NPS-AM-11-C8P02R02-023

EXCERPT FROM THE

PROCEEDINGS

OF THE

EIGHTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM WEDNESDAY SESSIONS VOLUME I

An Architecture-Centric Approach for Acquiring Software-Reliant Systems

Lawrence Jones and John Bergey, Software Engineering Institute

Published: 30 April 2011

Prepared for the Naval Postgraduate School, Monterey, California 93943

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

The research presented at the symposium was supported by the Acquisition Chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request Defense Acquisition Research or to become a research sponsor, please contact:

NPS Acquisition Research Program
Attn: James B. Greene, RADM, USN, (Ret.)
Acquisition Chair
Graduate School of Business and Public Policy
Naval Postgraduate School
555 Dyer Road, Room 332
Monterey, CA 93943-5103
Tel: (831) 656-2092
Fax: (831) 656-2253
E-mail: jbgreene@nps.edu

Copies of the Acquisition Sponsored Research Reports may be printed from our website www.acquisitionresearch.net

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Preface & Acknowledgements

During his internship with the Graduate School of Business & Public Policy in June 2010, U.S. Air Force Academy Cadet Chase Lane surveyed the activities of the Naval Postgraduate School's Acquisition Research Program in its first seven years. The sheer volume of research products—almost 600 published papers (e.g., technical reports, journal articles, theses)—indicates the extent to which the depth and breadth of acquisition research has increased during these years. Over 300 authors contributed to these works, which means that the pool of those who have had significant intellectual engagement with acquisition issues has increased substantially. The broad range of research topics includes acquisition reform, defense industry, fielding, contracting, interoperability, organizational behavior, risk management, cost estimating, and many others. Approaches range from conceptual and exploratory studies to develop propositions about various aspects of acquisition, to applied and statistical analyses to test specific hypotheses. Methodologies include case studies, modeling, surveys, and experiments. On the whole, such findings make us both grateful for the ARP's progress to date, and hopeful that this progress in research will lead to substantive improvements in the DoD's acquisition outcomes.

As pragmatists, we of course recognize that such change can only occur to the extent that the potential knowledge wrapped up in these products is put to use and tested to determine its value. We take seriously the pernicious effects of the so-called "theory–practice" gap, which would separate the acquisition scholar from the acquisition practitioner, and relegate the scholar's work to mere academic "shelfware." Some design features of our program that we believe help avoid these effects include the following: connecting researchers with practitioners on specific projects; requiring researchers to brief sponsors on project findings as a condition of funding award; "pushing" potentially high-impact research reports (e.g., via overnight shipping) to selected practitioners and policy-makers; and most notably, sponsoring this symposium, which we craft intentionally as an opportunity for fruitful, lasting connections between scholars and practitioners.

A former Defense Acquisition Executive, responding to a comment that academic research was not generally useful in acquisition practice, opined, "That's not their [the academics'] problem—it's ours [the practitioners']. They can only perform research; it's up to us to use it." While we certainly agree with this sentiment, we also recognize that any research, however theoretical, must point to some termination in action; academics have a responsibility to make their work intelligible to practitioners. Thus we continue to seek projects that both comport with solid standards of scholarship, and address relevant acquisition issues. These years of experience have shown us the difficulty in attempting to balance these two objectives, but we are convinced that the attempt is absolutely essential if any real improvement is to be realized.

We gratefully acknowledge the ongoing support and leadership of our sponsors, whose foresight and vision have assured the continuing success of the Acquisition Research Program:

- Office of the Under Secretary of Defense (Acquisition, Technology & Logistics)
- Program Executive Officer SHIPS
- Commander, Naval Sea Systems Command
- Army Contracting Command, U.S. Army Materiel Command
- Program Manager, Airborne, Maritime and Fixed Station Joint Tactical Radio System

- Program Executive Officer Integrated Warfare Systems
- Office of the Assistant Secretary of the Air Force (Acquisition)
- Office of the Assistant Secretary of the Army (Acquisition, Logistics, & Technology)
- Deputy Assistant Secretary of the Navy (Acquisition & Logistics Management)
- Director, Strategic Systems Programs Office
- Deputy Director, Acquisition Career Management, US Army
- Defense Business Systems Acquisition Executive, Business Transformation Agency
- Office of Procurement and Assistance Management Headquarters, Department of Energy

We also thank the Naval Postgraduate School Foundation and acknowledge its generous contributions in support of this Symposium.

James B. Greene, Jr.
Rear Admiral, U.S. Navy (Ret.)

Keith F. Snider, PhD
Associate Professor

Panel 2 – Advancing Open Architecture Acquisition

Wednesday, May 11, 2011	
11:15 a.m. – 12:45 p.m.	**Chair: Christopher Deegan,** Executive Director, Program Executive Office for Integrated Warfare Systems ***Delivering Savings with Open Architecture and Product Lines*** Brian Womble, USN, and William Schmidt, Mike Arendt, and Tim Fain, IBM ***An Architecture-Centric Approach for Acquiring Software-Reliant Systems*** Lawrence Jones and John Bergey, Software Engineering Institute ***Advances in the Acquisition of Secure Systems Based on Open Architectures*** Walt Scacchi and Thomas Alspaugh, Institute for Software Research

Christopher Deegan—Executive Director, Program Executive Officer, Integrated Warfare Systems (PEO IWS). Mr. Deegan directs the development, acquisition, and fleet support of 150 combat weapon system programs managed by 350 military and civilian personnel with annual appropriations of over $5 billion.

Mr. Deegan holds a Bachelor of Science degree in Industrial Engineering from Penn State University, University Park, Pennsylvania and a Master of Science degree in Engineering from The Catholic University of America, Washington, DC. He is a graduate of the Program Managers Course, Defense Systems Management College, Fort Belvoir, VA. He is a Certified Acquisition Professional and is Level III certified in three DA WIA career fields: Program Management; Research and Systems Engineering; and Business, Cost Estimating and Financial Management.

Mr. Deegan is the only Comptroller employee to be recognized by the Association of Scientists and Engineers as "NAVSEA Engineer of the Year" (1993). He received the Assistant Secretary of the Navy (Research, Development and Acquisition) and NAVSEA Acquisition Excellence Awards (1996), the David Packard Award for Governmental Excellence (1996), the Navy's Meritorious Civilian Service Award (1997), the Navy's Competition and Procurement Excellence Award (2003), and a Meritorious Unit Commendation Medal as a member of the SEA WOLF Program Office (2006). Mr. Deegan was awarded the Presidential Rank of Meritorious Executive in October 2007.

