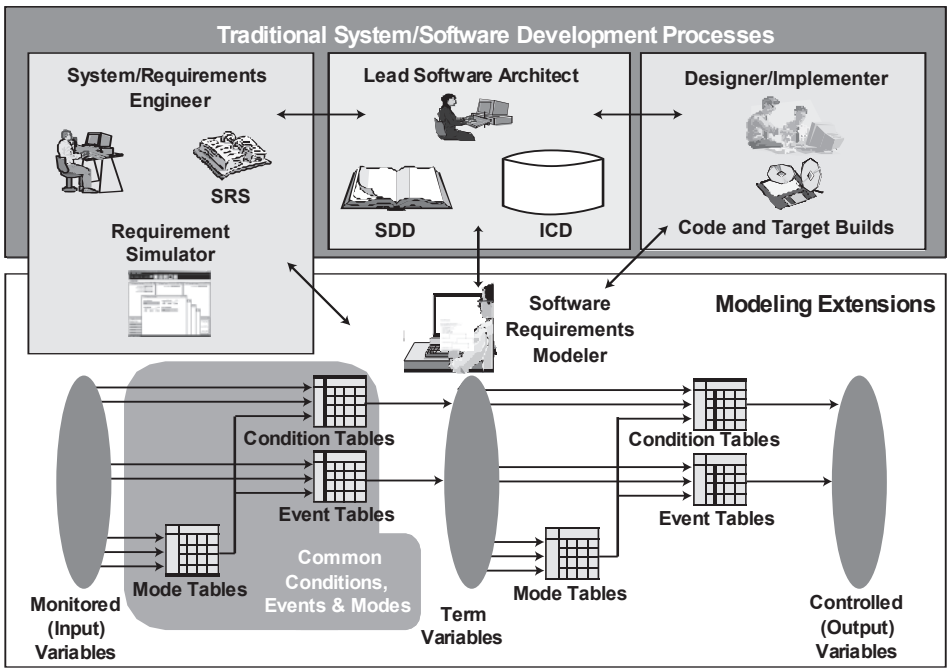
The modeling process and requirement simulation exposed a large number of requirement defects. Fortunately, these defects were identified early in the project and corrected before the software implementation process. Integration problem reports (IPRs) provided a key measure of the defects against requirements, design, and implementation. The number of IPRs was reduced significantly when compared to a prior and similar project, the C-5 Avionics Modernization Program (AMP). This article provides IPR measurement and tracking data that substantiates the claimed process improvements and program benefits. This data supports the conclusion that the C-5 Program process—when compared to the C-5 AMP—detected defects earlier, had about half of the total number of defects, and (on average) corrected the defects twice as fast.

The C-5M Program will continue new development releases for the next several years. The C-5 AMP, now in sustainment, plans to apply the improved processes successfully applied by the C-5M Program. Investigations by the C-5 AMP suggest that large or complex upgrades can be developed and maintained more cost-effectively using the improved processes described in this article.

Process Overview

The C-5M project operates as a physically co-located integrated product team. Figure 1 provides a conceptual overview of the roles and flow of the artifacts that ultimately result in the target software; it also represents both the traditional process steps and roles (top of the figure), and modeling extensions (bottom of the figure) used to develop the C-5M software. The system engineer develops textual requirements as well as any other type of analytical model that is captured in a software requirement specification (SRS). The SRS requirements are developed

Figure 1: *Process Roles and Flow*



using a requirement specification language (RSL) that has a structured syntax and restricted set of verbs (e.g., acquire, validate, provide, and derive). The RSL was developed to complement the SCR modeling method. In parallel, there is a continuous requirement flow-down process. The lead software architect identifies the components of the software architecture and works with the software requirements modelers to formalize the requirements and associated interfaces into models.

