Quality Attributes

Architecting Systems to Meet Customer Expectations

Paul R. Croll, CSC

Abstract. This paper addresses the use of quality attributes as a mechanism for making objective decisions about architectural tradeoffs and for providing reasonably accurate predictions about how well candidate architectures will meet customer expectations. Typical quality attributes important to many current systems of interest include performance, dependability, security, and safety. This paper begins with an examination of how quality attributes and architectures are related, including some the seminal work in the area, and a survey of the current standards addressing product quality and evaluation. The implications for both the customer and the system developer of employing a quality-attribute-based approach to architecture definition and tradeoff are then briefly explored. The paper also touches on the relationship of an architectural quality-attribute-based approach to engineering process and process maturity. Lastly the special concerns of architecting for system assurance are addressed.

It Is About the Architecture

A recent presentation on systemic root cause analysis of failures in DoD programs [1] pointed out that:

"DoD operational test and evaluation results from October 2001 through September 2006 indicated that of 29 systems evaluated, approximately. 50% were deemed 'Not Suitable', or 'partially Not Suitable' and approximately 33% were deemed 'Not Effective', or 'partially Not Effective'."

The presentation went on to say that one of the top 10 emerging systemic issues, from 52 in-depth program reviews since March 2004 was inadequate software architectures.

If we are to be successful in delivering systems that meet customer expectations, we must start as early as possible in the design process to understand the extent to which those expectations might be achieved. As we develop candidate system architectures and perform our architecture tradeoffs, it is imperative that we define and use a set of quantifiable system attributes tied to customer expectations, against which we can measure success.

In 2006, the National Defense Industrial Association (NDIA) convened a Top Software Issues Workshop [2] to examine the current most critical issues in software engineering that impact the acquisition and successful deployment of software-intensive systems.

The workshop identified 85 issues for further discussion, which were consolidated into a list of the top seven. Of those issues impacting software-intensive systems throughout the lifecycle, two emerged that were focused specifically on the relationship between software quality and architecture:

- Ensure defined quality attributes are addressed in requirements, architecture, and design.
- Define software assurance quality attributes that can be addressed during architectural tradeoffs.

As is true in the defense systems case above, most systems we encounter today contain software elements and most depend upon those software elements for a good portion of their functionality. Modern systems architecture issues cannot be adequately addressed without considering the implications of software architecture.

Architecture and Quality

What is an architecture? IEEE Std 1471-2000 [3, 34] defines an architecture for software intensive systems as:

"The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."

More recently, Firesmith et al [4], their Method Framework for Engineering System Architectures (MFESA), have defined system architecture as:

"The set of all of the most important, pervasive, higher-level, strategic decisions, inventions, engineering tradeoffs, assumptions, and their associated rationales concerning how the system meets its allocated and derived product and process requirements."

The authors believe that system architecture is a major determinant of resulting system quality.

MFESA instructs that architectures can be represented by models, views, and focus areas. Models describe system structures in terms of their architectural elements and the relationships between them. These descriptions can be graphical or textual and include the familiar data and control flow diagrams, entity-relationship diagrams, and UML diagrams and associated use cases. Views are composed of one or more related architectural models. They use the example of a class view that describes all architectural classes and their relationships. Focus areas combine multiple views and models to determine how the architecture achieves specific quality characteristics.

What is quality and what are quality characteristics? IEEE Standard 1061-1998 [5], defines software quality as the degree to which software possesses a desired combination of attributes.

Similarly, ISO/IEC 9126-1:2001 [6], one of a four-part set of standards on software product quality, defines quality as:

"The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs."

The standard identifies a quality model with six quality characteristics: functionality, reliability, usability, efficiency, maintainability and portability. The other three standards in the 9126-series [7, 8, 9] address metrics for measuring attributes of the quality characteristics defined in ISO/IEC 9126-1.

