QUALITY ASSESSMENT OF SYSTEM ARCHITECTURES AND THEIR REQUIREMENTS (QUASAR)

Donald G. Firesmith
Software Engineering Institute (SEI), Pittsburg, Pennsylvania, USA
Peter S. Capell
Software Engineering Institute (SEI), Pittsburg, Pennsylvania, USA

The quality of a software-intensive system is largely determined by the quality of its architecture and the quality of the architecturally significant requirements that drive its development. Unfortunately, although quality requirements typically have critical architectural ramifications, requirements engineers using such popular techniques as use case modelling tend to emphasize functional requirements over non-functional quality requirements, with the result that the quality requirements are often poorly specified or not specified at all. Similarly, system architects tend to emphasize a system's logical decomposition structure into major functions and subfunctions and the corresponding static physical decomposition structure into a hierarchy of subsystems. The system architects often do not adequately document how (or even if) these subsystems and subsubsystems collaborate to sufficiently support the achievement of their derived and allocated quality requirements. To address these two problem areas, the SEI QUality Assessment of software-intensive System Architectures and their Requirements (QUASAR) method was created to provide acquisition organizations with a proven and efficient means to determine if quality requirements have been properly engineered and if system architectures sufficiently meet these requirements.

Keywords: *system architecture, system requirements, quality factors, assessment*

1. Introduction

It is *not* sufficient for a software-intensive system to merely perform its required functions. The system's many stakeholders have a legitimate interest not just in what the system must do, but also how well, how often, and under what conditions the system must do it. In addition, stakeholders are interested not just in how the system must behave under normal circumstances, but also how the system must behave under exceptional situations when it cannot do what it must ordinarily do.

Thus to fulfil its mission, a system must exhibit a large number of mandatory quality factors (a.k.a., quality attributes, quality characteristics, and 'ilities'). For example, the system may need to have a certain minimum amount of availability; it may need to be interoperable with certain other systems; it may need to meet certain performance requirements; and it may need to achieve a minimum level of security. In fact, the number of quality factors and their component quality subfactors can be quite large. For example, one way that the quality factor 'performance' can be decomposed is into jitter, latency, response time, schedulability, and throughput.

The following are some of the lessons learned when applying the QUASAR method. Different quality factors are important for different systems and different subsystems. For example, performance is critical for real-time systems, whereas security is critical for any system dealing with confidential or classified information. Properly engineering quality requirements requires significant time and resources; it cannot be accomplished during a short assessment meeting. Engineering quality requirements is the responsibility of the requirements team, not the architecture or assessment teams who are not qualified to engineer them. Finally, it is too late to develop the architecturally significant requirements during an architecture assessment because these requirements were needed to drive the engineering of the architecture and its associated test cases.

- **Informal Peer Reviews are Insufficient**

  Currently, many projects rely largely on relatively informal peer-level reviews by other requirements engineers and architects within the development organization. But the system architecture and the architecturally significant quality requirements that drive the architecture are much too important to leave to informal peer reviews.

  While very useful, these reviews depend greatly on the expertise, experience, objectivity, and authority of the requirements engineers and architects who act as peer reviewers. They tend to lack adequate input from independent experts in specialty engineering areas (such as reliability, safety, security, and usability) and the engineering of non-functional requirements. This leads to reviews of widely varying efficiency and effectiveness.

- **Assess the Requirements and Architecture Early in the Development Process**

  It is impractical to test quality into non-trivial systems late during the development cycle; quality must be built into a system from the beginning. Because the quality of the system requirements and architecture have such a major impact on the quality of a system, they should be assessed early during the system development process when time and resources are still available to correct any defects found.