The software requirements modeler develops requirement models from the SRS and interface control document (ICD) using a modeling tool that supports the SCR method. Models capture behavioral requirement and interface information (e.g., inputs, outputs, types, and ranges) extracted from an ICD. The modeling process often identifies requirement or interface problems that must be resolved through interaction between the system engineer or software architect. For example, interface specifications were captured in a database that is shared by the project team, including subcontractors, but they were not always complete or consistent during the early part of the program (e.g., the first 100 days). The requirement modeling process and associated tools force the interface information to be complete and consistent. Additional problems or anomalies are identified by the system engineers through requirement simulation of the models. Validated requirement models are linked to the software design document (SDD). The designers and implementers work directly from the SDD, requirement models, and interfaces to implement the code. These modeling-related extensions to the process help to improve the overall performance of the team. Better requirements and interface documentation allow software designers to focus on the detailed design and implementation of the code rather than chasing requirement issues or making assumptions that can result in costly rework.

Interface-Driven Requirement Modeling

SCR is a table-based modeling method that has been effective and easy to learn for most engineers [2]. The SCR modeling language has a well-defined syntax and semantics allowing for a precise and tool-analyzable specification of the required behavior. Models represent the required functionality of a component using tables to relate monitored variables (inputs) to controlled variables (outputs), as reflected in Figure 1. There are three basic types of tables: 1) mode transition tables, 2) event tables, and 3) condition tables. A *mode transition table* is a state machine, where related system states are called system modes, and the transitions of the state machine are characterized by events. An *event* occurs when any system entity changes value. A *condition* is a predicate characterizing a system state. A *term* is any function defined in terms of input variables, modes, or other terms. The SCR tables can be combined to specify complex relationships between monitored and controlled variables using mode or terms variables. This allows common conditions, events, and modes to be defined once and referenced multiple times.

"Better requirements and interface documentation allow software designers to focus on the detailed design and implementation of the code rather than chasing requirement issues or making assumptions that can result in costly rework."

Model developers should employ a goal-oriented approach and work backwards to specify functions and constraints for each output (controlled variable or term) of the component, using the following general guidelines:

- Create a table that assigns each computed value for the table output. The value can be specified using a simple assignment or complex function. This corresponds with the RSL action verb *provide*.
- Use a condition table to describe relationships between an output (or term) if the relationships are continuous over time. Identify all conditions that must be TRUE for each output assignment. Conditions relate to the RSL verb *validate*, because they are constraints on the inputs.
- Use an event table to describe relationships between an output (or term) if the relationships are defined at a specific point in time. Define the events and optional guard conditions that trigger the event for each output assignment.
- Use a mode transition table to describe relationships between an object if the relationship for a mode is defined for a specific interval of time (set of related system states). Identify the set of modes and define the event associated with each source-to-destination transition of the model transition table.
- If there are common conditions that are related to constraints (i.e., conditions or events) of functions of two or more outputs (or terms), then define a term table that can be referenced by other tables. Term reference can be performed in a table assignment, condition, or event. Term variables correspond to the RSL verb *derive*, as terms are intermediate variables that can be referenced by other tables.

See [3] for more details on the interface-driven modeling approach.

Requirement Validation

The modeling process forces the customer requirements to be translated into a language understood by the engineers. This formalization ensures common understanding between the system requirements engineer, lead software architect, and requirements modelers across the system and software interface boundary. Despite this, the system engineers responsible for developing textual requirements were initially reluctant to use the requirement simulator that was specifically designed to support the SCR method. However, after developing a few scenarios, they realized that their understanding of the requirements could be incomplete or incorrect, often due to the complexity of the system. This revised perspective was necessary since they had the domain knowledge to judge the correctness of the modeled requirements, making their input into the validation process critical.

The typical process for validating a scenario requires the system engineer to enter input values associated with a sequence of events using the GUI simulator. After each input, the system engineering observes and validates system outputs and internal states (e.g., terms). Model issues are exposed when sequences of events do not result in outputs that the system engineer expects for a particular scenario. The most common types of

problems identified through requirement validation result from inconsistencies in related model condition, mode, or event tables. Similar inconsistencies exist within textual requirements specifications, but the inconsistencies are often difficult to identify through manual inspection and reviews as the related requirement can often be separated in an SRS document by tens or hundreds of pages. Requirements models formally link the related requirements (as reflected by Figure 1), and tools can help detect inconsistencies or issues (e.g., logical contradictions, potential divide-by-zero situation) through simulation and automated analysis.