It should be noted that the 9126-series is being revised as part of the Software Product Quality Requirements and Evaluation (SQuaRE) series of standards. ISO/IEC 25010, Software engineering—SQuaRE—quality model [10] is the revision of ISO/IEC 9126-1:2001. ISO/IEC 25010 adds security and interoperability to the list of six quality characteristics defined in ISO/IEC 9126-1:2001. Additionally ISO/IEC 25030, Software engineering—SQuaRE)—quality requirements [11] defines the concept of internal software quality as the "capability of a set of static attributes (including those related to software architecture) to satisfy stated and implied needs when the software product is used under specified conditions." and the concept of

software quality in use, which is "the capability of the software product to enable specific users to achieve specific goals with effectiveness, productivity, safety and satisfaction in specific contexts of use."

Functional properties determine what the software is able to do. Quality properties determine how well the software performs. In other words, the quality properties show the degree to which the software is able to provide and maintain its specified services.

ISO/IEC/IEEE 15288 [12] addresses the confluence of architecture and quality in the context of the system lifecycle. The Architectural Design Process (6.4.3) provides for the creation of design criteria for quality characteristics and the evaluation of alternative designs with respect to those criteria. There is also a Specialty Engineering view of the lifecycle processes in that focuses on the achievement of product characteristics that have been selected as being of special interest.

Quality Attribute-based Approaches to Architecting Systems

In the seminal report on Quality Attributes by Barbacci et al [13] the authors indicate that:

"Developing systematic ways to relate the software quality attributes of a system to the system's architecture provides a sound basis for making objective decisions about design tradeoffs and enables engineers to make reasonably accurate predictions about a system's attributes that are free from bias and hidden assumptions. The ultimate goal is the ability to quantitatively evaluate and trade off multiple software quality attributes to arrive at a better overall system."

Franch and Carvallo [14] suggest that for an effective quality model, the relationships between quality attributes must be explicitly stated to understand potential attribute clash when defining software architectures. They posit three types of relationships between attributes:

- Collaboration, in which increasing the degree to which one attribute is realized increases the realization of another.
- Damage, in which increasing the degree to which one attribute is realized decreases the realization of another.
- Dependency, in which the degree to which one attribute is realized, is dependent upon the realization of at least some sub-characteristics of another.

For example, as Häggander et al [15] point out using the example of a large telecommunication application, system architects must balance multiple quality attributes, such as maintainability, performance and availability. Focusing solely on the attribute of maintainability often results in poor system performance and conversely focusing on performance and availability alone may result in result in poor maintainability. Explicit architectural decisions can facilitate optimization among quality attributes.

Architectural Design and Tradeoff

Bass and Kazman [16] suggest five foundational structures that together completely describe an architecture and that can serve as the basis for understanding the relationship of architectural decisions to quality attributes:

- Functional structure is the decomposition of the functionality that the system needs to support
- Code structure is the code abstractions from which the system is built.
- Concurrency structure is the representation of logical con currency among the components of the system.
- Physical structure is just that, the structure of the physical components of the system.
- Developmental structure is the structure of the files and the directories identifying the system configuration as the system evolves.

Bass and Kazman [16] further suggest some likely relationships between the architectural structures described above and examination of the impact of architectural decisions upon specific quality attributes. They suggest for example that:

- Concurrency and physical structures are useful in understanding system Performance.
- Concurrency and code structures are useful in understanding system security.
- Functional, code, and developmental structures are useful in understanding system maintainability.

Wojcik et al [17] describe an Attribute-driven Design (ADD) method in which the approach to defining software architecture is based on software quality attribute requirements. ADD produces an initial software architecture description from a set of design decisions that show:

- Partitioning of the system into major computational and developmental elements.
- What elements will be part of the different system structures, their type, and the properties and structural relations they possess.
- What interactions will occur among elements, the properties of those interactions, and the mechanisms by which they occur.

In the very first step in ADD, quality attributes are expressed as the system's desired measurable quality attribute response to a specific stimulus. Knowing these requirements for each quality attribute supports the selection of design patterns and tactics to achieve those requirements.

Kazman et al [18] describe an Architecture Tradeoff Analysis Method (ATAM) that can be used when evaluating an architecture, including those produced by the ADD method above, in order to understand the consequences of architectural decisions with respect to quality attributes. As the authors point out, ATAM is dependent upon quality attribute characterizations, like those produced through ADD, that provide the following information about each attribute:

- The stimuli to which the architecture must respond.
- How the quality attribute will be measured or observed to determine how well it has been achieved.
- The key architectural decisions that impact achieving the attribute requirement.

