JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5. No. 2, March-April 2006

Architecture-Related Requirements

Donald Firesmith, Software Engineering Institute, U.S.A.
Peter Capell, Software Engineering Institute, U.S.A.

Abstract

The engineering of architecture-related requirements has proven to be a very difficult task for requirements engineers. It is also a task that experience has shown could use significant improvements in practice. These requirements are critically important because they drive the development of the system and software architectures, which in turn largely determine if major product qualities are adequately achieved. They also form the basis against which the architectures are assessed. In this column, we describe the three major kinds of architecture-related requirements, discuss the most important characteristics they should have, describe the responsibilities of their stakeholders, and warn of the major negative consequences they can have on downstream activities when they are not properly engineered.

1 INTRODUCTION

We have recently been supporting the assessment of the architecture of an extremely large and complex software-intensive system of systems. As with any other system, its architecture is primarily driven by its requirements. Of course not all requirements are equally important in their impact on a system's architecture. Some individual requirements have a major influence on the architects' choices, whereas other requirements will not change the architecture. Finally, although the writing of this column has been inspired by our participation in the architecture assessment, the issues raised are pervasive in all systems. In fact, the challenges in our experience occur in most projects from the development of the smallest software application to extremely large and complex systems of systems being attempted. Inadequate specification of architecture-related requirements has influenced other known methods of assessing the quality of software architectures [Clements et al. 2002]. The problem of inadequate architecture-related requirements begins with requirements engineering and is primarily the responsibility of requirements engineers to solve.

In this issue, we take up the issue of architecture-related requirements. What are they and how do they differ from other requirements? Are there different kinds of architecture-related requirements? What are the key characteristics of such requirements, and what are the negative consequences when such requirements are inadequately specified or not specified at all. Who are the different types of stakeholders of such requirements, and in what way are such requirements important to them?

2 ARCHITECTURE-RELATED REQUIREMENTS

As the name implies, **architecture-related requirements** are those requirements that have a significant impact on the architecture of a system. At the highest level of abstraction, these can be classified as:

- **Quality Requirements.** A quality requirement is a requirement specifying that a system must have a minimum required amount of a specific quality [Firesmith 2003b]. Quality requirements specify a minimum level of a quality factor such as affordability, availability, capacity, configurability, correctness, efficiency, extensibility, interoperability, maintainability, modifiability, performance, portability, producibility, reliability, reusability, robustness, safety, scalability, security, stability, sustainability, testability, and usability. A quality requirement specifies that under certain conditions, the system or subsystem shall exhibit a quality criterion demonstrating that one or more associated quality subfactors exist beyond a minimum threshold on an associated quality measure. For example, the following is an example of a quality (performance) requirement: "When not in degraded mode (*condition*), the mortgage processing system shall correctly process mortgage applications (*quality criterion*) with a throughput (*performance quality subfactor*) of at least 100 applications per second (*threshold on quality measure*)." On a requirement-by-requirement basis, quality requirements tend to have an inordinate influence on the architecture, much more than typical functional, data, or interface requirement.
- **Architecturally-Significant Requirements.** These are functional, data, and interface requirements that implicitly have a significant impact on the architecture. For example, the primary functional requirements associated with a major system function tend to have a major influence on *system* architectures because many systems are functionally decomposed into subsystems. On the other hand, functional requirements tend to have less of an impact on *software* architectures because they are often decomposed along different lines using software architecture patterns and different methods (e.g., object-orientation).
- **Architecture Constraints.** An architecture constraint is an architecture decision that is mandated on the architects as if it were a normal requirement. By its very nature as a mandated architectural choice, this kind of constraint clearly influences the architecture and thus meets the definition of an architecture-relevant requirement.