- **Define Architecture and Architecturally significant Requirements**

  There is a surprising amount of confusion within the system architecture community as to the meaning of the word 'architecture'. For example, many architects confuse representations of the architecture with the architecture itself. Many architects feel that a system architecture is nothing more than the top-level structure of the system, and therefore feel that a structural model is the architecture (e.g., the UML model or SysML model *is* the architecture). Even within architectural modeling, some system architects and system engineering organizations recognize only a very limited number of architectural structures (for example, a static logical functional decomposition of the system into functions and subfunctions and a static physical aggregational decomposition of the system into subsystems and subsubsystems). Yet as illustrated in Figure 1, a system has many different kinds of logical and physical static and dynamic structures that the architect should model and keep consistent. Additionally, architects make many architectural decisions that are not captured within a model, no matter how important the architectural models are. For example, architects may mandate the use of a single design approach, such as object-orientation, across the entire system or mandate that everyone use a safe subset of a given programming language. QUASAR therefore defines an architecture more generally as a collection of the most important, pervasive, top-level, strategic inventions, decisions, engineering trade-offs, associated rationales, and the assumptions about how a system and its subsystems will meet their derived and allocated requirements.

  Similarly, as mentioned in the first principle above, "functional requirements" is not the only or even most important type of architecturally significant requirements. Commonly, quality requirements have the most important impact on the architecture of a system

shown us that the best approach to assessing a complex architecture is by reviews organized according to respective quality factors ("ilities"). Attempts to assess the system architecture by subsystem make it difficult to determine if there is sufficient support for quality requirements, because the implementation of these requirements involves the collaboration of many subsystems spread across the system architecture.

- **Developers Must Make Their Case to the Assessment Team**

  It is the responsibility of the requirements engineers and architects to make their case for the quality of the requirements and architecture to the assessment team. The requirements engineers and architects are the people who best know what they did, why they did it, and where they documented it. Specifically, requirements engineers should know (a) the stakeholder needs and goals that drove their requirements, (b) the architecturally significant requirements they engineered including associated requirements decisions, inventions, trade-offs, assumptions, and rationales, and (c) where they specified this information. Similarly, the architects should know (a) the architecturally significant requirements that drove their architecture, (b) their architectural inventions, decisions, trade-offs, assumptions, and rationales, and (c) where they documented their architectural work products.

  It would be highly inefficient, ineffectual, and inappropriate to have the assessment team independently try to show that the requirements or architecture have (or do not have) sufficient quality. Similarly, it is the responsibility of the requirements engineers and architects (and not the assessors) to ensure the quality of the requirements and architecture. Although there may exist times and circumstances under which it would be appropriate for the acquisition organization to collaborate with the development organization to jointly engineer the requirements and the architecture, it is *not* appropriate during an assessment for members of the assessment team to either help requirements engineers and architects do their work or to do their work for them. Rather, the assessors should restrict themselves to pointing out defects and areas of risk so that the relevant stakeholders understand the current state of the requirements and architecture and so that the requirements engineers and architects can fix the defects and manage the risks.

- **Assessors Must Actively Assess the Requirements and Architecture Teams' Quality Cases**

  The assessment team should not just passively listen to the quality cases made by the requirements and architecture teams, taking what is presented on faith. Members of the assessment teams should use their professional training, experience, and judgment to assess whether the claims are correct and complete for all relevant quality factors, quality subfactors, quality goals, and quality requirements. They should assess whether the arguments are clear, compelling, correct, complete, and sufficient to justify belief in the associated claims. They should assess whether the evidence is credible (i.e., official) and sufficient to support the arguments. The assessment team should also assess whether the claims, arguments, and evidence is appropriate for the point in the system development/life cycle during which the assessment takes place.

  The assessment team is responsible for ensuring that the requirements engineers and architects have made their critical requirement and architecture decisions, inventions, trade-offs, assumptions, and rationales visible to the important stakeholders who take part in the assessment, particularly to the members of the acquisition organization with the authority to control the passing of major milestones. The work products used as evidence during assessments should be restricted to "official" project documentation, generated during the normal day-to-day development of the system requirements and architecture. This ensures the credibility of the evidence and minimizes the amount of additional documentation to be specifically developed for the assessment.