Measurement Data

The C-5M avionics software was developed in incremental blocks with each release of a software block occurring on-schedule and with full functionality. These fully functional, on-time software releases provide evidence justifying the improved process. However, IPR-related measurement data tracked on both the C-5M Program and the C-5 AMP provides an objective way to compare the program processes. Ideally, a program should have objectives such as to: 1) minimize the total number of IPRs, 2) detect IPR-related defects early, and 3) reduce the time to correct any IPR-related defect. The following IPR measurement data provides comparative data that relates to the stated objectives. Four base measures and one derived measure are used to substantiate the process improvements of the C-5M Program:

- Base measure: Number of days since first program IPR (i.e., referred to as program start).
- Base measure: Number of IPRs.
- Base measure: Date the IPR opened.
- Base measure: Date the IPR closed.
- Derived measure: Average days to approval = Date the IPR closed - Date the IPR opened.

The C-5 measurement analyst noted that a C-5M IPR was often more detailed and specific than a C-5 AMP IPR, due to the more detailed and precise requirements defined for C-5M using the improved processes. For example, a C-5M IPR might state very specific details such as:

```
the color of the pressure
readout should be red, not
yellow, when the pressure
exceeds the pressure
threshold limit.
```

For the C-5 AMP, a similar IPR might state:

```
the warning messages have
invalid color settings.
```

As a result, the IPRs on C-5 AMP often had a broader scope relative to C-5M IPRs and required more time to investigate and implement the necessary corrective action, sometimes to the point where a single C-5 AMP IPR was equivalent in scope to two or more C-5M IPRs.

The next section provides measurement data that supports this claim. For example, the number of days to approve an IPR for the C-5 AMP was at some times almost three times longer than the number of days for an IPR-approval for the C-5M Program. Note that the numbers of Total IPRs and Days To Approve IPRs are not included in the measurement data in order to protect program-privileged information.

"The modeling process forces the customer requirements to be translated into a language understood by the engineers. This formalization ensures common understanding ... across the system and software interface boundary."

Measurement Analysis

Figures 2, 3, and 4 (on the following page) compare C-5M data against C-5 AMP measurement data in 100-day increments from the start of each respective program[1]. Figure 2 shows the total number of accumulated IPRs. The number of IPRs for the C-5M Program was slightly higher than the C-5 AMP through the first 400 days of the program. The number of C-5 AMP IPRs increases significantly at about the 500th day of the program; by the 800th day, the number of C-5 AMP IPRs is about double the number of C-5M IPRs. Figure 2 shows a linear prediction that the expected number of IPRs for the C-5M Program over the next 200 days will total about a third of the number of IPRs for the C-5 AMP; this prediction is substantiated by the measurement data shown in Figure 3.

Figure 3 shows the number of IPRs added during each 100-day period (e.g., IPRs from 300th day to the 400th day). This view of the data shows that more IPRs were detected early in the C-5M Program. This suggests that requirement modeling and validation helped to detect defects early. Figure 3 also shows that starting at day 500, the IPRs for the C-5 AMP increased significantly. The number of C-5 AMP IPRs was nearly double the number of IPRs for the C-5M Program at the same point in time. The C-5M IPRs discovery rate continues to remain significantly lower than the C-5 AMP defects from day 500 through day 800.

Figure 4 provides data on the number of days required to approve a system change that corrects an IPR-related defect. The IPR approval process for the C-5M Program was longer at the beginning. The process defined in this article was not completely refined during C-5M's first 100 days, and additional improvements were being put into place during the early days of program execution. However, the data indicates that the average approval period continued to decline through day 800 of the program. The combined data indicates that at 800 days, the number of detected IPRs by the C-5M Program is less but the time to correct the defect is significantly shorter than that for the C-5 AMP. The combined data supports the conclusion that the requirement modeling and validation provides a significant improvement in early defect identification, faster defect removal, and correction.