Key Characteristics of Architecture-Related Requirements

Properly-engineered requirements exhibit widely agreed-upon characteristics [Firesmith 2003a], and the following characteristics are therefore particularly important to the engineering of architecture-relevant requirements:

- **Complete.** In order to be considered "complete," requirements are written using systematic methods designed to express all relevant preconditions, (e.g. system modes, environmental states, and operational profile) under which the requirement is operative. Quality requirements also need to include quality criteria that describe the system in terms of some quality factor or subfactor as well as some minimum threshold on an associated quality measure. At an architectural level, incomplete requirements pose a number of risks such as invisible architecture-related implications, untraceable component failure modes, unanticipated requirements conflicts discovered downstream, and so on.
- **Feasible.** The feasibility of architecture-related requirements involves understanding what constraints a single requirement might have as a cascading consequence to the entire system. When vague goals such as "The system shall be reliable, safe, and secure" are incorrectly specified as requirements, they are inherently infeasible because no system is 100% reliable, safe, or secure no matter what architectural decisions are made.
- **Unambiguous.** Ambiguity in architecture-related requirements creates unnecessary conflict among stakeholders who will each have their own idea of what the requirement means. For example, unless a quality requirement quantitatively specifies exactly how much quality is "adequate," then there is a high risk that the customer, requirements engineer, architect, and tester will disagree as to the acceptability of the system and its architecture.
- **Verifiable.** Like all other requirements, architecture-related requirements should be verifiable via such traditional approaches as testing, demonstration, analysis, and inspection. Just because it is more difficult to verify some types of architecture-related requirements such as certain quality requirements, that is not an adequate reason to allow unverifiable requirements. If you cannot verify it, you do not know if you have met it. If you cannot verify architecture-related requirements, you do not know if you architecture is sufficient.
- **Validatable.** Requirements must be validatable in terms of the needs and desires of their stakeholders. Architecture-related requirements are in this respect no different except that their validation becomes a matter of system-wide concern where validity must be assessed across all major subsystems and their sub-subsystems.

3 STAKEHOLDERS OF THE REQUIREMENTS

Because of their critical role in the quality and acceptability of systems, architecture-related requirements are important to many different stakeholders, although in different ways and for different reasons. Specifically, the main stakeholders of architecture-related requirements are:

- **Requirements providers**, who are responsible for providing and validating architecture-related requirements of the system being specified.
- **Requirements engineers,** who are responsible for engineering them
- **Requirements evaluators**, who are responsible for evaluating their quality
- **Architects**, who are responsible for ensuring that their architectures sufficiently support them
- **Architecture assessors**, who are responsible for assessing the architecture against its support for them
- **System integrators**, who are responsible for passing on the architecture-related requirements to subcontractors who must supply the corresponding architectural components
- **Independent testers**, who are responsible for integration and system testing of the resulting architectural components

In the following sections, we define these roles and provide a snapshot of the problems that can arise when persons performing these roles fail to properly fulfill their assigned responsibilities.

Requirements Providers

Requirements providers (i.e., stakeholders such as customers, marketing representatives, sales representatives, user representatives, operators, user support engineers, etc.) have the following responsibilities with regard to architecture-related requirements:

- **Provide requirements.** These people must clearly state all of their needs that will significantly impact the systems' architecture. Especially important are necessary system qualities because the architects' major decisions will either enable the achievement of these system characteristics or make their achievement difficult if not impossible. Only these stakeholders can determine what system qualities the system must exhibit. And it is these system characteristics that will largely drive the development of the architecture.
- **Provide architecture constraints.** Systems do not exist in a vacuum. Few systems are truly "green field"; most are new versions of existing systems. Similarly, systems are typically deployed into existing environments and must interoperate with existing systems. Systems are often maintained by the acquiring organization, which has expertise in certain technologies. All of these are legitimate reasons for the customer to mandate architecture constraints on the development organization.
- **Validate requirements.** People who are sources of the architecture-related needs are also the people who typically must validate the correctness of the resulting requirements.
- **Architecture oversight.** Customer representatives often have oversight responsibilities with regard to system development. This typically includes

exercising due diligence in the oversight of the development of the system architecture and insuring its quality.