An Architecture-Centric Approach for Acquiring Software-Reliant Systems

Lawrence Jones—Dr. Jones is a Senior Member of the Technical Staff within the Research, Technology, and Systems Solutions Program (RTSS) and leads the Product Line Practice Initiative at the Software Engineering Institute (SEI) of Carnegie Mellon University. Prior to joining the SEI, he served a career in the U.S. Air Force and is the former Chair of the Computer Science Department at the Air Force Academy. His PhD in computer science is from Vanderbilt University. He is a senior member of the IEEE and ACM, a Fellow of ABET and the CSAB, and a member of the ABET Board of Directors Executive Committee. [lgj@sei.cmu.edu]

John Bergey—Mr. Bergey joined the SEI in 1993 as a Visiting Scientist and became a member of the technical staff in 1995. Currently, Mr. Bergey is a member of the Research, Technology, and Systems Solutions Program (RTSS) and is active in the Architecture Centric Engineering and Product Line Practice initiatives. His role in these initiatives is to proactively assist DoD programs in applying SEI technologies (e.g., product line practices, the ATAM, and the QAW) to improve their acquisition practices and reduce software acquisition risk. Before coming to the SEI, Mr. Bergey was a Software Division Manager with the U.S. Naval Air Development Center. [jkb@sei.cmu.edu]

Abstract

Because software plays a critical role in nearly every complex Department of Defense acquisition, there is increased emphasis on reducing acquisition risk for software-reliant systems. An architecture-centric acquisition approach has proven to be effective. In this paper, we present the basics of architecture-centric engineering. Then we give a structure for incorporating these practices into the acquisition life cycle, illustrated by an example. We overview how these practices are being implemented in current programs. We conclude by demonstrating how a program can use this approach in synergy with current system engineering practices to achieve an appropriate emphasis on software in a system acquisition.

Introduction

Software plays an increasingly critical role in Department of Defense (DoD) acquisitions and is often cited as the reason for frequent cost overruns, schedule slippages, and quality problems. As a result, there is increased emphasis on finding effective ways to reduce risk when acquiring software-reliant systems. While there is no “silver bullet,” an architecture-centric acquisition approach has proven to be an effective way of reducing acquisition risk (Nord, Bergey, Blanchette, & Klein, 2009).

Informally, the software architecture is the blueprint describing the software structure of a system. Moreover, the architecture is the key enabler to achieving the all-important system quality attributes such as modifiability, response time, security, reliability, availability, interoperability, and usability (i.e., the system’s non-functional requirements).

An architecture-centric approach involves the following:

- determining the architecturally-significant quality attribute requirements and describing them in a meaningful way,
- creating the architecture and documenting it,
- evaluating the architecture for suitability in supporting the required quality attributes, and
- ensuring the system is implemented according to the architecture.

While the design of the architecture and subsequent implementation of the system are responsibilities of the supplier, the acquirer has a central role in determining the quality attribute requirements that will drive the architectural design and in evaluating the suitability of the architecture to meet mission goals of the system. The bottom line responsibility is ensuring that the delivered system conforms to a high-quality architecture that will be responsive to the system's needs, both now and in the future.

This paper provides the background and context for architecture-centric engineering practices.[1] Then we present a structure to incorporate these practices into key points in the acquisition life cycle as a means of risk reduction. Application of these practices is explored in different phases of the DoD system acquisition life cycle. We provide examples of how different elements of these practices are being implemented or will be piloted in current programs. We conclude with demonstrating how a program can use this approach in synergy with current system engineering practices to achieve an appropriate emphasis on software in a system acquisition.

Software Architecture

The software architecture is the foundation for any software-reliant system. It represents the earliest design decisions that are both the most difficult to get right and the hardest to change downstream. The software architecture will allow or preclude nearly all of the system's quality attributes. These qualities are all largely pre-cast when the software architecture has been established. No amount of later tuning will compensate for a poorly constructed software architecture. Experience has shown that an unsuitable software architecture will eventually result in some sort of disaster for a project. Disaster may mean failure to meet performance goals, failure to interoperate as needed, or inordinate sustainment costs, among other problems.

Informally, software architecture is the blueprint describing the software structure of a system. Formally, "the software architecture of a computing system is the set of structures needed to reason about the system, which comprise software components, relations among them, and properties of both" (Clements et al., 2011).

This definition carries several implications.

- Software architecture is an abstraction of a system. This allows concentration on the aspects that have system-wide import. Architectural views have proven appropriate to reason about whether a system can meet its mission goals, regardless of the system's scale.
- Systems can and do have many structures. This is analogous to the electrical, framing, and plumbing structures in a building. Views of the different structures are important to understanding the complete system.
- Every software-intensive system has an architecture, by definition. Just having an architecture is different from having an *intentional* architecture that is known to everyone. Bottom line: If you don't develop an architecture, you will get one anyway—and you might not like what you get!

[1] For some of the background information, the authors consolidated and summarized information from various reports and presentations by our colleagues at the Software Engineering Institute (SEI). Some of this wording has been used directly; some wording has been refined over time by the entire team. We wish to acknowledge the contributions of Paul Clements, Mike Gagliardi, Rick Kazman, Mark Klein, and our teammates in the Architecture Centric Engineering Initiative of the SEI.

Proper specification of required functionality is critical, to be sure. However, if functionality were all that mattered, any monolithic software structure would do. When we become concerned with evolving the system throughout its life cycle, or making sure that it responds quickly enough to the user's input, or guarding against unauthorized access of data, and so forth, we become concerned with structuring the system in a way to provide these qualities and more. That's what architecture is about.

Time and care must be invested in the software architecture because making changes later is extremely costly and often impossible. The software architecture should then guide the implementation. Throughout the development process, the software architecture must play a role that is both prescriptive and descriptive. Even in an incremental acquisition and development approach, the core system and software architectural decisions that support the important quality attribute goals for the system must come first, and then they can be enhanced in future increments or spirals. An architecture-centric approach is key to the development of systems that meet both their functional and quality goals. Figure 1 illustrates the critical role of quality attributes in an architecture-centric software development approach.