- **Ensure that the Assessment Method is Well Documented and Repeatable**

  The assessment method should have a regular and well-defined structure consisting of phases, tasks, steps, roles, and work products so that the method is repeatable. The method should be both

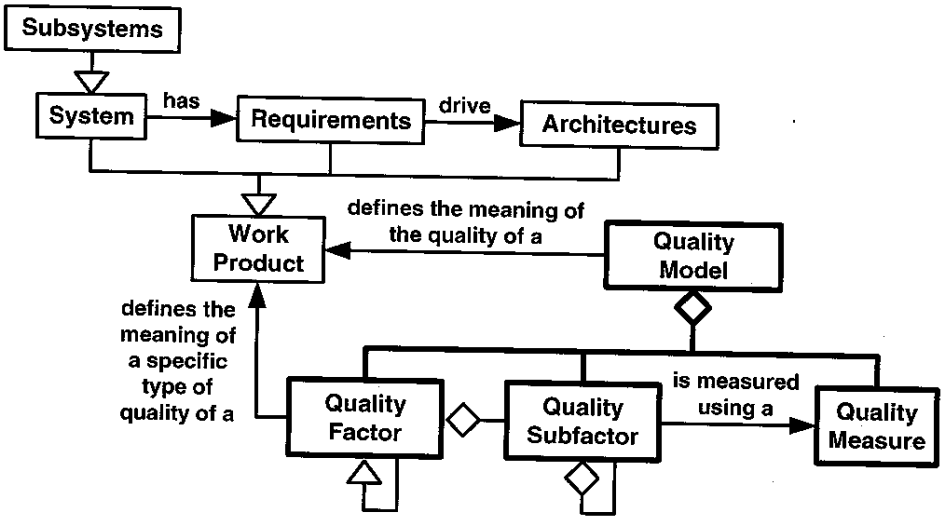

Fig. 2 Structure of a Quality Model

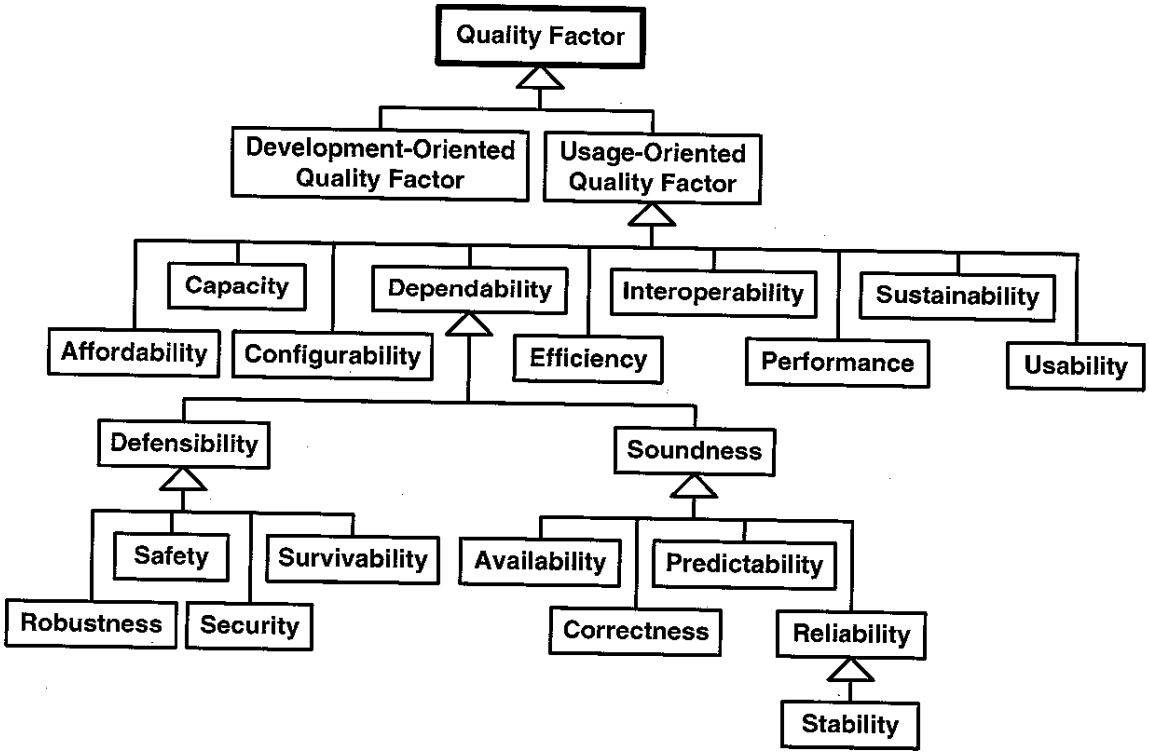

Fig. 3 Decomposition of Quality Factor

Quality Measures

A quality measure is defined as a unit of measure that provides a range of numerical values enabling the measurement of the quality of a work product or process by documenting the degree to which the work product or process possesses (or shall possess) a specific quality factor or

teams to provide adequate arguments and evidence to convince the assessment team that their architecture sufficiently supports these quality requirements.

5. The Foundation of QUASAR: Quality Cases

QUASAR is based on the concept of quality cases, a generalization of safety cases from the safety community [Bishop and Boomfield 1999]. Specifically, QUASAR uses requirements quality cases for the assessment of architecturally significant system requirements and architectural quality cases for the assessment of system architectures. Quality cases are used by the system requirements teams and architecture teams to make their case for the sufficiency of the quality of their requirements and architectures respectively. As illustrated in Figure 5, a quality case is composed of the following three types of components:

Claims

These are the developers' claims that their work products (such as requirements or architecture work products) sufficiently support achieving system quality (whereby quality is defined in terms of the qualify factors and quality subfactors defined in the official project quality model) and are themselves sufficiently complete and of sufficient quality.

Arguments

There should be clear, compelling, and relevant developer arguments justifying the assessors' belief in the developers' claims. Arguments essentially consist of the developers' decisions, inventions, analysis and simulation results, trade-offs, associated rationales, and assumptions.

Evidence

There should be sufficient credible evidence to support the developers' arguments. Acceptable evidence includes official project diagrams, models, requirements specifications and architecture documents; requirements repositories; analysis and simulation reports; test results; and demonstrations witnessed by the assessors.

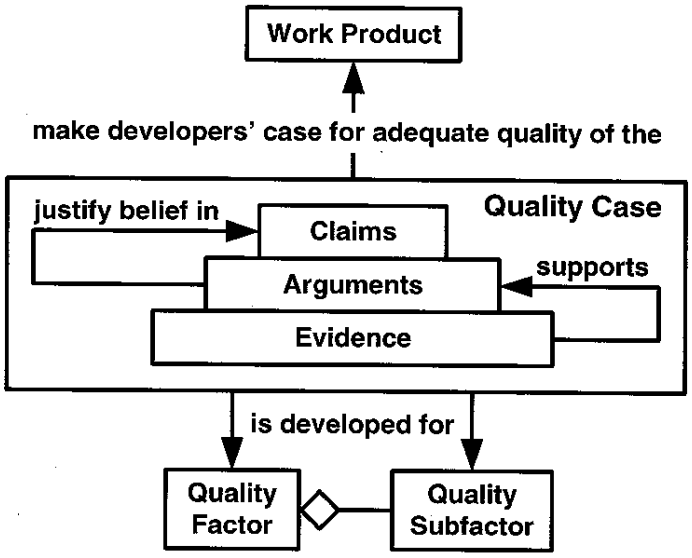

Figure 5: Structure of a Quality Case

For each architecturally significant quality factor or quality subfactor, the requirements teams develop and present corresponding requirements quality cases and the architecture teams develop and present corresponding architectural quality cases. Quality cases are based on existing project evidence,

engineers and architects understand the level of detail and completeness needed to answer the questions 'How good is good enough?' and "How complete is complete enough?"