- **Accept the system.** Once the system is built, the customer representatives typically must accept the system based on its fulfillment of its associated requirements including architecture-related requirements.

If the people who provide requirements fail to meet their responsibilities with regard to architecture-related requirements, then the following *negative* consequences can be expected:

- **Provide requirements.** Being neither requirements engineers nor architects, it is not easy for these people to provide architecture-related requirements. In fact, they will rarely if ever think of requirements in that manner. Although quality requirements are typically quite important to them (especially if the requirements are obvious, such as many interoperability and performance requirements), they often have a difficult time making such requirements unambiguous, feasible, and quantitative with minimum acceptable thresholds. Also, not being architects or engineers, they often have no concept as to the ramifications of their requirements in terms of cost and schedule. Thus, they may confuse unreasonable desires with mandatory requirements. Similarly, they can misidentify unnecessary constraints as necessary requirements (see next bullet).
- **Provide architecture constraints.** Although there are many valid reasons for stakeholders to mandate architecture constraints, there are also many invalid reasons. For example, they may mandate an architecture decision because it is the only option that they are familiar with, thereby unnecessarily tying the architects' hands. Ultimately, there are three potential problems:
  - Missing valid architecture constraints,
  - Poorly provided valid constraints (e.g., incomplete or mistaken constraints), and
  - Inappropriately specified constraints.
- **Validate requirements.** Because people often make implicit assumptions and take certain things for granted, stakeholders will often not realize that important architecture-related requirements are missing or incompletely specified. It can take an experienced requirements engineer to elicit this information that a subject matter expert considers too obvious to mention.
- **Accept the system.** The system qualities that are important to the people who provide system requirements are most obvious when they are missing. Similarly, although they may have a very hard time during requirements elicitation telling requirements engineers just how much of different quality factors their system must have, they are often quite capable of pointing out to the development organization that the system is not good enough. Thus, a lack of well-engineered architecture-related requirements can easily lead to a system that technically meets its requirements but which is not acceptable to its stakeholders.

Requirements Engineers

Requirements engineers have the following responsibilities with regard to architecture-related requirements:

- **Identification.** Requirements engineers must identify all of the requirements that are relevant to the architecture. Given the natural emphasis on functional requirements and the requirements engineer's limited resources for performing requirements identification (e.g., elicitation, invention, and *derivation*), such requirements are all too often never identified. This is especially true of quality requirements, which tend to be missing or difficult to identify in customer or higher-level requirements documents. Thus, a key responsibility of requirements engineers is to derive the appropriate architecture-relevant requirements when engineering the requirements for individual subsystems of the system.
- **Analysis.** Requirements engineers must analyze requirements to ensure that they have the necessary characteristics of good requirements. Thus, requirements engineers are responsible for ensuring that the requirements are complete, consistent, feasible, mandatory, unambiguous, and so on.
- **Specification.** Requirements engineers must properly specify the architecture-relevant requirements so that the other stakeholders of these requirements can read them.
- **Verification.** Requirements engineers should verify the derivation of architecture-related requirements from higher-level goals, concepts of operations, requirements models, and requirements. They should also verify the consistency of these requirements with requirements conventions such as standards for requirements structures and contents.
- **Validation.** Requirements engineers should validate the architecture-related requirements with the stakeholders that are their sources.
- **Management.** Requirements engineers must store the requirements, provide appropriate metadata, maintain them under configuration control, and freeze them at the appropriate time so that architecture, design, and implementation can be completed and so that the current version of the system can be released.

If requirements engineers fail to meet their responsibilities with regard to architecture-related requirements, the following *negative* consequences can be expected:

- **Identification.** Important architecture-related requirements may not be identified. Missing requirements are hard to identify during requirements analysis, verification, and independent evaluation. If the requirements are totally missing in higher-level requirements specifications, specified in an incomplete or ambiguous manner, or only implied by other requirements, then it becomes the requirements engineer's task to attempt to derive explicit subsystem requirements from implicit system requirements – an inherently risk-laden job. If missing requirements fall through the cracks, the architect is unlikely to adequately incorporate them into the architecture and is unlikely to include them in engineering trade-offs between conflicting requirements. Too often, lower-level requirements engineers and architects end up guessing the requirements because they are given inadequate guidance.