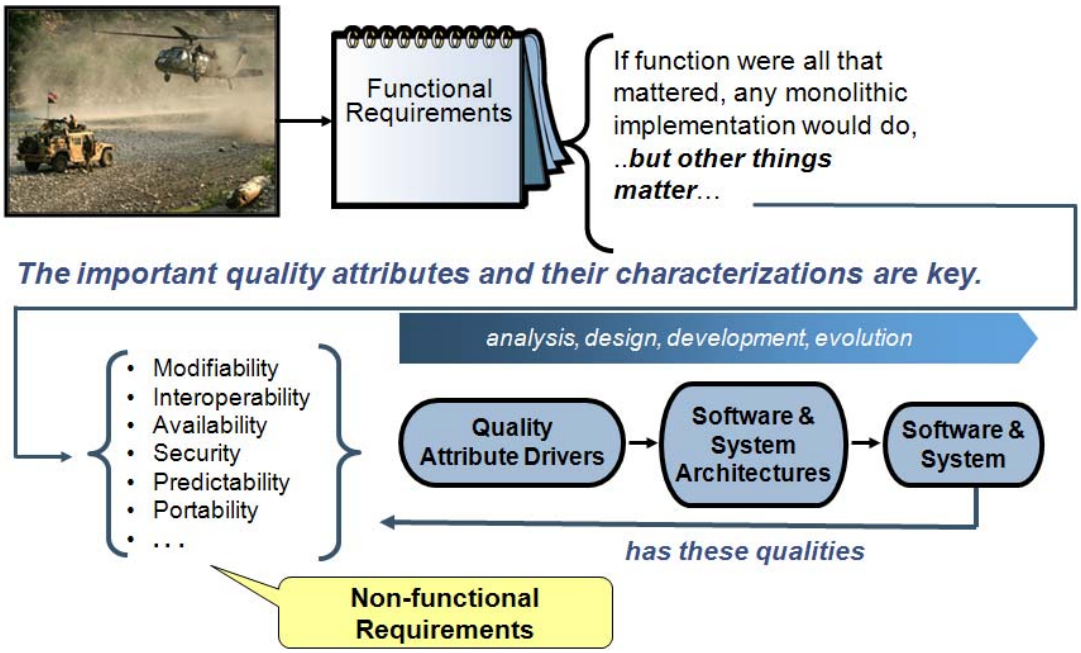

Figure 1. The Role of Quality Attributes in Architecture-Centric Software Development

Architecture-Centric Engineering

Architecture-centric engineering is the discipline of effectively using architecture to guide system development.

Architecture-centric development for a software-reliant system involves

- creating the business or mission case for the system,
- understanding the requirements,
- creating or selecting the software architecture,
- documenting and communicating the software architecture,
- analyzing or evaluating the software architecture,

- implementing the system based on the software architecture, and
- ensuring that the implementation conforms to the software architecture.

Over the past 18 years, the Carnegie Mellon® Software Engineering Institute (SEI) has developed and promulgated a series of architecture-centric methods supporting this development approach. These methods are given in **Error! Reference source not found.**. While the design of the architecture and subsequent implementation of the system are responsibilities of the supplier, the acquirer has the all-important oversight role for these activities. Additionally, the acquirer has a central role in the following:

- creating the business or mission case for the system,
- determining the quality attribute requirements that will drive the architectural design,
- evaluating the suitability of the architecture to meet mission goals of the system, and
- ensuring that sufficient architecture documentation exists to support the evaluation, implementation, and evolution of the system.

We will provide some background on methods to support these activities before describing how to apply them in an acquisition.

Table 1. Architecture-Centric Engineering Activities and Some Supporting SEI Methods

Architecture-Centric Engineering Activity *Italicized activities indicate those that are of particular importance to acquirers.*	**Supporting SEI Methods**[2] * indicates methods that have been extended to address systems engineering.
*creating the **business or mission case** for the system*	Pedigreed Attribute eLicitation Method (PALM)
*understanding the **requirements***	Quality Attribute Workshop (QAW); Mission Thread Workshop (MTW)*
creating and/or selecting the architecture	Attribute-Driven Design (ADD) and Architecture Expert tool (ArchE)
***documenting and communicating** the architecture*	Views and Beyond Approach; Architecture and Analysis Design Language (AADL)
***analyzing or evaluating** the architecture*	Architecture Tradeoff Analysis Method (ATAM)*; System of Systems Architecture Evaluation*; Cost Benefit Analysis Method (CBAM); AADL
implementing the system based on the architecture	
ensuring that the implementation **conforms** to the architecture	Architecture Reconstruction and Mining tool (ARMIN)

[2] Descriptions of these methods may be found at http://www.sei.cmu.edu/architecture/tools/.

evolving the architecture so that it **continues to meet business and mission goals**	Architecture Improvement Workshop (AIW) and ArchE
ensuring **use of effective architecture practices**	Architecture Competence Assessment

Creating the Business Case for the System

Justification for a system is part of the DoD planning, programming, and budgeting system. From this basis, it is essential to derive a succinct description of the key business and mission drivers to guide the architects in making design decisions and tradeoffs. While details of how to accomplish this are beyond the scope of this paper, identification of these drivers must come from the acquirer.

The Pedigreed Attribute eLicitation Method (PALM) is an SEI method that elicits and captures the business goals for an organization that lies behind the development of a software-intensive system (Clements & Bass, 2010). These business goals, often poorly understood and poorly articulated, serve as the foundation for many of the quality attribute and behavioral requirements for a system. Kazman and Bass (2005) categorized 190 distinct business goals from 25 architecture evaluations, including 18 government systems. Clements and Bass (2010) updated this list and classified goals into categories . They identified the following business goal categories:

- maintaining growth and continuity of the organization,
- meeting financial objectives,
- meeting personal objectives,
- meeting responsibility to employees,
- meeting responsibility to society,
- meeting responsibility to country,
- meeting responsibility to shareholders,
- managing market position,
- improving business processes, and
- managing quality and reputation of products.

Understanding the Requirements

Functional requirements specify what a system is supposed to do (e.g., retrieve database information in response to a user request). There are many techniques for specifying functional requirements, and systems analysts have been doing this for a long time using well-developed techniques. The same can't be said for specifying quality requirements. Considering the example of database information retrieval: How rapidly must information be retrieved? How secure must the transmission be? These are the quality aspects of this functional requirement. Understanding the quality attribute requirements is essential because these requirements largely drive the architecture.

Quality attributes are rarely captured effectively in requirements specifications (e.g., a system performance specification); they are often vaguely understood and weakly articulated. Just citing the desired qualities is not enough; it is meaningless to say that the system shall be "modifiable" or "interoperable" or "secure" without details about the context. Yet, these sorts of imprecise, and therefore un-testable, specifications abound.

The practice of specifying quality attribute scenarios can remove this imprecision and allows desired qualities to be specified and evaluated meaningfully. A quality attribute scenario is a short description of an interaction between a stakeholder and a system and the

response from the system. A quality attribute scenario (see the example in Figure 2) consists of six parts:

- Source of stimulus—the entity (e.g., human, computer, any actuator) that generated the interaction with the system;
- Stimulus—what triggers the interaction with the system;
- Environment—the conditions under which the stimulus occurs (e.g., under overload conditions or normal operations);
- Artifact—the part (or whole) of the system that is stimulated;
- Response—what the system does in response to the stimulus; and
- Response measure—the measurable response necessary for the requirement to be tested.