6. Example Architecture Quality Case

The following is a general representative example architecture quality case for the quality subfactor protocol interoperability. As with any architecture quality case, it consists of the architects' claims, arguments, and evidence:

Claims

The subsystem X architects would make the following claims concerning the sufficiency of their architecture to meet its derived and allocated reliability-related goals and requirements:

1. Subsystem X architecture sufficiently supports achievement of the following protocol interoperability goals:
    1. Subsystem X correctly uses the interface protocols of all relevant external systems.
    2. Subsystem X will use open interface standards (i.e., industry standard protocols) when communicating with all external systems
2. Subsystem X architecture sufficiently supports achievement of the following protocol interoperability requirements:
    1. The subsystem shall use open interface standards (i.e., industry standard protocols) when communicating with external systems across all key interfaces identified in document X.
    2. The subsystem shall use the Ethernet over RS-232 for communication across interface X with external system Y
    3. The subsystem shall use HTTPS for communicating securely when performing function X across interface Y with external system Z

Arguments:

The subsystem X architects present their following arguments to the assessors:

1. **Layered Architecture**
   The subsystem uses a layered architecture.
   Rationale: Interface layer supports interoperability.
2. **Modular Architecture**
   The subsystem architecture is highly modular.
   Rationale: Architecture includes modules (proxies) for interoperability.
3. **Wrappers and Proxies**
   The subsystem architecture includes proxies that wrap the interfaces to external subsystems.
   Rationale: Proxies localize and wrap external interfaces.
4. **Service Oriented Architecture (SOA)**
   The subsystem service oriented architecture uses XML, SOAP, and UDDI to publish and provide Web services over the Internet to external client systems.
   Rationale: Standard languages and protocols support interoperability between heterogeneous systems.

Evidence:

The subsystem X requirements engineers would make the following evidence available to the assessment team and be able to present it at the assessment meeting if called on to do so:

1. **Context Diagram**
   (shows external interfaces requiring protocols)
2. **Architectural Class Diagrams**
   (show modularity and the location of proxies and Web services)
3. **Allocation Diagram**
   (shows modularity and the allocation of software modules to hardware)

An SAI meeting typically lasts from ½ day to 2 days, depending on the speed with which consensus can be reached.

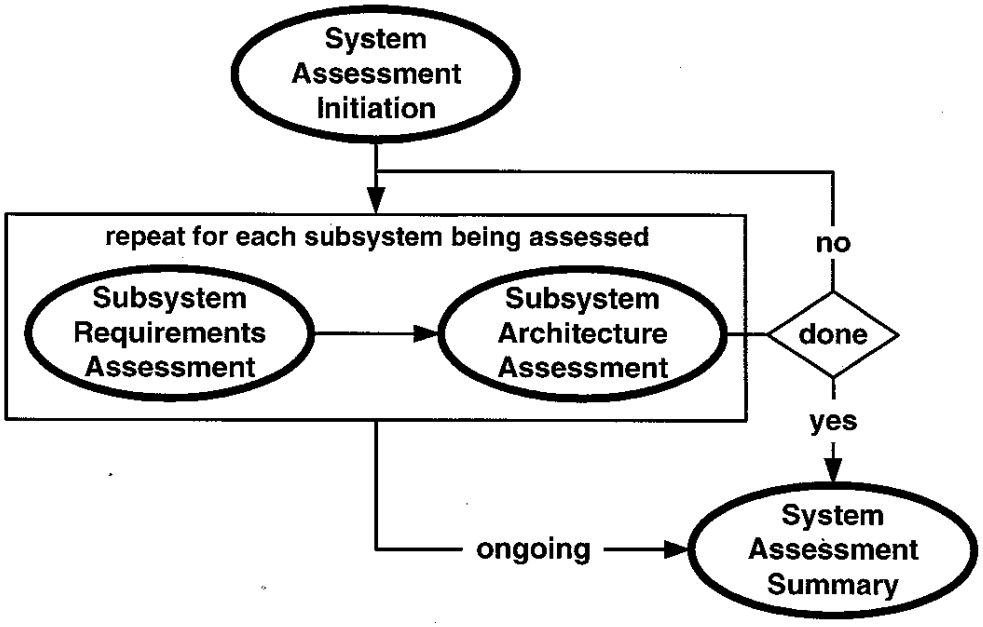

Figure 8: Overview of QUASAR Assessment Process

Phase 2) Subsystem Requirements Assessment (SRA)