- **Analysis.** All too often, the architecture-relevant requirements are inadequately analyzed and end up not having the characteristics of good requirements. For example, quality needs are often written as ambiguous, infeasible, and unverifiable goals such as “the system shall have high reliability” or “the system shall be safe” instead of as true engineering requirements.
- **Specification.** Many times, important quality requirements are specified in plans, process documents, or specialty engineering documents (e.g., reliability plans or safety policies) rather than in requirements specifications with the other requirements. This tends to prevent the architect from knowing about them and incorporating them into the architecture in a timely manner, resulting in significant architecture rework, increased development costs, and slipped schedules.
- **Verification.** Architecture-related requirements (especially quality requirements) are often difficult to trace because they do not map easily to other requirements or to architectural components. Because they tend to cut across many functional requirements and architectural components, the mapping is typically many-to-many, cluttering up the requirements traceability matrices and making them difficult to develop and maintain. Many specified quality requirements and other architecture-relevant requirements do not have the correct structure, ensuring that they are incomplete and therefore ambiguous and unverifiable. For example, quality requirements often do not specify the conditions under which they apply or more importantly, do not provide a minimum threshold on an appropriate unit of measure that specifies at which point the system would exhibit adequate quality.
- **Validation.** It is notoriously difficult to pin down the requirements stakeholders (e.g., customer and user representatives) as to just how much quality the architecture must have. Too often, they do not know or they want “wiggle room” to change their minds. The resulting “requirements” are often incomplete, infeasible, and untestable[1]. The architect must guess how good is good enough in order to complete the architecture and make engineering trade-offs, especially among conflicting architectural goals.
- **Management.** Too often, architecture-relevant requirements are not identified as such in the requirements repository, so that the architects have a difficult time identifying them among the huge number of requirements that do not significantly impact the architectures.

Given how often architecture-relevant requirements are either *not* engineered or else *inadequately* engineered, requirements engineering with respect to architecture-related requirements is often quite ineffective in practice.

[1] They may state that “the system shall be reliable,” but not say how reliable or in what way. They may state that “the system shall be safe and secure,” when it is impossible for any system to be totally safe or totally secure. Finally, such “requirements” are inherently not testable with a finite number of tests.

Requirements Evaluators

During the evaluation (i.e., quality engineering) of the requirements and their associated specifications, requirements evaluators have the following responsibilities with regard to architecture-related requirements:

- **Quality control.** Requirements evaluators must ensure that the architecture-relevant requirements, the repository in which they are stored, and the specifications that have been published all have adequate quality.
- **Quality assurance.** Requirements evaluators must ensure that the requirements engineering process used to identify, analyze, specify, verify, validate, and manage the architecture-related requirements has been properly followed and is effective in producing architecture-related requirements of adequate quality.

If the requirements evaluators fail to meet their responsibilities with regard to architecture-related requirements, the following *negative* consequences can be expected:

- **Quality control.** Missing and inadequately engineered architecture-related requirements will not be found and are therefore unlikely to be remediable in a timely and cost-effective manner.
- **Quality assurance.** Problems with the requirements engineering method that have allowed architecture-related requirements to be inadequately engineered have also not tended to be found in practice and therefore the problems continue.

Given how often that missing and inadequately engineered architecture-relevant requirements are not identified during requirements evaluations, quality control and assurance in this area is often quite ineffective in practice.