Leveraging Models

Approximately 90 percent of the detailed software design descriptions rely on the requirement models. The requirement model is linked to the SDD rather than having the design specified in-text. Models represent both high-level and low-level requirements (i.e., derived requirements). Unusual or complex designs are documented in the SDD using text, flow diagrams, or other engineering drawings (as needed). This is another process efficiency gained through leveraging the requirement modeling process. The model provided a formal, precise statement of the requirements that could be referenced directly in the SDD.

Common modeling patterns—such as data retrieval, data validation, and filtering—were identified and evolved for the different software components. The system

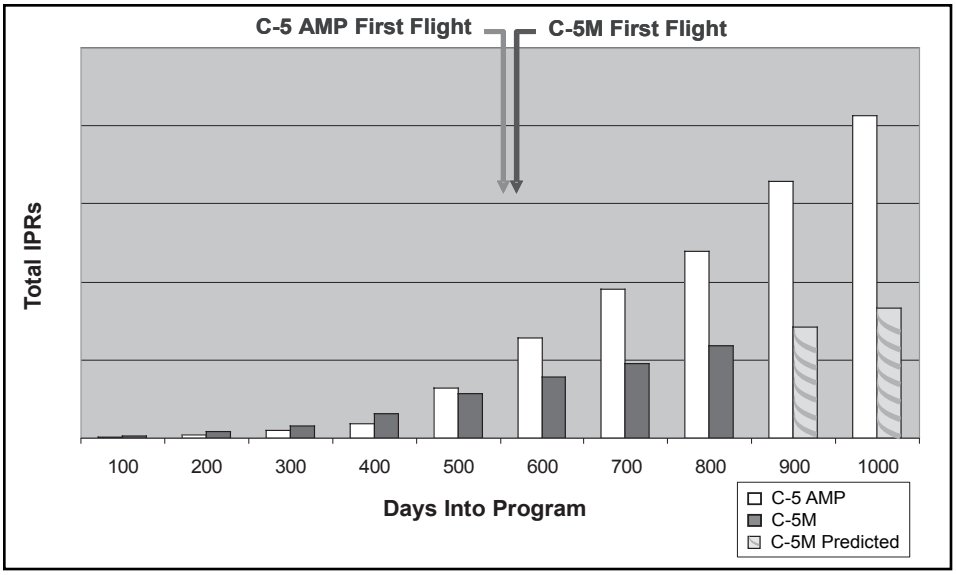


Figure 2: *IPRs vs. Days Into Program*

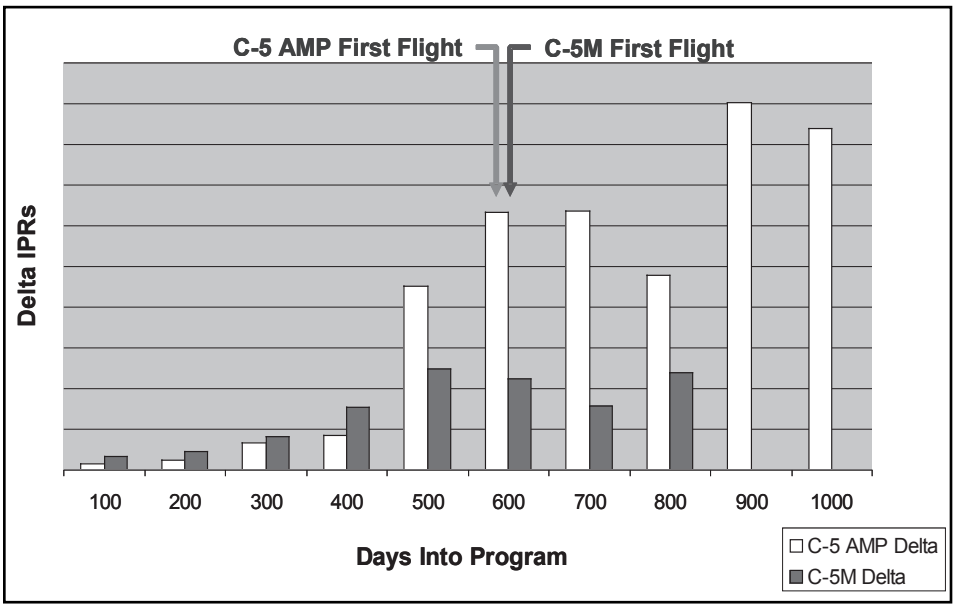


Figure 3: *Delta IPRs vs. Days Into Program*

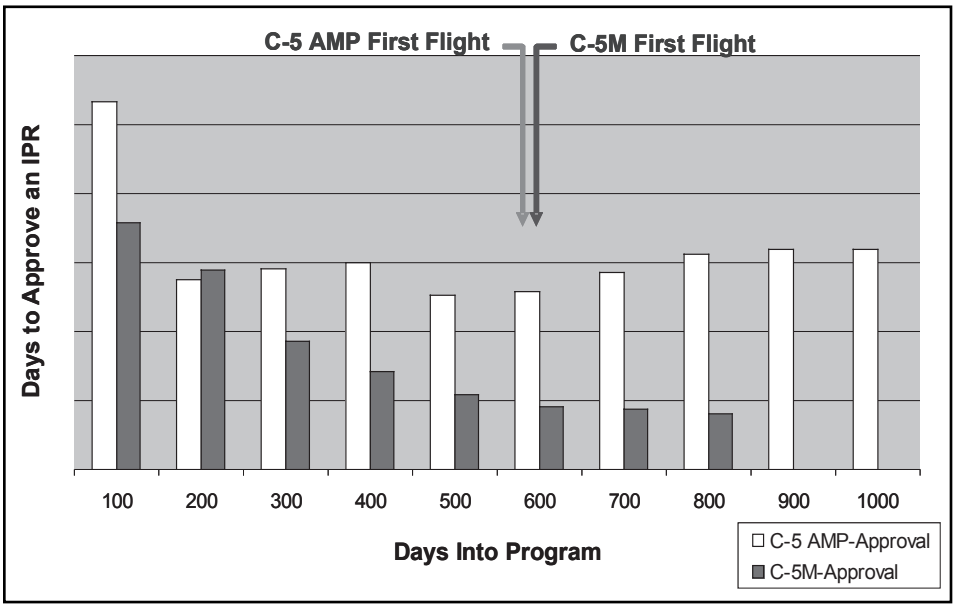


Figure 4: *Average Days to IPR Approval*

engineer and lead software modeler validated the patterns that were captured as model templates. The novice team members began development of requirement models using model templates. The model templates helped to promote additional consistency into the modeling process.

Summary

