Quality Attributes for Mission Flight Software: A Reference for Architects

Jonathan Wilmot
NASA
Goddard Spaceflight Center
Greenbelt, MD 20771
301-286-2623
Jonathan.J.Wilmot@NASA.gov

Lorraine Fesq
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-7224
Lorraine.M.Fesq@jpl.nasa.gov

Dan Dvorak
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-1986
Daniel.L.Dvorak@jpl.nasa.gov

***Abstract*— In the international standards for architecture descriptions in systems and software engineering (ISO/IEC/IEEE 42010), "concern" is a primary concept that often manifests itself in relation to the quality attributes or "ilities" that a system is expected to exhibit — qualities such as reliability, security and modifiability. One of the main uses of an architecture description is to serve as a basis for analyzing how well the architecture achieves its quality attributes, and that requires architects to be as precise as possible about what they mean in claiming, for example, that an architecture supports "modifiability." This paper describes a table, generated by NASA's Software Architecture Review Board, which lists fourteen key quality attributes, identifies different important aspects of each quality attribute and considers each aspect in terms of requirements, rationale, evidence, and tactics to achieve the aspect. This quality attribute table is intended to serve as a guide to software architects, software developers, and software architecture reviewers in the domain of mission-critical real-time embedded systems, such as space mission flight software.**

TABLE OF CONTENTS

1. INTRODUCTION

In the process of architecting, developing and evaluating software architectures, the discussion of quality attributes comes up quickly. Questions arise about which attributes are being addressed in the architecture, what are the definitions of those attribute terms, and what is the evidence of those attributes in the architecture or implementation? While there is a significant body of work available on the topic of quality attributes, they tend to be broad, incomplete, or leave the terms open for interpretation. In order to use quality attributes as part of a software architecture assessment, NASA's Software Architecture Review Board (SARB) set out to create a more complete and objective list with defined metrics that could be used during SARB reviews. The result is a table where each attribute is formatted as a row with columns for descriptions, requirements, rationale, metrics and common approaches for how that attribute can be achieved in an architecture implementation. Table 1 shows example rows of the table.

2. BACKGROUND – PURPOSE AND DEVELOPMENT OF THE QA TABLE

Purpose

The Quality Attribute Table presented in this paper is intended to document a set of software architecture quality attributes that can be used within the domain of mission-critical, real-time, embedded systems. This is the primary domain of NASA's Software Architecture Review Board which focuses on astronautic and aeronautic systems. This paper provides background, rationale, and a description of how the QA Table could be applied. The QA Table has multiple intended purposes: as a guide for software architects, project teams, and implementers during development of an architecture, and as guide for project teams and reviewers to assess an architecture's suitability for a given mission(s). It is important to reiterate that the set of quality attributes in this table are the ones deemed most important in the domain of space mission flight software. Thus, readers who are more accustomed to enterprise software or web services, for example, may not see the attributes and aspects that are most important to their domain.

Background on the SARB

NASA's Software Architecture Review Board (SARB) was formed in 2009 following a recommendation from the final report of the Flight Software Complexity Study [1]. Its charter is to manage and/or reduce flight software complexity through better software architecture and to help improve mission software reliability. The SARB does that by providing constructive feedback to flight projects during the formative stages of software architecting, well before a project reaches its Preliminary Design Review (PDR). Depending on the needs and importance of a project, reviews have varied in duration from a couple teleconferences with verbal feedback to a two-day face-to-face meeting resulting in a documented board report. In preparation for a review, the board typically holds two-to-three brief discussions with the architect to obtain preliminary documentation, understand driving requirements, and decide where to focus attention during the review. Those discussions often center on software quality attributes of particular importance to spacecraft flight software. The QA table described in this paper serves as an important reference that the board shares with architects and uses during reviews. The QA table and other materials used in preparing for a review are maintained on the SARB Community of Practice page of the NASA Engineering Network website at https://nen.nasa.gov/web/software/sarb.