Architects

Architects have the following responsibilities with regard to architecturally-significant requirements:

- **Identification.** If architecture-relevant requirements (or at least quality requirements) have not been identified and tagged as such by the requirements engineers, then the architects must identify them so that they can drive the architects' decisions and properly influence the resulting architecture.
- **Allocate.** The architects must properly allocate (and trace) the architecture-relevant requirements to the associated architectural components (e.g., subsystems).
- **Analyze.** The architects must understand the ramifications of the architecture-relevant requirements on their architecture. They must make engineering trade-offs between the conflicting requirements in order to ensure a globally optimized architecture.[2]

[2] Locally optimizing architecture support for competing individual [types of] architecture-relevant requirements (e.g., interoperability, reliability, safety, and security) can result in a globally suboptimal architecture.

- **Incorporate.** The architects must develop an architecture that adequately supports the derived architecture-relevant requirements that have been allocated to their part of the overall architecture.
- **Verify.** The architects must verify that their architecture adequately supports its allocated architecture-relevant requirements.

The failure of others to meet their responsibilities upstream from architecture can contribute to the architects failing to meet their responsibilities with regard to architecture-related requirements, causing the following kinds of *negative* consequences:

- **Identification.** If the architects are unaware of all of the architecture-relevant requirements allocated to their architecture, then these requirements will not drive the architects' decisions and properly influence the resulting architecture.
- **Allocate.** Missing requirements will be ignored by the architects and will therefore not be properly allocated to the architectural components that should implement them.
- **Analyze.** Ambiguous requirements will be difficult for the architects to understand and analyze. Missing or inadequately specified architecture-relevant requirements will not properly influence engineering trade-offs between different potential architectures.
- **Incorporate.** When architecture-relevant requirements are missing or ambiguous, the architect's resulting architecture is unlikely to adequately support them.
- **Verify.** How can the architects be expected to verify that their architecture adequately supports its architecture-relevant requirements if the requirements have not been properly derived and allocated to appropriate architectural components?

Architecture Assessors

Architecture assessors have the following responsibilities with regard to architecturally-significant requirements:

- **Understanding.** The assessors are responsible for understanding the requirements that drive the architecture they are assessing. This includes understanding the ramifications the requirements have on the architecture, which often involves significant experience as an architect, as a subject matter expert in the specialty engineering area of the requirements (e.g., reliability, safety, or security), and in the application domain of the subsystem being assessed (e.g., avionics, chemical engineering, finance, or transportation).
- **Assessment.** The assessors are responsible for assessing the architecture's support for the architecture-related requirements that have been derived and allocated to the architecture they are assessing.

The failure of others to meet their responsibilities upstream from architecture can contribute to the architecture assessors failing to meet their responsibilities with regard to assessing the architecture, which can cause the following *negative* consequences:

- **Understanding.** The assessors are unlikely to know of and understand all of the architecture-related requirements that should have driven the architecture.
- **Assessment.** The assessors be not be able to make a judgment as to whether the architecture is adequate. The assessors cannot judge sufficient quality if they do not understand *how good* is *good enough* because the relevant requirements are either missing or inadequately engineered.
- **Finger pointing.** The architects can legitimately complain about a poor assessment grade due to an assessment of inadequate architecture quality because the requirements they are being assessed against were either not allocated to them (scope creep) or else the requirements were ambiguous (and therefore subjective). Subsystem requirements engineers can in turn point the finger at the requirements engineers of higher-level subsystems or the system requirements engineers and argue that higher-level requirements did not exist from which to derive the requirements. Ultimately, the top-level requirements engineers can argue that they did not receive any customer requirements from which to derive the architecture-related requirements, something that is especially common with performance-based contracts where general capabilities are required and the details (including system requirements) are left to the contractor. Finally, the customer can argue that the architecture-related requirements are implicit in the capabilities contracted for and that it is the development contractor’s responsibility to identify and derive them.

System Integrators

System integrators have the following responsibilities with regard to architecturally-significant requirements:

- **Subcontract requirements.** The system integrators must pass on the allocated architecture-related requirements to subcontractors or vendors who must supply the corresponding architectural components.
- **Subcontract oversight.** The system integrators must exercise subcontract oversight regarding further engineering of these requirements and their incorporation into the corresponding architectural components.