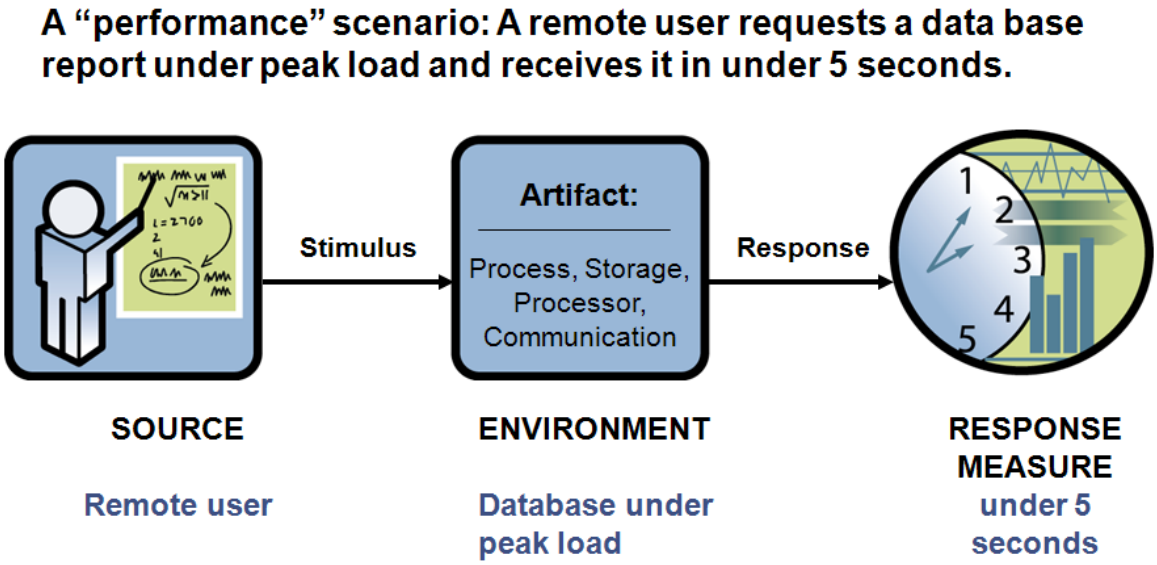

Figure 2. Example of a Quality Attribute Scenario

The Quality Attribute Workshop (QAW) is an SEI method to elicit and articulate detailed quality attribute requirements for a system (Barbacci, 2003). The QAW is a facilitated method that engages diverse stakeholders early in the life cycle to elicit, collect, and organize software quality attribute requirements in the form of scenarios *before* the architecture is created. Appendix A describes the specific steps for conducting a QAW.

The results of the QAW should be followed by an analysis and planning activity to determine further steps such as continuing the elaboration of quality attribute requirements. For example, some of these quality attributes will be high priority and will have architectural significance. These attributes should be fed into the design process along with functional requirements and any technical and business constraints.

Analyzing or Evaluating the Architecture

It is nearly always cost-effective to evaluate software quality as early as possible in the life cycle; problems that are found early are easier and cheaper to correct. Architecture evaluations are a type of quality evaluation that can be conducted early in the life cycle (indeed, throughout the life cycle) and both quantitative and qualitative benefits have been

reported (Clements, Kazman, & Klein, 2002). The net result is improved architectures providing the basis for better systems.

Architecture analysis techniques may be classified as *questioning techniques* or *measuring techniques*. Questioning techniques include scenario-based analysis, or if the domain is more mature, questionnaires or checklists. Measuring techniques rely on quantitative measures of some sort, such as coupling and cohesion of the modules, or the depth of an inheritance hierarchy. Simulations or prototypes can also be used to measure qualities. A comprehensive evaluation strategy might use both types of techniques. Relative to questioning techniques, measuring techniques have the advantage of providing quantitative information but have the drawback of requiring investment in developing some working artifact to measure. Thus, questioning techniques are more amenable to early life cycle use.

The SEI's Architecture Tradeoff Analysis Method® (ATAM®)[3] is a scenario-based questioning technique for analyzing and evaluating software architectures. It not only can be used to evaluate architectural decisions against specific quality attributes; it also allows engineering tradeoffs to be made among possibly conflicting system quality goals. In this way, the ATAM evaluation can detect areas of potential risk in meeting quality goals within the architecture of a complex software-reliant system. It has also proven useful at various points in the life cycle, including post-deployment support. Clements et al. (2002) provide details and examples of the ATAM and other software architecture evaluation methods.

The ATAM has several advantages. It can be done early, quickly, and inexpensively. The method involves project decision makers, other stakeholders (including managers, developers, maintainers, testers, re-users, end users, and customers), and a software architecture evaluation team. These groups collaborate to determine the critical quality attributes of the system and effectively evaluate the consequences of architectural decisions in light of specified quality attributes and business goals. The method helps to ensure that the right questions are asked to uncover the following:

- risks—software architecture decisions that might create future problems in some quality attribute;
- non-risks—software architecture decisions that appear to be good and should be documented;
- sensitivity points—properties of one or more components (and/or component relationships) that are critical for achieving a particular quality attribute response (That is, a slight change in a property can make a significant difference in a quality attribute.); and
- tradeoffs—decisions affecting more than one quality attribute.

This information can be aggregated into risk themes—statements of risk patterns (i.e., higher level, cross-cutting risks) that "percolate up" and resonate more with stakeholders because they point out systemic causes of risk that relate more directly to the ability of the architecture to meet mission goals.

The steps of the ATAM are provided in Appendix B.

[3] Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Documenting and Communicating the Architecture

Architecture is the conceptual glue that holds every phase of a project together for its many stakeholders. Documenting the architecture is the crowning step to crafting it. If you go to the trouble of creating a good architecture, you *must* describe it appropriately so that others can find the information they need. Architecture documentation is necessary to support the evaluation, implementation, and evolution of the system (Clements et al., 2011).

When discussing architecture representation, the related terms *structure* and *view* are used. A view is a representation of a set of system elements and the relationships among them. A structure is the set of elements itself, as they exist in software and hardware.

A discussion of the types of structures and views and their usefulness is beyond the scope of this paper. Suffice it to say that non-runtime views of software are useful for

- project planning;
- allocating work assignments;
- designing for modifiability, reusability, portability, extensibility, etc.;
- facilitating incremental development; and
- a host of other critical purposes.
- Runtime views are useful to show how software will handle
- hazards, faults, and errors;
- fault tolerance/reconfigurations;
- performance;
- data (e.g., quality, timeliness, ownership, access privileges); and
- interface boundaries.

An architecture is a multidimensional construct, too involved to be seen all at once. Recall that systems are composed of many structures. Different views allow different stakeholders to concentrate on certain aspects of a system while de-emphasizing (or ignoring for the moment) other aspects. Thus, views are a way to separate concerns and manage complexity. The choice of views to document depends on the nature of the system and the stakeholder needs. Different stakeholders will have different uses for the architecture documentation. Documenting an architecture is then a matter of documenting the relevant views, and then adding documentation that applies to more than one view (see Figure 3).