ATAM takes proposed architectural approaches and analyzes them based upon quality attributes, generally specified in terms of scenarios addressing stimuli and responses. ATAM also identifies sensitivity points and tradeoff points.

ATAM describes stakeholders' interaction with the system. Stakeholders bring different views to the system and may include users, maintainers, developers, and acquirers. Scenarios specify the kinds of operations over which performance needs to be measured, or the kinds of failures the system will have to withstand. ATAM uses three types of scenarios:

- Use case scenarios, describing typical uses of the system.
- Growth scenarios, addressing planned changes to the system.
- Exploratory scenarios, addressing any possible extreme changes that would stress the system.

Making the Case for Architectural Quality

How do stakeholders know that the system will exhibit expected quality characteristics? Firesmith et al [4] suggest that one method is the quality case, or more specifically for evaluating architectures, the architectural quality case. Quality cases consist of the set of claims, supporting arguments, and supporting evidence that provide confidence that the system will in fact demonstrate its expected quality characteristics. Common types of quality cases include safety cases [19], and security cases [20], and the more generalized assurance cases [21]. Architectural quality cases describe the architectural claims, supporting arguments, including architectural decisions and tradeoffs, architectural representations, and demonstrations that the architecture will exhibit its expected quality characteristics.

The implications for both the customer and the system developer of employing a quality-attribute- based approach to architecture definition and tradeoff, documented in part by a quality case, are that:

- Customer quality requirements will have been distilled into architectural drivers [17] that will have shaped the system architecture.
- Tradeoffs will have been made to optimize the realization of important quality characteristics, in concert with customer expectations.
- The level of confidence that the resultant architecture will meet those expectations will be known.
- Customers will be knowledgeable of any residual risk they are accepting by accepting the delivered system.

There are also architectural implications regarding sustainment of a system over its lifecycle. Croll [22] cites that with respect to sustainment, paying insufficient attention to sustainment issues early in the lifecycle, including licensing, and product support can lead to problems when commercial products or components inevitably change or when their suppliers either discontinue support or go out of business. In the hardware world Diminishing Manufacturing Sources and Materials Shortage (DMSMS) analyses are generally done when integrating commercial components as part of an approach for managing the risk of obsolescence [23]. DMSMS analyses focus on supplier viability for the product of interest, which could be considered an attribute of component maintainability. Certainly, such analyses, where necessary, should be part of the quality case. The specification and realization of architectures which are resilient with respect to the substitution of alternate software components can further enhance system quality through the lifecycle.

Process Maturity Does Not Guarantee Product Quality

We spend much time these days focusing on the maturity of our engineering processes and heralding process maturity ratings such as those associated with the CMMI® [24], for development and the ISO 9000 series [25][26][27], for quality management systems, as indicators of our ability to deliver quality products – products that meet the customer's expectations and that continue to do so throughout their lifecycle. What our customers have found, however, is that often process maturity does not guarantee product quality. This is especially true for the highly software intensive systems we now build, where performance, dependability, and failure modes are less well understood.

For example, although the CMMI embodies the process management premise that, the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it [24], Hefner [28] points out:

Several recent program failures from organizations claiming high maturity levels have caused some to doubt whether CMMI improves the chances of a successful project.

He goes on to say, "an CMMI appraisal indicates the organization's capacity to perform the next project, but cannot guarantee that each new project will perform in that way."

Understanding and Leveraging a Supplier's CMMI Efforts: A Guidebook for Acquirers [29] further underscores the problem and offers several cautions for acquirers, with respect to supplier claims of process maturity.

- A CMMI rating or CMMI level is not a guarantee of program success.
- Organizations that have attained CMMI maturity level ratings do not necessarily apply those appraised processes to a new program at program startup.
- Organizations that claim CMMI ratings are not always dedicated to [maintaining] process improvement [through out the development effort].
- Organizations may sample only a few exemplar programs and declare that all programs are being executed at that CMMI level rating.
- Organizations that claim a high maturity level rating (level 4 and 5) are not necessarily better suppliers than a level 3 supplier. Maturity levels 4 and 5, when compared across different suppliers, are not created equal.

