Architecting for Sustainable Software Delivery

Ronald J. Koontz, Boeing
Robert L. Nord, SEI

Abstract. With increasing emphasis on avionics system rapid development and reduced cycle times, software architecting practices can be applied with agility to enhance evolving stakeholder concerns while sustaining long-term business goals.

Software intensive systems, and in particular military avionics platforms, are facing both shrinking defense budgets and the continued expectation for more advanced mission capabilities. The business case is that it is much more affordable to extend existing platform capabilities than to consider new platform designs. Over a product lifecycle, business goals and objectives continue to evolve as capabilities are realized. Paramount to enhancing current platform capabilities is an extensible and sound avionics system and mission-computing software architecture.

This article describes the role that five architecture practices are continuing to play in enabling the Apache Block III program to achieve long-term business goals. Agility is applied according to Jim Highsmith's definition, which describes it in terms of balancing flexibility and stability [1]. Such agility enables architectural development to follow a "just-in-time" model that complements iterative and incremental enhancement development and integration [2]. Delivery of capabilities is not delayed pending the completion of exhaustive requirements and design activities and reviews. At the same time, architectural agility maintains a steady and consistent focus on continual architectural evolution in support of emerging capabilities.

The Networked Common Operating Real-Time Environment (NCORE) software architecture for the Apache Longbow helicopter Mission Processor provides a case study to illustrate the architecting practices that support agility and sustainment of long-term business goals. The NCORE architecture was initially developed and flight-tested during the jointly funded Army Aviation Technology Directorate (AATD) and Boeing Manned/Unmanned Common Architecture Program, Phase II (MCAP II). The MCAP II risk reduction program initially focused on the evaluation of emerging software technologies such as real-time Java.

Since then, the NCORE architecture continues to evolve on the Apache Longbow Block III Program. The driving architectural requirements include safety, performance, availability/reliability, modifiability, and interoperability. As initial platform capabilities are realized and as avionics computing lifecycles shorten, increased emphasis is placed on extensibility and the desire to host applications developed by third parties. In parallel, embedded infrastructure software must be architected to reduce overall time and cost to incorporate hardware upgrades (proactive obsolescence management) and to enable the hosting of new or existing application-level software.

In the section that follows, each practice is described and illustrated with an NCORE architecture example. Next, incremental delivery of new capabilities is described in terms of how it is realized by combining all of the practices. Finally, the essential characteristics of the practices are summarized according to agile development and architecture criteria. This summary provides a checklist to aid learning for others developing software-reliant systems, and provides feedback on whether their application is on track to help meet project goals.

Architecture Practices That Balance Flexibility and Stability

Architecture best practices are a set of actions, methods, techniques, and/or strategies applied to software architecting and the software lifecycle that are well proven and known to yield desired outcomes without introducing unnecessary program risk. Those architecting practices that are leveraged by the NCORE architecture and that can be broadly applied to avionics platforms are now described and analyzed from the point of view of agility.

Incremental/Iterative Development

NCORE architecture and application software artifacts are developed using an incremental and iterative development lifecycle [3], notionally shown on a fiscal year calendar in Figure 1. Based on periodic customer statements of work, incremental capabilities are planned, developed, tested, and fielded. For each statement of work iteration, integrated build and release plans are developed. To enhance testability and integrability, software builds contain two or more mini-builds that accelerate design, development, automated testing, and platform integration.

This incremental/iterative development approach parallels agile software development, with cross-functional/agile teams of requirements/integrators, coders, and testers working according to agreed-upon release planning (the build plan). Incremental/iterative

development further enables user feedback and refinement, fixes, and overall product enhancements to be folded back into product deliverables as the software matures.

Iterative loops within each build cycle map to mini-builds, and the quantity of mini-builds is adjusted to accommodate the functional complexity of the build. For example, a typical six-month build cycle may be decomposed into two or more mini-builds where multiple parallel teams execute independent work threads. Each team defines incremental build content that can most effectively produce overall build content objectives at the time of final build release. Incremental build content definition minimally considers work thread complexity and resource availability relative to overall build objectives.

Informed Technology-Insertion Decision Making

Informed technology-insertion decision making is built upon communication and knowledge sharing and is characterized as a cyclic and iterative process of understanding stakeholders' concerns, making and documenting decisions, and evaluating the consequences. This communication provides near real-time, two-way dialog between architects and stakeholders. Both push and pull communication strategies are concurrently employed:

Push: architecture and software design documentation. Information about the architecture is periodically provided to stakeholders.