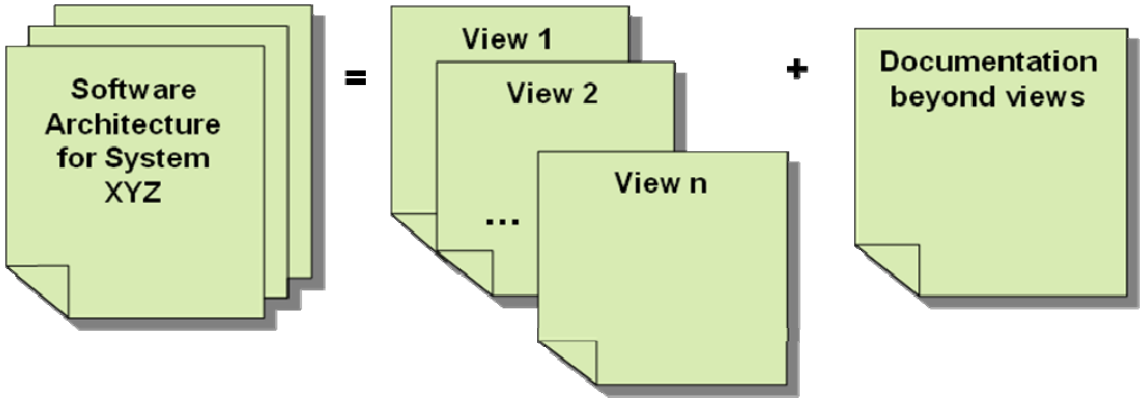

Figure 3. Views-Based Architecture Documentation

Successful Application of Architecture Practices in the DoD

So how well do these practices work in the DoD context? A workshop involving twelve Army programs that had conducted ATAM or QAW exercises was held to understand the impact the ATAM evaluations and QAWs had on system quality and the practices of the acquisition organization. These exercises were sponsored by the Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA[ALT]) as part of the Army Strategic Software Improvement Program (ASSIP) (Nord et al., 2009).

The results of the workshop showed the following:

- 6 of 12 reported cost less than or equal to traditional techniques.
- 10 of 12 reported quality of results greater than or equal to traditional techniques.
- 10 of 12 reported that the techniques helped to understand and control cost and schedule.
- 12 of 12 reported increased understanding of system's quality attribute requirements, design decisions, and risks.
- 12 of 12 reported that the methods were good mechanisms for communication among stakeholders—a majority reported *very substantial* or *significant* improvement in stakeholder communication.
- 8 of 12 reported t*hat the methods* impr*oved the ar*chitecture—of these, most reported *significant* improvement in their architecturally significant artifacts.

While these are positive results, the results were not nearly as dramatic as they could have been because the go-aheads for conducting each of these QAWs and ATAM-based architecture evaluations were made *after* the respective contracts were already awarded. We refer to this as a *reactive approach* because it involves ad hoc contractual changes and the development contractors may justifiably view such engagements as being more intrusive because they can potentially perturb cost and schedule. A *proactive approach* is preferred because it avoids these problematic aspects. In a proactive approach, architecture-centric activities are specified as an integral part of the acquisition planning effort and are subsequently incorporated in the RFP/contract from the outset. The result is that all bidders can appropriately understand, cost, and integrate the required architecture-centric activities into their development approach and subsequently include them in their technical proposals.

In addition to the quantitative results, the following quotes attest to the enthusiasm of many participants: "The ATAM architecture evaluations resulted in improved documentation, improved communication, reduced risk in schedule and cost, and a higher quality product to the warfighter"; and "The importance of having had the backing of Army senior leadership and ASSIP funding is that the beneficiaries— the Army programs—went from "Nay-Sayers" to "Yea-Sayers."

Architecture-Centric Acquisition

Architecture-centric acquisition involves using the architecture and architecture-centric practices as a *contractual* means to reduce risk and gain early confidence that the system being acquired will meet its mission goals. Moreover, it enables a program office to perform its *contract management and technical monitoring* function with greater effectiveness and results in the delivery of a more capable and higher quality product to the warfighter. The underlying premise is that architecture is of enduring importance because it is the right abstraction for planning and assigning work and for performing ongoing analyses throughout a system's lifetime. It provides an acquisition focus and technical monitoring lens

that is commensurate with an acquisition organization's responsibilities and limited resources.

Motivation for Architecture-Centric Acquisition

Software plays a critical role in most system acquisitions and is often the reason programs cite for cost overruns, schedule slippages, and quality problems. As a result, there is increased emphasis on finding effective ways for an acquisition organization to reduce risk when acquiring software-reliant systems. An architecture-centric acquisition approach has proven to be an effective way of reducing risk—especially software acquisition risk.

There are several motivations for adopting an architecture-centric acquisition approach:

- Independent program assessments have shown that many development and system performance problems are architectural in nature.
- Early identification of architectural risks saves money and time.
- Acquisition organizations need a proactive means to validate key measures of system adequacy, such as Key Performance Parameters (KPPs), Key System Attributes (KSAs), and Technology Readiness Levels (TRLs), earlier in the development cycle.
- Help is needed to alleviate a significant DoD "pain point" (i.e., not treating software engineering aspects on a par with systems engineering considerations).
- DoD programs need proven acquisition practices to acquire systems having robust architectures.
- The efficacy of the software architecture has a direct impact on the warfighter.

The benefit of using an architecture-centric acquisition approach is that it provides a program with the leverage to require a fully developed architecture that can be evaluated for risks and give early insight into necessary design tradeoffs. Such an approach gives maintainers, sustainers, testers, architects, verifiers—all stakeholders—a view that they would not otherwise have and enables them to better understand how the system is being designed. Even more importantly, they have a voice in the development of the system about their needs and the important quality attributes. It is the architecture focus, though, from end-to-end that enables programs to identify problems before they prove too costly and time-consuming to fix. By being proactive and building in a continuum of architecture support in an acquisition (e.g., specification, evaluation, improvement),acquisition professionals are able to obtain the timely information they need to carry out their acquisition responsibilities more effectively.

Key Elements of an Architecture-Centric Acquisition

Architecture-centric acquisition is all about reducing risk by ensuring that the acquisition organization and the development contractor (or contractors) carry out good architecture-centric development practices to support program goals. From the perspective of an acquisition program, the key elements of an architecture-centric acquisition include

- determining the system's architecturally significant requirements and specifying them in a meaningful way;
- commissioning the development of the architecture and ensuring it is appropriately documented;

- evaluating the architecture to determine its suitability to support the architecturally significant requirements, mission and business goals, KPPs and KSAs, and TRLs;
- leveraging other promising architecture-related practices so a program office can perform its acquisition responsibilities more effectively.