Although process maturity can in many cases improve project performance [30], special attention to the engineering processes is required to ensure that customer quality expectations are realized in resultant products.

A Current Concern: Architecting for System Assurance

Stakeholder discussion over the last several years has demonstrated a reasonably consistent view of the problem space. System assurance can be viewed as the level of confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system. The President's Information Technology Advisory Committee report entitled Cyber Security: A Crisis of Prioritization [31] states, "… the approach of patching and retrofitting networks, computing systems, and software to 'add' security and reliability may be necessary in the short run but is inadequate for addressing the Nation's cyber security needs." The report further suggests, "we simply do not know how to model, design, and build systems incorporating integral security attributes."

As Croll points out [22], the systems engineering challenge, with respect to assurance, is in integrating a heterogeneous set of globally engineered and supplied proprietary, open-source, and other software; hardware; and firmware; as well as legacy systems; to create well-engineered integrated, interoperable, and extendable systems whose security, safety, and other risks are acceptable—or at least tolerable.

Baldwin [32] underscores this challenge for DoD systems by describing a vision for assurance in which the requirements for assurance are allocated among the right systems and their critical components, and such systems are designed and sustained at a known level of assurance.

The National Defense Industrial Association System Assurance Guidebook [33] describes practices in architectural design that can improve assurance. The Guidebook suggests some general architectural principles for assurance:

- Isolate critical components from less-critical components.
- Make critical components easier to assure by making them smaller and less complex.
- Separate data and limit data and control flows.
- Include defensive components whose job is to protect other components from each other and/or the surrounding environment.
- Beware of maximizing performance to the detriment of assurance.

The Guidebook also suggests using system assurance requirements, design constraints and system assurance critical scenarios for architectural tradeoff analysis, and documenting the results in the assurance case.

Summary

If we are to be successful in delivering systems that meet customer expectations, we must start as early as possible in the design process to understand the extent to which those expectations might be achieved. As we develop candidate system architectures and perform our architecture tradeoffs, it is imperative that we define and use a set of quantifiable quality attributes tied to customer expectations, against which we can measure success.

Standards like ISO/IEC TR 9126, Parts 1-4, ISO/IEC 25010, and ISO/IEC 2530 can help stakeholders define quality attributes from both an internal perspective, useful for addressing architectural design, and a quality in use perspective addressing system realization.

Methods have been documented to aid in understanding the relationship of architectural decisions to quality attributes, for defining software architecture is based on software quality attribute requirements, and for understanding the consequences of architectural decisions with respect to quality attributes.

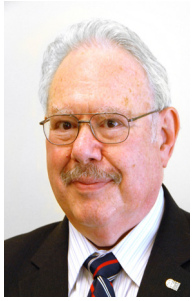
Architectural quality cases describe the architectural claims, supporting arguments, including architectural decisions and tradeoffs, architectural representations, and demonstrations that the architecture will exhibit its expected quality characteristics. They are extremely useful in providing customers with an understanding of any residual risk they are accepting by accepting the delivered system.

Several recent program failures from organizations claiming high maturity levels have caused some doubt about whether process maturity improves the chances of a delivering a successful product. This is especially true for the highly software intensive systems we now build, where performance, dependability, and failure modes are less well understood.

Of special concern these days is architecting systems for system assurance. Given our track record in architecting systems to meet assurance concerns, guidance is needed to support assurance-specific architectural design and tradeoff analysis, as well as appropriate documentation of assurance claims, arguments, and supporting evidence, so that customers understand the degree to which the architecture mitigates assurance risks.

Disclaimers:

ABOUT THE AUTHOR



Paul Croll is a Fellow in CSC's Defense Group where he is responsible for researching, developing and deploying systems and software engineering practices, including practices for cybersecurity.

Paul has more than 35 years experience in mission-critical systems and software engineering. His experience spans the full lifecycle and includes requirements specification, architecture, design, development, verification, validation, test and evaluation, and sustainment for complex systems and systems-of-systems. He has brought his skills to high profile, cutting edge technology programs in areas as diverse as surface warfare, air traffic control, computerized adaptive testing, and nuclear power generation.