Pull: architecture evaluations. Periodic architecture evaluations collaboratively pull information from the business management, architecture team, and stakeholder community.

Together, these activities contribute to enhanced knowledge sharing across the integrated team.

The NCORE architecture description is centered on module, Component-and-Connector (C&C), and allocation views [3]. Several styles employ separation of concerns to capture architecturally significant artifacts. To maximize resources and avoid duplication, overlapping stakeholder concerns are combined by the architects into a concise set of architectural views. As the architecture evolves, the architects carefully analyze current concerns and anticipate future stakeholder needs to determine whether new views are required or whether existing views can be enhanced.

Table 1 shows the stakeholder-to-documentation-artifact-type matrix developed by

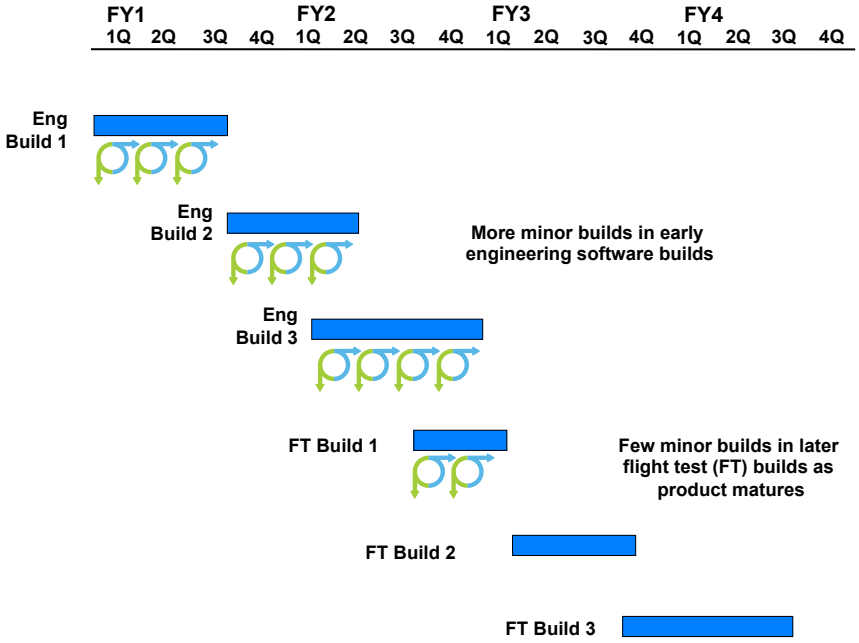


Figure 1: Notional NCORE Incremental/Iterative Development Lifecycle

	Module Views				C&C Views				Allocation Views			Other
Stakeholder	Decomposition	Generalization	Uses	Layered	Publish-Subscribe	Shared-Data	Client-Server / Peer-to-Peer	Communicating-Processes	Deployment	Implementation	Work Assignment	SOW & Performance Specifications
Current and future architects	d	d	d	d	d	d	d	d	d	s	o	d
Government project management	d	o	s	o	s	s	o	o	s	o		d
Contractor's project management	s	o	s	s	s			o	o	s	d	d
Member of development team	d	d	d	d	d	d	d	d	s	s	d	s
Testers and integrators	s	s	d	s	s	s	s	s	s	d	s	s
Third-party developers	d	s	s	s	d	o	s	s	o	o		o
Maintainer	d	s	d	s	s	s	s	s	s	s		
Analyst for performance	d	s	d	s	s	s	s	d	d	d		o
Analyst for safety	d		d	s	s	d	s	d	d	d		d
New stakeholder	x	x	x	x	x	x	x	x	x	x	x	s
Key: d = detailed information, s = some details, o = overview information, x = anything												

Table 1: NCORE Stakeholders and Architecture Information That They Find Useful

the architecture team when initially developing NCORE architectural views. According to the Table 1 key, an individual stakeholder level of concern is identified as either "detailed," "some details," "overview," or "anything." The "anything" concern level signifies access to all readily available artifacts which can be browsed by any new stakeholder seeking rapid and broad architecture understanding. Software architects and development team members seek "detailed" Module and C&C view content, which convey static and runtime concerns, respectively. Overall, decomposition and layered module views are most popular across the stakeholder community based on the multi-faceted and layered NCORE architecture and the overlapping concerns they address.