The Development of the QA Table

Development of this QA table began in late 2013 as part of the National Space Universal MOdular Architecture (SUMO) effort, shown in Figure 1, initiated by the Office of the Director of National Intelligence (ODNI) with a goal to "Reduce the cost of satellites while introducing modular concepts that can encourage innovation." [2] One of the tactics was to have a common software architecture supporting a competitive marketplace of software and hardware components. As part of the process, the SUMO software architecture team began evaluating existing software architectures currently in use at US agencies (e.g., NASA, DoD, NRO) along with those of several spacecraft vendors. Within a few weeks of starting these evaluations, it became clear that a list of quality attributes with consistent definitions and defined metrics was not available, at least within the domain of flight software. Initially the team gathered the attributes from architectures being evaluated as defined within the respective Architecture Description Documents (ADD). Work continued to merge and harmonize the list up until the SUMO effort was disbanded a few months later.

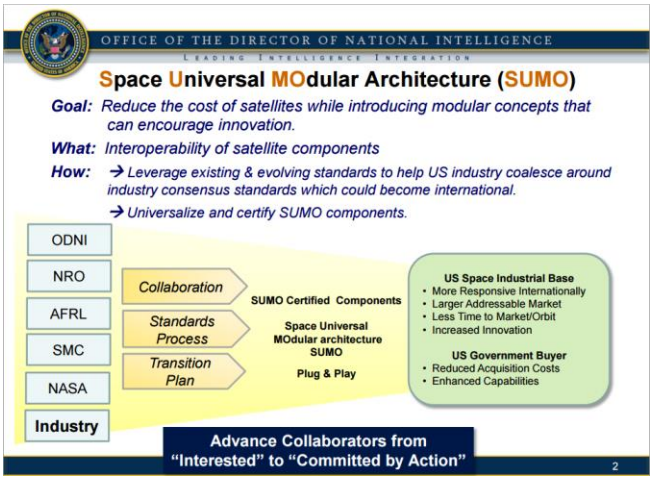

Figure 1 Overview of SUMO

In that relatively short time, the SUMO software team was successful in creating a draft QA table and had started using it as part of its architectural assessments. As some SUMO team members were also members of the SARB, it was proposed that the SARB should continue to mature the QA work. It is important to note that early ODNI sponsorship provided a level of access across US agencies and industry, as shown in Figure 1, that NASA's SARB could not have achieved on its own. This led to a broader and more relevant QA table, as each organization had different use cases and perspectives.

To continue the process of identifying relevant QAs for the Table, the authors reviewed papers, references and books [3, 4, 5, 6, 7, 8, 9] and collected a fairly comprehensive list of attributes. Some attributes, such as "Manageability", were considered outside the scope of embedded flight software (FSW), and were removed. Others were deemed similar to, or overlapping with other QAs, and were combined in the table (see Column B description). Once the list was completed, it was vetted and refined by the SARB. The SARB then worked through the process of how the table would be used, and identified the columns described in the next section. Members of the SARB selected QAs that were of the most interest to them, and filled in the rows of the table. These entries then were reviewed by the entire SARB team, and updated into the current version, posted on the SARB Community of Practice Website.

3. A DESCRIPTION OF THE QA TABLE

The QA Table is organized as a set of rows for the selected attributes with the columns in those rows specifying the associated descriptions, properties, and parameters. Each attribute has one or more "Aspect of" that provide a context for the remaining columns in that row. It became clear early on that without context to narrow the scope of a QA, it was extremely difficult to generate the text for the remainder of the row. For example, with the QA "Portability," questions arose: portability of what? Applications, services, architectural frameworks? It was only with a "Portability"

QA in context of "Operating Systems" that we could then specify the requirements, rationale, evidence and tactics to achieve application and framework portability across operating systems. Specifying context was seen as a key missing element with existing QA documents which tended to keep the attributes overly broad and unsuited for the SARB target domain.

The team started with the draft list developed by the SUMO architecture working group and then pulled additional attributes from: architecture documentation provided in previous SARB reviews, papers and books on software architecture, and information from the Internet. After much discussion on the many potential attributes, the SARB team arrived at fourteen key quality attributes for flight software: Portability, Interoperability, Modifiability, Performance, Availability, Reusability, Predictability, Usability, Scalability, Verifiability, Manage complexity, Security, Safety, and Openness. Many of these had related terms that were added to the description as "Also Known As" (AKA) terms. The AKA terms were viewed as being synonym of a QA, or as defining a subset of one of the fourteen QAs chosen and could be directly captured in the "Description" column or conceptually in the "Aspect of" column.

Column A: The Quality Attributes

The first column in each row is the quality attribute to be addressed. This column contains the chosen term indicating the non-functional requirement or property of the architecture to be implemented or reviewed. The term was selected through consensus by the SARB members, since different perspectives led to differing opinions as to which terms best fit the desired property.