This article describes the approach and benefits derived through the use of requirement modeling for the C-5M Program. Requirement modeling helped to develop better requirements and interface information to support the design and implementation process. The systems engineers used requirement simulations of the models to validate the correctness and consistency of the requirements. The requirement modeling and simulation processes uncovered a large number of requirement defects prior to software implementation. In addition, measurement data substantiates the claimed process improvements and program benefits. Measurement data supports the conclusion that the C-5M Program process detected defects earlier, had about one-half of the total number of defects, and on average corrected the defects twice as fast as the C-5 AMP. Also described were the substantial benefits seen by the Lockheed Martin Aeronautics Company and its customer from early validation of the systems and software requirements. The significant benefits realized on the C-5M Program have resulted in plans to incorporate this requirement modeling process into the C-5 AMP sustainment efforts for large and complex software-system upgrades.◆

References

1. Faulk, Stuart, et al. Experience Applying the CoRE Method to the Lockheed C130J. Proc. of the Ninth Annual Conference on Computer Assurance, IEEE 94CH34157. Gaithersburg, MD, June 1994: 3-8.
2. Kelly, V., et al. Requirements Testability and Test Automation. Proc. of the Lockheed Martin Joint Symposium. June 2001.
3. Blackburn, Mark R., Robert D. Busser, and Aaron M. Nauman. "Interface-Driven, Model-Based Test Automation." CrossTalk May 2003 <www.stsc.hill.af.mil/crosstalk/2003/05/blackburn.html>.