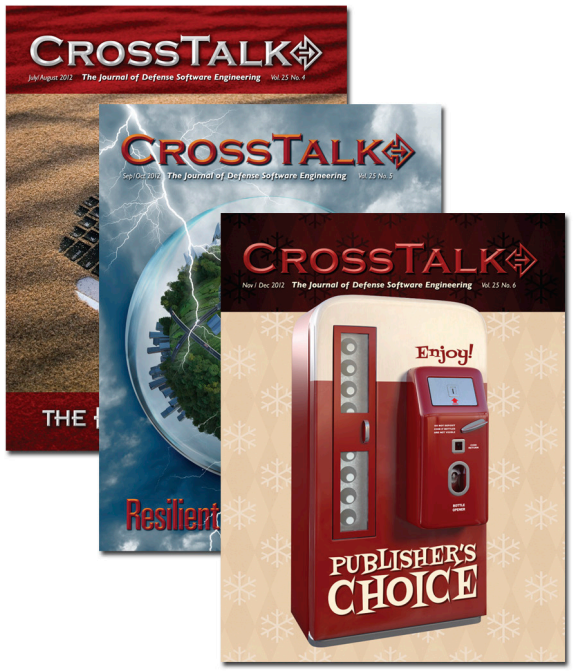
Paul is also the IEEE Computer Society Vice President for Technical and Conference Activities, and has been an active Computer Society volunteer for more than 25 years, working primarily to engage researchers, educators, and practitioners in advancing the state of the practice in software and systems engineering. He was most recently Chair of the Technical Council on Software Engineering and is also the current Chair of the IEEE Software and Systems Engineering Standards Committee. Paul is also the past Chair and current Vice Chair of the ISO/IEC JTC1/SC7 U.S. Technical Advisory Group (SC7 TAG).

Paul is also active in industry organizations and is the Chair of the NDIA Software Industry Experts Panel and the Industry Co-Chair for the National Defense Industrial Association Software and Systems Assurance Committees. In addition, Paul is Co-Chair of the DHS/DoD/NIST Software Assurance Forum Processes and Practices Working Group advancing cybersecurity awareness and practice.

CSC
17021 Combs Drive
King George, VA 22485
Phone: 540-644 6224
E-mail: pcroll@csc.com