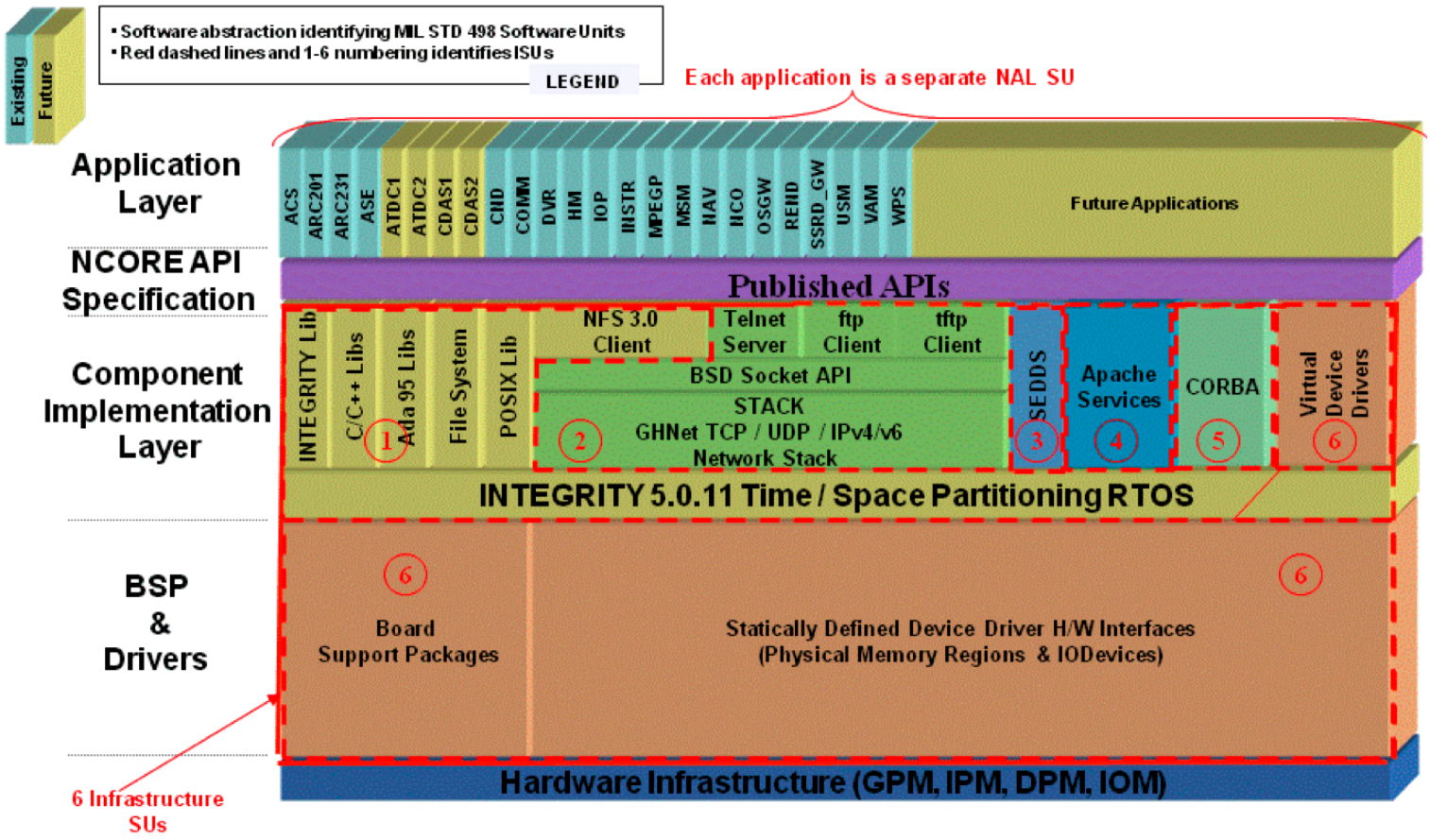


Figure 2: NCORE Software Unit Identification View

Frequent architecture information exchange among stakeholders fosters creativity and identifies continual opportunities for further exploitation of NCORE architecture capabilities at acceptable program risk levels.