Column B: Description of the QA and other terms used to describe the quality

Each Quality Attribute identified in Column A is defined in Column B to help the user understand what is meant by the term. For example, "Portability" is defined as "A design and implementation property of the architecture and applications supporting their use on systems other than the initial target system." Numerous references were used to define each QA, including Webster's dictionary, papers, journal articles and books [3, 4, 5, 6, 7, 8, 9].

For a number of QAs, multiple terms were identified as too similar to deserve a separate row in the table, so instead, the authors noted them as "AKA" synonyms of the primary QA. For example, the terms Adaptability, Upgradeability, Variability, Flexibility, Evolvability, Extensibility, and Extendibility are noted in Column B as synonyms of Modifiability.

Column C: Aspects of the QA

The term "Aspect of" is intended to define a context for the attribute. The "State/behavior" aspect of the QA "Predictability" can be rephrased as "the predictability of the state/behavior of the architecture." The QA "Portability" has numerous entries for "Aspect of" that help provide context; they allow the architect or evaluator to individually specify whether the application or system is portable across real-time/non-real-time implementations, across operating systems, across avionics platforms, or across any combination of those aspects.

Column D: Requirements

Column D contains sample requirements that the architecture must satisfy to claim support of a quality attribute. These requirements are verifiable statements, and are specific to each "Aspect of" row, as they need to be associated with a specific QA context. Unlike functional requirements, many of the QA requirements need to be confirmed by inspection or demonstration. For example, to claim the QA "Portability" with an "Aspect of" operating systems, it must be shown that the same application source code could be compiled and executed on two or more operating systems without modification to the application source code. This proof would be listed in Column F, the "Evidence of/verification" column. Also note that requirements may have a more subjective scale associated with them. To reuse the "Portability" example, if the architecture required just a slight application modification, that should rank higher in satisfying the QA than an architecture that required significant modification. The Requirements in column D are offered as examples that could be used by projects.

A	B	C	D	E	F	G	H	I
Attribute	Description with AKA terms bolded	Aspects of	Requirement	Rationale	Evidence of/verification	Tactic to achieve	Project specified: Project Prioritization	Project specified: Project intended variation
Portability	A design and implementation property of the architecture and applications supporting their use on systems other than the initial target system.	Real-time and non-real-time	The architecture shall support application execution in real-time (hard and soft) and non-real time environments	1) Supports both flight and test run-time environments and as well as deployments to potentially lower cost non-real-time systems.	Demonstrate execution on a real time flight target and a non real-time target with no changes to the application	Application logic is separated/abstracted from execution environment/framework	(NA, Low, Med, Hi, Priority is intended to allow trades when QAs come in into conflict.)	List intended targets. (non-real-time, soft real-time, hard real-time, Time-Triggered)
		Operating systems	The architecture shall support application execution on a range of operating systems without modification of the application	Operation system selection is a project choice and is typically based on cost, Quality of Service requirements, and target platform support (Board support package)	Demonstrate execution on multiple operating systems with no changes to the application (Automated tool driven changes may be considered)	Standards and abstractions. For example, 'segregate operating system calls in an abstraction layer; use multi-OS standards such as POSIX or ARINC 653; MBSE with multi-OS code generator'		
		Processor/platform	The architecture shall support application execution on a range of processors and platforms without modification of the application	Processors and platforms are typical variation points project to project. Enabling projects to select processors and platforms with minimal affects to applications allows for system optimization	Is the architecture Processor/Platform interface abstraction sufficient such that applications can be rehosted on another Processor/platform without modification (Additional points for the number of supported platforms)	Standards and abstractions. For example, these tactics could include 'segregate hardware interactions to a hardware abstraction layer; disallow use of platform-specific extensions to programming language; MBSE with multi-platform code generator; component library for standards-		
		Services	The architecture shall provide a common set of standard service interfaces	Services will not have to be recreated for each software instantiation. Application software will not have to implement service functions.	Is the list of common/standard services sufficient such that applications can be rehosted on another architecture instantiation without modification	Standardize and abstract interfaces to common services. Analyze services that are common to the system domain and ensure that the service interface abstraction hides variation points.		
		Middleware	The architecture shall isolate the application	Enables use of 3rd party middleware without vendor	Is the middleware abstraction sufficient to support the	Standards and abstractions		

Table 1 Snapshot of the QA table showing example of one quality attribute

Column E: Rationale

The "Rationale" column documents how each QA requirement adds value to an architecture for a project or projects. The team did not list all possible rationale, but focused on the one or two considered most important. For example, a project may choose to ensure that the architecture shall support application execution in real-time and non-real-time environments. The rationale for this is to allow the architecture to support both flight and test (e.g., desktop) run-time environments, which is described in the Rationale.