Using architecture-centric practices throughout the acquisition life cycle of a software-reliant system leads to

- early identification of important system qualities enabling a rigorous specification of a system's often-neglected non-functional requirements,
- cost savings in integration and testing,
- predictable product quality supporting the achievement of mission goals, and
- cost-effective system evolution and accommodation of future upgrades.

All these things translate to reduced acquisition risk. Moreover, an architecture-centric focus provides an acquirer with

- an effective means to perform its contract oversight and technical monitoring function;
- the right level of abstraction that aligns with limited program office resources and time and effort,
- early insight into system and software design suitability,
- a risk-based means of managing a contractual development effort, and
- a product focus that complements process-focused activities.

Of all the architecture-centric practices available, architectural evaluations serve as the lynchpin for an architecture-centric acquisition approach (Bergey, 2009).

An Example Acquisition

An architecture-centric acquisition approach involves having the government specify in the request for proposal (RFP)/contract the particular architecture-related activities or practices it wants included in the contract to reduce acquisition risk. Figure 4 depicts an example of an architecture-centric acquisition. The bottom portion of the figure shows a detailed expansion of the Engineering and Manufacturing Development phase.

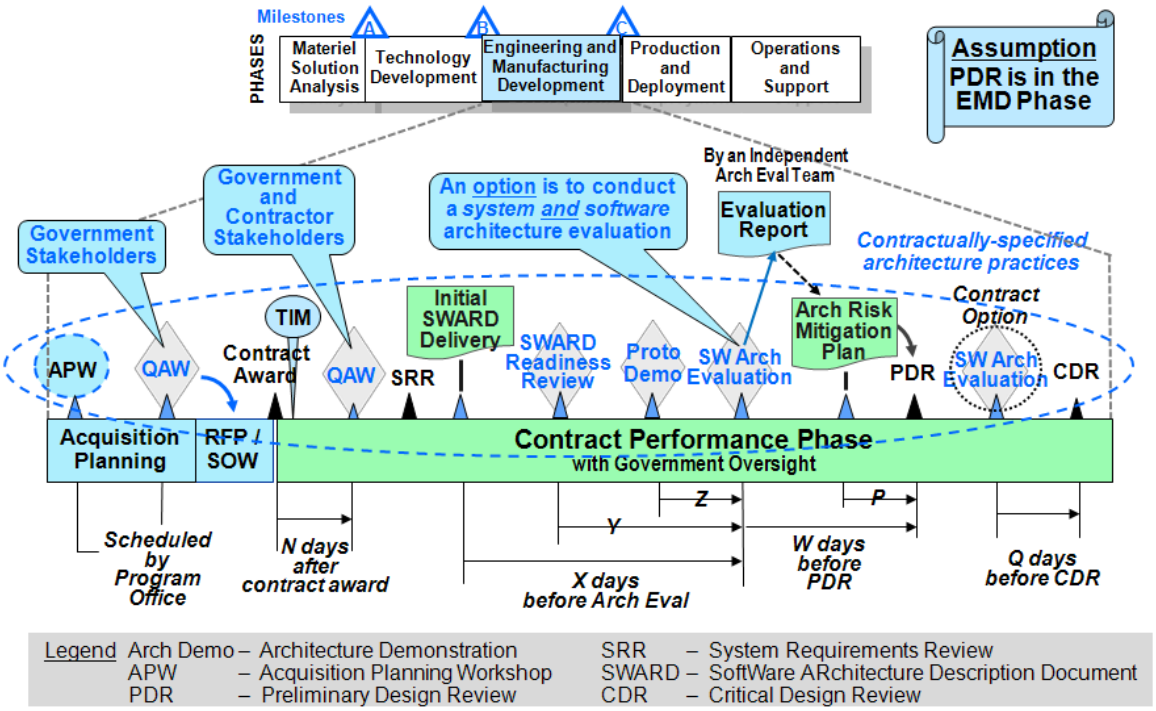

Figure 4. Example Architecture-Centric Acquisition Approach

As shown in the example, an architecture-centric approach begins in the Acquisition Planning phase before there is a contract, a developer, or an architecture. Before a system is released for bids, a program conducts a Quality Attribute Workshop (QAW) to bring together all of the government stakeholders in a facilitated session. This initial QAW involves eliciting and specifying the quality needs of the stakeholders. These needs can be prioritized and serve as a basis for determining the requirements that will drive the architectural design. Once a contract is awarded, another QAW is conducted with both government and contractor stakeholders to review, elicit, and refine the quality requirements (and derived requirements) so there is a common understanding of the system requirements.

Prior to the Preliminary Design Review (PDR), an external team (commissioned by the government) will evaluate the development contractor's software architecture using a sub-set of the quality attribute scenarios from the QAWs. A prerequisite for conducting such an evaluation is the contractual delivery of a Software Architecture Description Document (SWARD). Since the SWARD plays such a key role in an evaluation, an Architecture Documentation Readiness Review (ADRR) will be conducted to ensure that the documentation is adequate to support the architecture evaluation. In the event that the documentation is deficient in some way, there is sufficient time for the developer to correct the deficiencies.

Ideally, the architecture evaluation team should be invited to a prototype demonstration prior to the architecture evaluation. This will allow the team to gain additional knowledge of the system (beyond just the documentation). The demonstration may involve a structured walk-through, a simulation, or a working prototype on appropriate computer resources—whatever is sufficient to demonstrate the required capabilities of the software. This demonstration will be in accordance with a Prototype Demonstration Plan and Procedures (PDPP) document that is specified as a contractual deliverable.

Following the architecture evaluation, the evaluation team produces a report describing the risks (and risk themes) corresponding to the architecture's ability to achieve the desired system qualities. In response, the development contractor produces a risk mitigation plan that is a contractual deliverable. The contractor's plan for mitigating the risks is then presented and discussed during the PDR, which is held soon after the evaluation.

The whole of these activities constitutes an architecture-centric approach. Determining what should be included for a particular program is best accomplished by first conducting an acquisition planning workshop (APW) with key stakeholders, which is the first activity depicted in the example. Once the government awards the contract, a technical interchange meeting (TIM) is held with the winning contractor to discuss the architecture-centric activities so government plans and expectations are fully understood.

An acquisition program can adopt such an architecture-centric acquisition approach independent of the specific development methodology the contractor chooses to adopt (e.g., agile, model-based, spiral, or incremental). This is because the architecture-related activities are specified as event-driven so they are appropriately coordinated with (and feed key information into) the mandatory decision points that are prescribed in the DoD 5000 life cycle management system. These mandatory decision points, which are independent of the contractor's specific development approach, involve such activities as a System Requirements Review (SRR), Preliminary Design Review (PDR), Critical Design Review (CDR), and other milestone reviews.