Frequent Architecture Analysis and Improvement

Customer collaboration over contract negotiation is a well-known agile development practice. Agile design involves alternative analysis and trade-offs among evolving stakeholder concerns and among the significant design decisions made to address them. Periodic architecture evaluations enable and compliment continuous stakeholder education. Stakeholder knowledge sharing enables product improvement through exploitation of diverse viewpoints. Investing in frequent improvement, recognized as an "inspect and adapt" best practice, is about enhancing product capability based on direct stakeholder input and stated business goals.

A team of experts, composed of members of the Carnegie Mellon University SEI and the U.S. Army, evaluated the Apache Block III NCORE software architecture by applying the Architecture Tradeoff Analysis Method® (ATAM®) [4]. The evaluation team pulled information from the business management, architecture team, and stakeholders in a goal-directed fashion. The evaluation team analyzed the architecture with respect to the business drivers and stakeholders' concerns about quality attributes [5]. Discovered risks served as feedback and guidance to the architects and business managers, and these risks became the focus of follow-on mitigation and improvement activities. This continual cycle influences and sustains evolution of the architecture and business drivers in a predictable manner. In addition to identifying risks, other important benefits to performing evaluations include clarification of stakeholder concerns about quality attributes and enhanced communication among the stakeholders [6].

Reports from the field validate and affirm the value of conducting periodic peer reviews during the design phase of the software development lifecycle [7, 8]. Apache quarterly software design reviews led by software architects and subject matter experts further enhance customer communications and stakeholder engagement. Customer comments from these reviews are iteratively incorporated in the design artifacts to continually improve product quality.

Strategies and Patterns for Sustained Evolution

Strategies for architectural governance and patterns for open systems and extensibility sustain long-term product evolution. Architecture evolution is enhanced through process-driven oversight centered on balancing flexibility and stability.

An Architecture Review Board (ARB) is tasked with governing the architecture to ensure that it evolves in a disciplined way. The ARB acts as an internal design review team of culturally diverse architects (differing viewpoints) and subject matter experts (specific domain knowledge) who are chartered to ensure minimally

complex, consistent, and informed designs—all aimed at minimizing implementation time and cost and reducing the amount of rework.

The NCORE architecture is documented as a series of views [3] as required by evolving stakeholder needs. The software unit identification view is shown in Figure 2, and additional architecture documentation can be found in the work of Koontz [9, 10, 11].

Designing for extensibility promotes continued evolution and uses design patterns such as real-time technical metric reporting, publish-subscribe, client-server, and layering. Designing as an open system through nonproprietary application programming interfaces enables third-party software integration and the ability to move applications between CPUs during integration (to achieve load balancing, for example).

Prototyping and the Research & Development (R&D) Test Platform

The Apache Block III program continues to benefit from using and leveraging multiple prototyping activities. The MCAP II program, jointly funded by the AATD and Boeing, serves as an R&D flight test platform for evaluating emerging technologies targeted for production Apache. Targeted early prototyping significantly reduces program risk through technology culling and selective maturation. Examples of network-centric experimentation now transitioning into Apache Block III include tactical communication data links for H.264 video streaming, soldier radio waveform, wideband networking waveform, Link-16, and manned/unmanned teaming. Agility in the form of cross-functional teams and R&D-focused culture is paramount to the success of proof-of-concept technology demonstrations that further enable rapid system integration with acceptable program risk [12].

After emerging technologies are initially demonstrated and selected for product integration (new technology insertions), they are typically rapidly prototyped within the production environment. Prototyping at this later point in the lifecycle enables parallel requirements definition and software development, a recognized and proven agile practice. Agile application shortens overall integration lifecycles by merging requirements definition with software development, test, and integration process steps.

Applying Architecture Practices to Support Sustainable Delivery

Now that the five practices have been individually explained, they are applied in combination to demonstrate how they support the evolving system and delivery of new capabilities.

Evolving stakeholder needs and business goals identified through architecture evaluation lead to new requirements for selected capabilities. Initially, NCORE started with a primary focus on open-systems architecture, performance, and reliability and is now moving toward flight safety and extensibility (realizing the planned technology refresh must satisfy very long-term program objectives). These new requirements trigger the infrastructure and application-insertion timelines in response.