During this subsystem-specific phase, the requirements team presents its requirements quality cases, upon which the assessment team makes its assessment of the sufficiency of the requirements to meet the quality-related goals of the stakeholders, which were discovered and documented during requirements engineering. The architecture team also presents a representative sample of the kinds of information that it intends to include in its architecture quality cases.

To provide adequate time to review the preparatory materials, the preparation task of an SRA should typically last from one to three calendar weeks. The amount of effort needed to prepare these materials naturally varies depending on the prior existence of well-engineered architecturally significant subsystem requirements. Because the SRA meeting can be expected to take 1-2 hours per requirements quality case, the duration of the meeting will depend on the number of quality factors assessed. From one to two weeks may pass during the follow-through task while the SRA assessment report is produced. The amount of effort invested in subsystem requirements assessments also naturally depends on the number of subsystems assessed.

Phase 3) Subsystem Architecture Assessment (SAA)

During this second subsystem-specific phase, the architecture team presents its architecture quality cases, upon which the assessment team makes its assessment of the sufficiency of the architecture to meet its derived and allocated quality-related requirements.

To provide adequate time to review the preparatory materials, the preparation task of an SAA should also typically last from one to three calendar weeks. The amount of effort needed to prepare these materials will also naturally vary depending on the prior existence of a well-documented subsystem architecture. Because the SAA meeting can be expected to take 1-2 hours per architecture quality case, the duration of the meeting will depend on the number of quality factors assessed. From one to two weeks may pass during the follow-through task while the SAA