Note

1. At the request of Lockheed Martin management, the horizontal line values for Figures 2, 3, and 4 have been removed.

About the Authors

Steven D. Allen is a staff engineer with the Lockheed Martin Aeronautics Company. He has more than 24 years of experience in the analysis, design, and development of software for integrated software subsystems. In his role over the last few years as a requirements engineer for the mission processing subsystem of the C-5M Program, Allen has been active in the process, procedures, and use of current software tools to clearly define and validate the software system requirements through the use of requirement modeling, simulation, and verification techniques.

Mark B. Hall is a senior staff engineer with the Lockheed Martin Aeronautics Company. He has more than 20 years of experience in the analysis, design, and development of software for integrated avionics. As the software architect for the mission processing subsystem of the C-5M Program, Hall has been active in process and procedures to clearly define the software requirements and validate them through the use of modeling and simulation. He has a bachelor's degree electrical engineering from Southern Polytechnic State University and an MBA from Kennesaw State University.

Mark D. Mansfield is a staff engineer with the Lockheed Martin Aeronautics Company. He has more than 10 years of experience in the analysis, design, and development of software for integrated avionics. In his role over the last few years as the systems engineer for the mission processing subsystem of the C-5M Program, Mansfield has been active in the process and procedures to clearly define the system and software requirements and to validate them through the use of modeling and simulation. He has a bachelor's degree in aeronautics from Embry-Riddle Aeronautical University.

Verlin Kelly is a staff specialist for the Lockheed Martin Aeronautics Company and has 41 years of experience in embedded systems and software and automated support systems. He has developed software engineering training curriculum and courses as well as co-authored presentations to the Systems and Software Technology Conference on "Quantitatively Managing Multi-Company Software Teams" and "Requirement Testability and Test Automation." Kelly has a master's degree in operational mathematics from the University of Texas at Arlington (UTA) and a bachelor's degree in physics and mathematics from Baylor University. He is involved in system/software Unified Modeling Language development and automated testing and has served on the industry advisory council for the computer science and engineering departments at the UTA and Texas Christian University.

Mark R. Blackburn, Ph.D., is a Systems and Software Consortium fellow and co-inventor of the T-VEC (also known as test-vector) system. He has more than 20 years of software systems engineering experience, spending most of his time helping companies adopt model-driven engineering tools and methods. Blackburn is a frequent speaker at conferences and symposia, and has authored more than 70 papers covering a broad spectrum of topics such as modeling, verification, software safety, security, reliability, and measurement. He has a bachelor's degree in mathematics from Arizona State University, a master's degree in mathematics from Florida Atlantic University, and a doctorate in information technology from George Mason University.

Systems and Software Consortium
2214 Rock Hill RD
Herndon, VA 20170
Phone: (703) 742-7136
Fax: (703) 742-7350
E-mail: blackburn@software.org

Coming Events

March 31-April 2
Spring 2009 Software Test & Performance
San Mateo, CA
www.stpcon.com

April 1-4
2nd International Conference on Software Testing, Verification, and Validation
Denver, CO
http://bitterroot.vancouver.wsu.edu/icst2009

April 6-8
DTIC 2009
Alexandria, VA
www.dtic.mil/dtic/announcements/conference.html

April 6-9
2009 NanoTechnology for Defense Conference
Burlingame, CA
www.usasymposium.com/nano/default.htm

April 20-23
21st Annual Systems and Software Technology Conference
Salt Lake City, UT
www.sstc-online.org

April 20-24
DISA Customer Partnership Conference
Anaheim, CA
www.afcea.org/events/disa/landing.asp

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**