REFERENCES

1. D. Castellano. Systemic Root Cause Analysis. NDIA Systems Engineering Division Strategic Planning Meeting, December, 2007.
2. G. Draper (ed.), Top Software Engineering Issues Within Department of Defense and Defense Industry. National Defense Industrial Association, Arlington, VA, August 2006.
3. IEEE 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, 2000.
4. D. Firesmith, P. Capell, D. Falkenthal, C. Hammons, D. Latimer, and T. Merendino. The Method-Framework for Engineering System Architectures (MFESA): Generating Effective and Efficient Project-Specific System Architecture Engineering Methods. To be published.
5. IEEE Standard 1061-1992. Standard for a Software Quality Metrics Methodology. New York: Institute of Electrical and Electronics Engineers, 1992.
6. ISO/IEC 9126-1: Information Technology - Software product quality - Part 1: Quality model. ISO, Geneva Switzerland, 2001.
7. ISO/IEC TR 9126-2: Software Engineering - Product quality - Part 2: External metrics. ISO/IEC, Geneva Switzerland, 2003.
8. ISO/IEC TR 9126-3 Software engineering – Product quality - Part 3: Internal metrics. ISO/IEC, Geneva Switzerland, 2003.
9. ISO/IEC TR 9126-4: Software engineering – Product quality - Part 4: Quality in use metrics. ISO/IEC, Geneva Switzerland, 2004.
10. ISO/IEC CD 25010, Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) Quality model. ISO/IEC, Geneva, Switzerland, 2007.
11. ISO/IEC CD 25030, Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Quality requirements. ISO/IEC, Geneva Switzerland, 2007.
12. ISO/IEC 15288:2002, Systems Engineering - System Life Cycle Processes, ISO/IEC, Geneva Switzerland, 2002.
13. M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock. Quality Attributes, CMU/SEI-95-TR-021. Software Engineering Institute, Carnegie Mellon University, December 1995.
14. X. Franch and J. Carvallo. "Using Quality Models in Software Package Selection", IEEE Software, pp. 34-41. New York: Institute of Electrical and Electronics Engineers, 2003.
15. D. Häggander, L. Lundberg, and J. Matton, "Quality Attribute Conflicts - Experiences from a Large Telecommunication Application," Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems (ICECCS'01), New York: Institute of Electrical and Electronics Engineers, 2001.
16. L. Bass and R. Kazman, Architecture-Based Development, CMU/SEI-99-TR-007. Software Engineering Institute, Carnegie Mellon University, April 1999.
17. R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood, Attribute-Driven Design (ADD), Version 2.0, CMU/SEI-2006-TR-023. Software Engineering Institute, Carnegie Mellon University, November 2006.
18. R. Kazman, M. Klein, and P. Clements, ATAM: Method for Architecture Evaluation, CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, August 2000.
19. W. Greenwell, E. Strunk, and J. Knight, Failure Analysis and the Safety-Case Lifecycle, IFIP Working Conference on Human Error, Safety and System Development (HESSD) Toulouse, France, August 2004.
20. Systems Security Engineering Capability Maturity Model®, SSE-CMM®, Model Description Document Version 3.0. Systems Security Engineering Capability Maturity Model (SSE- CMM) Project (Copyright © 1999), June 15, 2003
21. T. Ankrum and A. Kromholz, "Structured Assurance Cases: Three Common Standards, Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05). New York: Institute of Electrical and Electronics Engineers, 2005.
22. P. Croll, "Engineering for System Assurance – A State of the Practice Report," Proceedings of the 1st Annual IEEE Systems Conference. New York: Institute of Electrical and Electronics Engineers, April 2007.
23. Diminishing Manufacturing Sources and Material Shortages (DMSMS) Guidebook. Office of the Under Secretary of Defense Acquisition, Technology, & Logistics, November 2006.
24. CMMI® for Development, Version 1.2, CMU/SEI-2006-TR-008, Software Engineering Institute, Carnegie Mellon University, August 2006.
25. ISO 9000:2000, Quality management systems – Fundamentals and vocabulary. ISO, Geneva Switzerland, 2000.
26. ISO 9001:2000, Quality management systems – Requirements. ISO, Geneva Switzerland, 2000.
27. ISO 9004:2000 Quality management systems – Guidelines for performance improvements. ISO, Geneva Switzerland, 2000.
28. R. Hefner. CMMI Horror Stories: When Good Projects Go Bad. SEPG Conference, March 2006
29. Understanding and Leveraging a Supplier's CMMI® Efforts: A Guidebook for Acquirers, CMU/SEI-2007-TR-004. Software Engineering Institute, Carnegie Mellon University, March 2007.
30. D. Goldenson and D. Gibson, Measuring Performance: Evidence about the Results of CMMI®. 5th Annual CMMI Technology Conference & User Group. National Defense Industrial Association, System Assurance Committee, Arlington, Virginia, November 2005.
31. President's Information Technology Advisory Committee (PITAC), Cyber Security: A Crisis of Prioritization. National Coordination Office for Information Technology, Arlington, VA, 2005.
32. K. Baldwin. DOD Software Engineering and System Assurance New Organization – New Vision, DHS/DOD Software Assurance Forum, March 8, 2007.
33. National Defense Industrial Association System Assurance Guidebook, Version 0.89. National Defense Industrial Association, System Assurance Committee, Arlington, Virginia February 2008.
34. IEEE Std 1471-2000 has been superseded by ISO/IEC/IEEE 42010-2011 - Systems and software engineering – Architecture description



CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

25th Year Anniversary
Jul/Aug 2013 Issue
Submission Deadline: Feb 10, 2013

Securing the Cloud
Sep/Oct 2013 Issue
Submission Deadline: April 10, 2013

Real-Time Information Assurance
Nov/Dec 2013 Issue
Submission Deadline: June 10, 2013

Please follow the Author Guidelines for CROSSTALK, available on the Internet at <www.crosstalkonline.org/submission-guidelines>. We accept article submissions on software-related topics at any time, along with Letters to the Editor and BackTalk. To see a list of themes for upcoming issues or to learn more about the types of articles we're looking for visit <www.crosstalkonline.org/theme-calendar>.