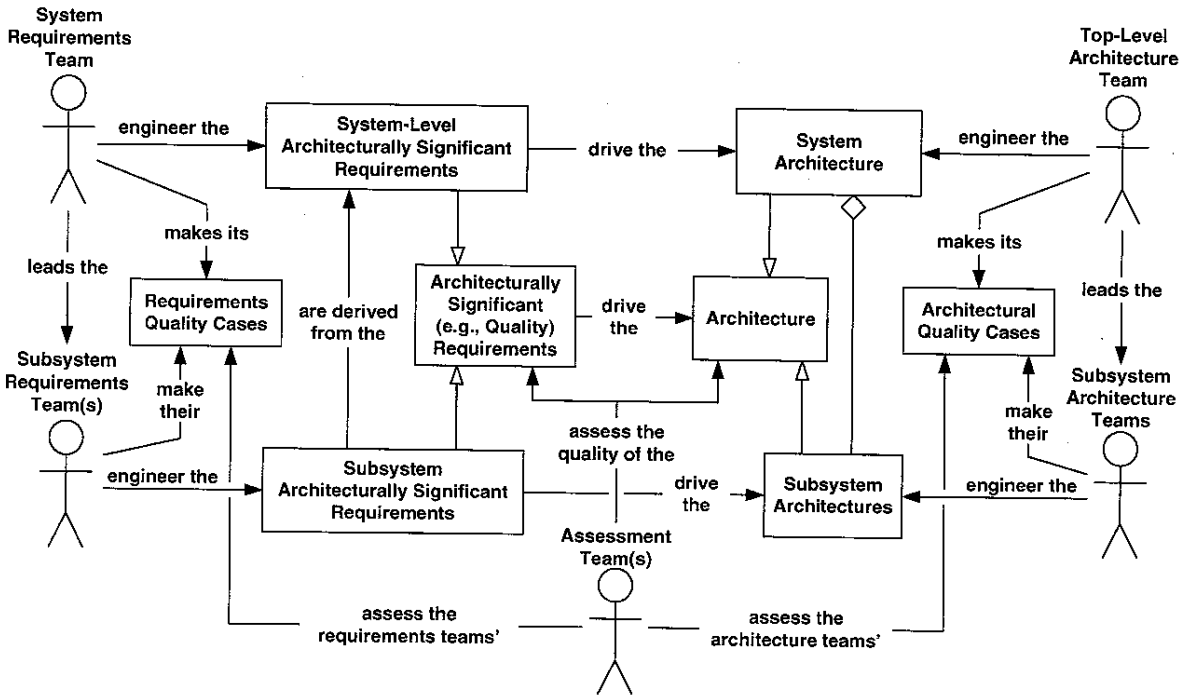

Figure 10: QUASAR Team Responsibilities

8. Why Use QUASAR?

There are many reasons why one should consider using QUASAR to perform quality assessments of system architectures and their requirements. Requirements and architecture are the first two opportunities to make major engineering mistakes. The QUASAR method helps identify these mistakes early so that they can be fixed early, which has long been known to significantly save both cost and schedule. The QUASAR method assesses the architecture and associated architecturally significant requirements, which significantly affect project organization and staffing according to Conway's Law [Wikipedia 2007]. The architecture and associated requirements also significantly affect system design, implementation, integration, testing, and deployment decisions. QUASAR emphasizes using a common project-specific quality model consisting of quality factors and quality subfactors to assess the quality of the requirements, architecture, and thereby the system. QUASAR ensures specification of *architecturally significant* requirements. QUASAR assessments help ensure that these requirements do not fall through the cracks. QUASAR assessments concentrate on the architecturally significant, quality-related requirements and their associated architectural decisions. These in turn *drive* the system/subsystem ultimate quality, development schedule, development costs, sustainment costs, maintainability and upgradeability, and acceptance and usage by stakeholders. The QUASAR method is very flexible in that it works in conjunction with any effective requirements engineering and architecting methods that the development organization may use. QUASAR uses existing requirements and architecture work products so that very few new work products are required. QUASAR can be used to assess any set of subsystems based in need, risk, and available resources (i.e., fits any system size, budget, schedule, and tier within the system hierarchy). QUASAR can assess subsystems in terms of any appropriate quality factors and quality subfactors such as availability, capacity, interoperability, modifiability, performance, reliability, robustness (including error, failure, and fault tolerance), safety, security, scalability, stability, and testability. QUASAR assessments help

11. Appendix

Donald Firesmith is a senior member of the technical staff at the Software Engineering Institute (SEI), where he helps the US Government acquire large, complex, software-intensive systems. Working in industrial software development since 1979, he has worked primarily with object technology since 1984 and has written 5 books on the subject. During the last five years, he has developed the world's largest (1,100+ webpage), free, and open source informational website of reusable process engineering components for constructing project-specific development methods. Based on the OPEN Process Framework (OPF), it is located at http://www.opfro.org. Currently completing his next book on the engineering of safety- and security-related requirements for software-intensive system, he can be reached at dgf@sei.cmu.edu. More information on Donald Firesmith including published books, published papers, and leadership positions is found at: http://www.donald-firesmith.com/Firesmith/Firesmith.html.

Dr. Peter Capell is a senior member of the technical staff at the Software Engineering Institute. He joined the SEI in 1992 developing training and software process assessment methods. In 2000, Capell then joined the Carnegie Mellon Research Institute (CMRI) developing advanced online training systems. Capell was also Director of Production at MountainTop Technologies overseeing software and training systems development. Capell is Adjunct Faculty of the School of Computer Science at Carnegie Mellon, past Director and past Education Chair of the Pittsburgh Chapter of IEEE and member of Sigma Xi. Capell is the author of many publications related to software process improvement as well as intelligent tutoring systems