For example, during an ATAM evaluation, an exploratory scenario identified height-above-ellipsoid as a common platform technique for improving weapons-delivery accuracy that could be easily implemented. This kind of change is supported and sustained by the architecture practices working interactively and in unison in accordance with Table 2. Alternatively, first-time Joint Tactical Radio (JTR) Link-16 integration is viewed as a much more complex insertion; however, application of the practices is equally relevant.

The incremental/iterative development shown in Figure 1, now being deployed for JTR Link-16 integration, allows for focused and time-phased requirements/architecture analysis, code, test, and integration activities for complex and less defined function deployment based on stakeholder priorities and choices. Time-phased incremental/iterative development enhances testability due to incremental design verification and just-in-time architectural decision making that must be coordinated and scheduled.

Informed technology-insertion decision making applies to both JTR Link-16 and height-above-ellipsoid. It enables communication and understanding among multiple stakeholders regarding a specific concern and its business priority. From Table 1, the statement of work and performance specifications represent formal and binding contractual agreements that convey customer requirements to the architects and program management. These documents are pivotal for Apache because they represent coupling between prior architecture evaluation and contractual requirements.

Frequent architecture analysis and improvement is centered on brainstorming exploratory scenarios using agreed-upon architecture artifacts. The architectural views, shown in Table 1, provide evidence that quality attributes are being satisfied during consideration of height-above-ellipsoid specific concerns relative to business goals and priorities.

Strategies and patterns for sustained evolution are employed and enforced by the ARB to ensure architectural integrity across the lifecycle. Patterns for open systems and extensibility provide support for making and localizing architecture change. Insertion agility is the result of identifying required architecture changes and employing an agile just-in-time methodology (e.g., adding secure socket library in support of Link-16 integration).

The NCORE architecture was first-flight proven in 2004, using the MCAP II Prototyping and R&D Test Platform. Initial NCORE demonstration validated the architecture capability to rapidly integrate new technologies and is the keystone open systems architecture enabler for the Apache Block III program. Additionally, network-centric experimentation has led to the customer's decision to choose Apache Block III as the first JTR Link-16 integration platform.

Based upon discussions with the stakeholders, height-above-ellipsoid is being provided in a future enhancement statement of work and will soon be implemented and deployed. JTR Link-16 software development and integration continues to mature and evolve with the delivery of incremental capabilities.

Agile Development and Software Architecture Enablers

Table 2 characterizes the five architecture practices using established criteria from agile software development and software architecture fundamentals, including response to change, customer collaboration, quality attributes, and architecture, so they can be applied to benefit the development of other software-

	Response to Change	Customer Collaboration	Quality Attributes	Architecture
Incremental / Iterative Development	Necessary processes are identified to respond to change	Functional requirements are communicated with focused criteria and business priority	Quality attribute requirements are defined and tied to business goals	Timeline of critical architectural decisions is clear and scheduled
Informed Technology-Insertion Decision Making	Dynamic environment and changing requirements are understood	Effective customer communication channels manage expectations	The importance of quality attribute requirements is understood	Architectural issues (e.g., technical debt) are tracked and managed
Frequent Architecture Analysis and Improvement	Waste is identified and trade-offs are managed (e.g., technical debt and defects)	Artifacts to keep multiple stakeholders informed are agreed upon and produced effectively	Quality attribute requirement analysis is in place and used to predict system properties	Evidence is provided that the architecture satisfies quality attribute requirements
Strategies and Patterns for Sustained Evolution	Impact of uncertainty on the project is acknowledged	Technology insertions are driven and targeted by the user	Quality attribute design is aligned to lifecycle maintenance	Just-in-time architecting enables technology-insertion agility
Prototyping and R&D Test Platform	High-potential technologies are identified to respond to change	Pipeline of emerging technologies and technology insertions are mapped to evolving business goals	Measurement environment is in place to monitor the implemented system quality and done criteria	Obsolescence risk management occurs via prototyping of newest avionics technologies (multicore processors)

Table 2: Mapping of Practices to Agile and Architecture Criteria

reliant systems across different domains. These criteria provide a quick look into the application of the practices and associated risks to enabling the ability to sustain software development and delivery at the expected velocity (pace) for large-scale, complex, multiyear projects [13].

Key Takeaways

• For software-intensive, multiyear projects, agile development, which is focused on rapid, short-term deliverables, must be complemented by sustainable architecture practices that ensure the incremental delivery of capabilities over the extended lifecycle of the product.