Moreover, the RFP will require the contractor to integrate the government-specified architecture-related activities (e.g., a QAW or architecture evaluation) with its own development processes. To ensure the development contractor will appropriately integrate the processes, the RFP/contract will require the contractor to describe its approach in its technical proposal and in a set of acquisition documents that are traditional contract deliverables. These contract deliverables include such documents as a Program Management Plan (PMP), Integrated Master Schedule (IMS), Risk Management Plan (RMP), and Software Development Plan (SDP). This approach helps ensure that the development contractor does not treat the architecture-centric activities as isolated events but instead appropriately integrates them with its own development methodology. This, in turn, results in the adoption of a coherent architecture-centric development approach.

Achieving Greater Synergy Between Systems and Software Engineering

One significant pain point in the DoD acquisition environment is the lack of synergy between systems engineering and software engineering activities. Disparate teams often perform these activities without closely communicating and coordinating their efforts during the course of acquisition planning and implementation. With the increased emphasis on systems engineering within the DoD acquisition community, software engineering considerations often take a backseat to systems engineering, despite the fact that software is a critical element in almost every DoD acquisition. The result is that there are often major disconnects in specifying the desired system properties and in ensuring a congruent implementation approach from both a system and software perspective. Inroads to alleviate this situation are not likely to come about naturally, or at least they have not to date. Rather, some kind of "forcing function" is needed to help change the existing behavior pattern that is prevalent in the DoD acquisition environment. Acquisition organizations can leverage an architecture-centric acquisition approach to help bring about desired changes.

As depicted in Figure 5, a suggested starting pointing is to establish a common system context diagram for use by system and software engineers that identifies all the

system actors, the system interfaces, and system artifacts from a black-box perspective. The purpose of this is to establish common terminology, a shared view, and common understanding of the system to be acquired for use by *both* system and software engineers. In conjunction with this, the program should adopt use cases to specify the functional requirements and quality attribute scenarios to specify the non-functional requirements.

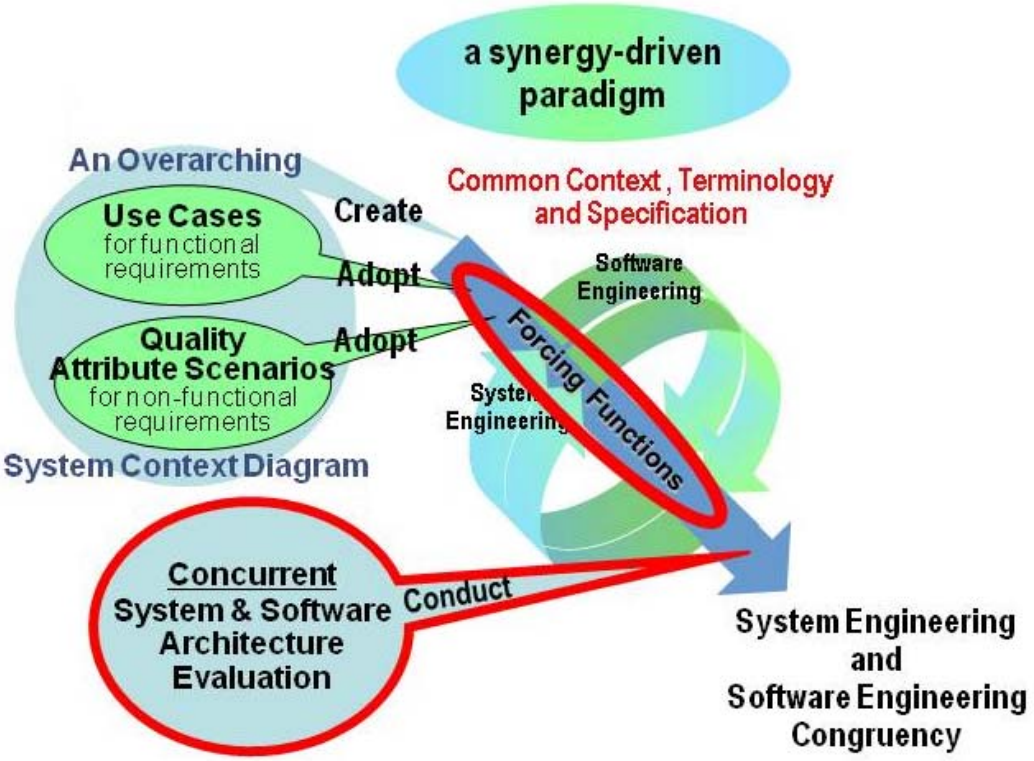

Figure 5. Promoting Synergy Between System Engineering and Software Engineering Activities

While these steps are not revolutionary, they are a significant improvement over the traditional DoD practice of specifying requirements in a stunted "shall" and "will" form. Moreover, they represent a significant step forward toward improving communication and achieving greater synergy between system and software engineering activities.

Beyond these measures, an architecture-centric acquisition approach can provide a major forcing function to promote greater synergy between system and software engineering on the contractor's side of the acquisition fence. This forcing function involves specifying a concurrent evaluation of both the system architecture and software architecture by an external evaluation team commissioned by the government program office. This evaluation has the effect of promoting greater communication and cooperation between the developer's system architect and software architect. The nature of the forcing function is that both architects will be required to participate in the architecture evaluation and will have to be prepared to walk the evaluation team through their respective architectures—back-to-back, one quality attribute scenario at a time—and describe how the respective architecture can achieve a particular quality attribute. This will serve to draw out any differing system and software engineering assumptions or discontinuities in the system and software architectural design. In addition, since the system and software architects are aware from the outset of the contract that the architecture evaluation is a future event that they will be involved in, it

has the effect of promoting earlier communication and coordination between the architects and their respective development teams. This is a "win-win" situation for both the government and the development contractor.

New Architecture-Centric Acquisition Practices

The SEI's proactive, architecture-centric approach has been tested and proved in several DoD software acquisitions[4] (SEI, 2011). To augment and improve the current approach, the SEI is currently exploring or piloting other architecture-related activities or practices. These promising practices include

- conducting an evaluation of two development contractors' architectures during a competitive down select,
- incorporating an architecture competency skills survey as part of a competitive acquisition,
- taking remedial action in the Operations and Support (O&S) phase to motivate a recalcitrant legacy system contractor to adopt good architecture practices,
- incorporating a set of architecturally significant metrics and an architecture improvement roadmap in an acquisition involving a major legacy system upgrade,
- incorporating an architecture-centric approach as part of a product line acquisition,
- incorporating "model-based development" as part of an architecture-centric acquisition, and
- an architecture-driven test approach to better focus a developer's testing efforts and provide more bang for the buck.