The failure of others to meet their upstream responsibilities can contribute to the system integrators failing to meet their responsibilities with regard to the architecture-related requirements, causing the following *negative* consequences:

- **Subcontract requirements.** If the architecture-related requirements are not passed on to the subcontractor or vendor, then the supplied architecture component may not meet the requirement and may therefore be unacceptable.
- **Subcontract oversight.** Naturally, the same problems associated with customer oversight apply here. In fact, poor subcontract oversight exacerbates the difficulty of customer oversight of the prime contractor.

Integration and System Testers

Testers have the following responsibilities with regard to architecturally-significant requirements:

- **Integration testing.** Testers must find defects that potentially interfere with the architectural components being successfully and incrementally integrated. This is made difficult if relevant architecture-related requirements (e.g., intraoperability) are not properly derived and allocated to the components being integrated.
- **System testing.** Testers must not only perform function testing based on the system's functional requirements; they must also test the architecture-related requirements, which are often considerably more difficult and expensive to test. For the quality requirements, this often calls for specialty-engineering tests such as reliability testing, safety testing, security (e.g., penetration) testing, stress testing (for capacity and scalability requirements), and usability testing.

The failure of others to meet their upstream responsibilities can contribute to the testers failing to meet their responsibilities with regard to the architecture-related requirements, causing the following negative consequences:

- **Integration and system testing.** Testers have a harder time developing integration and system tests based on architecture-related requirements without such properly engineered requirements.

4 CONCLUSION

Requirements that significantly impact the architecture of a system are very important to the quality and acceptability of the system. Some of the most important of these requirements are the quality requirements that specify a minimum level of quality that the system must exhibit. Unfortunately, there is a myth that quality requirements are for the most part unverifiable and testable, and this myth is a major reason why such architecture-related requirements are poorly engineered if at all. It is a mistake for requirements engineers to give up on engineering these requirements in order to avoid wasting precious resources by trying to do the impossible. It is this complacency that often prevents stakeholders and requirements engineers from learning how to properly engineer quality requirements. And the resulting missing requirements and poorly specified requirements have major consequences that negatively impact many important stakeholders. We recommend the reader to read Tom Gilb's latest book [Gilb 2005] that provides excellent guidance on how to unambiguously and quantitatively specify quality requirements.

REFERENCES

[Firesmith 2003a] Donald Firesmith, “Specifying Good Requirements,” *Journal of Object Technology (JOT)*, 2(4), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, pp. 77-87, July/August 2003. http://www.jot.fm/issues/issue_2003_07/column7

[Firesmith 2003b] Donald Firesmith: “Using Quality Models to Engineer Quality Requirements”, in *Journal of Object Technology*, vol. 2, no. 5, September-October 2003, pp. 67-75. http://www.jot.fm/issues/issue_2003_09/column6

[Gilb 2006] Tom Gilb, Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, Elsevier, 2005.

[Clements et al. 2002] Paul Clements, Rick Kazman, and Mark Klein, Evaluating Software Architectures: Methods and Case Studies, Addison Wesley, 2002.

[SEI 2005] Software Engineering Instititue, Quality Attribute Workshop, 2005. http://www.sei.cmu.edu/architecture/products_services/qaw.html

Disclaimers

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

The views and conclusions contained in this column are solely those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

About the authors

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 four years, he has developed the world's largest (1,100+ webpage), free, and open source informational website of reusable process engineering components. Based on the OPEN Process Framework (OPF), it is located at http://www.opfro.org. Currently writing a book on the engineering of safety and security-related requirements, he can be reached at dgf@sei.cmu.edu.

Dr. Peter Capell is a senior member of the technical staff at the SEI, where he works with Donald Firesmith supporting Government acquisitions. He is an Adjunct Faculty member of the Carnegie Mellon School of Computer Science. He is the author of publications related to software process improvement as well as intelligent tutoring systems. He is a past Director and past Education Chair of the Pittsburgh Chapter of IEEE, member of Sigma Xi, the International Visual Literacy Association (IVLA), and American Educational Research Association (AERA).