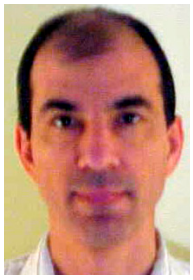
• Software architecture best practices support sustainable software delivery by leveraging established criteria from agile software development and by applying software architecture fundamentals that include response to change, customer collaboration, quality attribute trade-offs and analysis, and architecture governance. These practices are interrelated and interact to provide sustainable delivery of quality products.

• Architecting with agility can be applied across the lifecycle to continuously develop, deliver, and enhance software-reliant systems.

ABOUT THE AUTHORS

Ronald J. Koontz is a Boeing Company Technical Fellow with more than 20 years of expertise in real-time embedded systems. He is an SEI-certified ATAM evaluator and holds a Boeing Software Architect Certificate. He has served as a software architect on U.S. Army Apache Attack Helicopter Mission Processor projects since 2003. Prior to Apache programs, he led the design, implementation, and field testing of multiple software-intensive projects for The Boeing Company, Phantom Works.

Phone: 480-891-2065
E-mail: ron.j.koontz@boeing.com



Robert L. Nord is a senior member of the technical staff at the SEI and works to develop and communicate effective methods and practices for software architecture. He is coauthor of the practitioner-oriented books Applied Software Architecture and Documenting Software Architectures: Views and Beyond, published by Addison-Wesley, and lectures on architecture-centric approaches.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA
Phone: 412-268-1705
Fax: 412-268-5758
E-mail: rn@sei.cmu.edu

REFERENCES

1. Highsmith, J. A. Agile Project Management: Creating Innovative Products. 2nd ed. Boston: Addison-Wesley Professional, 2009.
2. Brown, N., R. Nord, and I. Ozkaya. "Enabling Agility Through Architecture." CrossTalk 23.6 (November/December 2010): 12-17.
3. Larman, C. and V. R. Basili. "Iterative and Incremental Development: A Brief History." IEEE Computer, 36(6), 2003: 47-56.
4. Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. Documenting Software Architectures: Views and Beyond. Boston: Addison-Wesley, 2003.
5. Clements, P., R. Kazman, and M. Klein. Evaluating Software Architectures: Methods and Case Studies. Boston: Addison-Wesley, 2002.
6. Ozkaya, I., L. Bass, R. Sangwan, and R. Nord. "Making Practical Use of Quality Attribute Information." Spec. issue of IEEE Software 25.2 (March/April 2008): 25-33.
7. Nord, R. L., J. Bergey, S. Blanchette Jr., and M. Klein. Impact of Army Architecture Evaluations (CMU/SEI-2009-SR-007). Software Engineering Institute, Carnegie Mellon University, 2009. <http://www.sei.cmu.edu/library/abstracts/reports/09sr007.cfm>
8. Edmondson, J. S., E. Lee, and C. G. Kille. "A Light-Weight Architecture Trade Off Process Based on ATAM." SEI Architecture Technology User Network (SATURN) Conference. Sheraton Station Square, Pittsburgh. 14-16 May 2007. <http://www.sei.cmu.edu/library/abstracts/presentations/ATO-Lite-for-SATURN-2007-2.cfm>
9. Forstrom, H. "Inexpensive ATAM-Peer Review Detects and Fixes Architecture Problems Early." SEI Architecture Technology User Network (SATURN) Conference. Radisson Green Tree, Pittsburgh. 30 April-1 May 2008. <http://www.sei.cmu.edu/library/abstracts/presentations/ATAM-peer-review-SATURN-2008.cfm>
10. Koontz, R. "Mission Processor Software Open Systems Architecture for the Apache Helicopter." Proceedings of the 63rd American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2007. 2253-2261.
11. Koontz, R. "Apache Mission Processor Software Architecture: Architectural Approaches." Proceedings of the 64th American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2008. 1507-1514.
12. Gannon, S.P. and R.E. Speir. "US Army Aviation Network Technologies Demonstrated at JEFX'08." Proceedings of the 65th American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2009. 812-823.
13. Koontz, R. "Apache Mission Processor Software Architecture: Architectural Decisions." Proceedings of the 65th American Helicopter Society International Annual Forum. Alexandria, VA: American Helicopter Society, 2009. 766-774.