One of these efforts is a pilot for an Army program, another is an Independent Research and Development (IRAD) project, and the others are in different phases of exploration with prospective customers of Navy and Air Force programs. We plan to report results of these efforts at a future symposium.

Summary and Conclusions

The key things to remember about an architecture-centric acquisition approach are the following:

- The quality and longevity of a software-intensive system are largely determined by its architecture.
- Early identification of architectural risks saves money and time.
- There are proven practices to ensure that acquisition organizations acquire systems with appropriate software architectures.
- There are concrete actions acquirers can take to adopt an appropriate architecture-centric approach and apply sound architectural practices.

An architecture-centric acquisition approach enables a program office to perform its technical oversight and technical monitoring function with greater effectiveness. The practices

[4] A number of reports related to software architecture and acquisition may be found at http://www.sei.cmu.edu/architecture/start/publications/dodacquisitioncontext.cfm.

- are commensurate with a program office's responsibilities, limited resources, time available, and key contractual events;
- provide early insight into critical requirements and design decisions that drive the entire development effort;
- provide a proven and effective means for discovering software design risks;
- enable risks to be mitigated early and cost-effectively; and
- help avoid test and integration problems and costly rework downstream.

The bottom line is the delivery of a more capable and higher quality product to the warfighter.

References

Barbacci, M., et al. (2003). *Quality attribute workshops (QAWs)* (3rd ed.; CMU/SEI-2003-TR-016) Retrieved from http://www.sei.cmu.edu/reports/03tr016.pdf

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Reading, MA: Addison-Wesley.

Bergey, J. K. (2009). *A proactive means for incorporating a software architecture evaluation in a DoD system acquisition* (CMU/SEI-2009-TN-0004). Retrieved from http://www.sei.cmu.edu/reports/09tn004.pdf

Clements, P., et al. (2011). *Documenting software architectures: Views and beyond* (2nd Ed). Reading, MA: Addison-Wesley.

Clements, P., & Bass, L. (2010). *Relating business goals to architecturally significant requirements for software systems* (CMU/SEI-2010-TN-018). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Clements, P., Kazman, R., & Klein, M. (2002). *Evaluating software architectures: Methods and case studies*. Reading, MA: Addison-Wesley.

Kazman, R., & Bass, L. (2005). *Categorizing business goals for software architectures* (CMU/SEI-2005-TR-021). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Nord, R., Bergey, J., Blanchette, S., & Klein, M. (2009, April). *Impact of army architecture evaluations* (CMU/SEI 2009-SR-007). Retrieved from http://www.sei.cmu.edu/reports/09sr007.pdf

Software Engineering Institute (SEI). (2011). Risk mitigation through architecture evaluation. Retrieved from http://www.sei.cmu.edu/solutions/acquisition/riskmitigation.cfm

Appendix A. Steps of the Quality Attribute Workshop

QAW Steps	Description
1. QAW Presentation and Introductions	QAW facilitators describe the motivation for the QAW and explain each step of the method. Next, the facilitators and stakeholders introduce themselves ,briefly stating their background, their role in the organization, and their relationship to the system being built.
2. Business/Programmatic Presentation	A stakeholder representing the business and/or programmatic concerns presents the system's business/programmatic context, high-level functional requirements, constraints, and quality attribute requirements.
3. Architectural Plan Presentation	A technical stakeholder presents the system architectural plans, including (1) plans and strategies for how key business/programmatic requirements will be satisfied; (2) key technical requirements, risks, and constraints—such as mandated operating systems, hardware, middleware, and standards—that will drive architectural decisions; (3) existing context diagrams, high-level system diagrams, and other written descriptions; (4) operational and system architectures, and architectural frameworks, tools, and architectural life-cycle processes being used; and (5) the prototyping and engineering studies underway to mitigate known risks.
4. Identification of Architectural Drivers	The facilitators share their list of key architectural drivers that include high-level requirements, business drivers, constraints, and quality attributes.
5. Scenario Brainstorming	The facilitators ask the stakeholders to brainstorm scenarios that are operationally meaningful with respect to the stakeholders' individual roles.
6. Scenario Consolidation	Similar scenarios are consolidated when reasonable.
7. Scenario Prioritization	Stakeholders vote to establish the priorities of the scenarios.
8. Scenario Refinement	The high-priority scenarios are refined in more detail. Facilitators further elaborate each one, documenting the following: the six parts of the scenario, the business/programmatic goals that are affected by this scenario, the relevant quality attributes associated with this scenario, and the questions and issues regarding the scenario.

Appendix B. Steps of the Architecture Tradeoff Analysis Method

There are nine specific steps in the basic ATAM evaluation that fall into four general types of activities: presentation, investigation and analysis, testing, and reporting.

Step	Description
Presentation Activities	
1. Present the ATAM	The method is described to the assembled stakeholders (typically customer representatives, the architect or software architecture team, user representatives, maintainers, administrators, managers, testers, integrators, etc.).
2. Present business drivers	The project manager describes the business goals that are motivating the development effort and, hence, the primary software architecture drivers (e.g., broad availability, time to market, high security).
3. Present software architecture	The architect describes the proposed software architecture, focusing on how it addresses the business drivers.
Investigation and Analysis Activities	
4. Identify architectural approaches	Architectural approaches are identified by the architect, but they are not analyzed.
5. Generate quality attribute utility tree	The quality attributes that comprise system "utility" (e.g., performance, reliability, security, modifiability, etc.) are elicited. These are specified down to the level of scenarios, annotated with stimuli and responses, and prioritized. A scenario is a short statement describing an interaction of a stakeholder with the system. Scenarios provide a vehicle for making vague qualities concrete.
6. Analyze architectural approaches	Based upon the high-priority factors identified in Step 5, the architectural approaches that address those factors are elicited and analyzed. For example, an architectural approach aimed at meeting performance goals will be subjected to a performance analysis. During this step, software architecture risks, sensitivity points, and tradeoff points are identified.
Testing Activities	
7. Brainstorm and prioritize scenarios	Based upon the example scenarios generated in the utility tree step, a larger set of scenarios is elicited from the entire group of stakeholders. This set of scenarios is prioritized via a voting process involving the entire stakeholder group.
8. Analyze architectural approaches	This step reiterates Step 6; but here, the highly ranked scenarios from Step 7 are considered to be test cases for software architecture approaches determined thus far. These test case scenarios may uncover additional software architecture approaches, risks, sensitivity points, and tradeoff points, which are then documented.
Reporting Activity	
9. Present results	Based upon the information collected during the ATAM evaluation (e.g., styles, scenarios, attribute-specific questions, the utility tree, risks, sensitivity points, tradeoffs), the evaluation team presents its findings to the assembled stakeholders and details this information, along with any proposed mitigation strategies, in a written report.