Column F: Evidence of/Verification

Column F is where the architect responds to Column D (Requirement); it is where the project provides evidence that the requirement *has* been verified, or how it *will* be verified. For example, one aspect of portability is OS portability, and the associated requirement (Column D) is: "The architecture shall support application execution on a range of operating systems without modification of the application." This requirement would be convincingly met if the project "demonstrates execution on multiple operating systems with no changes to the application," as stated in Column F.

Column G: Tactic to Achieve

A tactic is a design decision that influences the control of a quality attribute response [Bass et al, 2003]. Thus, Column G is where the project identifies design decisions to be used in meeting the requirements in Column D. Explicitly identifying such decisions enables experienced reviewers to challenge a decision if they feel the tactic is inadequate or insufficiently described. For example, in the aspect of Portability related to operating systems, the QA table provides "standards and abstractions" as general tactics that could be used to meet the Requirement. In a review, however, the project should spell out specific standards and abstractions.

Columns H-I: Project Prioritization and Project Intended Variation

Each row of the table has two columns for use by project software architects, implementers, and reviewers. "Project Prioritization" and "Project intended variation" are to be completed by project personnel in the very early stage of development concurrently with the system requirements. All QAs should be reviewed to decide/establish the priority of each (Not Applicable, Low, Medium or High) in Column H. For example, "Portability" may be High priority for a project creating a reusable software system meant to be instantiated by many users, whereas "Portability" would not be an important QA for a one-of-a-kind special software system intended for only one use. In addition, projects should specify any variations of a QA that are needed. For example, perhaps a project would like portability across only two operating systems. If both operating systems support POSIX, then the QA requirement could be met using POSIX as the choice for the "Standards and abstractions" tactic. These details should be captured in Column I. The intent of these two columns is to capture the intended QA goals of the system and have them consistently documented for early agreement by all stakeholders before the architecture and software development begins.

4. Use Cases

The QA Table has at least three primary use cases, as described in the following subsections. The first describes a Use Case from a software architect's and project team's perspective, where the table is used to evaluate and determine the priorities of each QA for a specific project. The second describes the use during software development,

and the third describes use in the review process to evaluate a software architecture with respect to each QA in the Table.

Architect/Project team Use Case

Quality attribute priority and variation points have a very significant impact on the architecture and should be used to directly inform the trades that must be performed and then reviewed by all stakeholders. If the architect intends the system to be a reusable application framework, then "Portability" would be documented as a high priority with the appropriate variation targets listed. However, a common tactic to achieve Portability is to add abstraction layers that can impede system performance. This conflict must be traded when implementing some of the "Tactics to achieve." In this case, the "Performance" QA would be rated lower in priority than the "Portability" QA.

This table is also intended to inform an architect and/or a project software team about why they should consider certain QAs in the architecture under development or being considered for a project. The "Rationale" for "Portability" across "Processors/platforms" has the potential to reduce the risk to a project if the processor needs to change due to performance or availability reasons, or if the project consists of several mission over a long period of time. These concerns may not have been considered, but are brought to light in the "Rationale". In this way, the "Rationale" has been used, and can be used by projects, to capture best practices.

Developer Use Case

Developers perform the task of implementing the architecture and need to be especially mindful of the "Tactic to achieve", "Evidence of/verification", "Project Prioritization", and "Project intended variation" columns for each attribute during the design and code process. Developers perform many of the detailed implementation trades, and provide the detailed evidence and verification products. For "Portability," these would include identifying specific standards that were used, and what middleware was selected or developed. During project reviews such as a Preliminary Design Review or Critical Design Review, the architect(s), project team, and stakeholders can review the current design, implementation, and trade documentation to ensure that the Quality Attributes are being instantiated as intended.

Reviewer Use Case

The SARB team engages projects early in the life-cycle, usually before a Preliminary Design Review. By the time the SARB holds a review, it has already interacted with the project software architect to identify driving requirements and associated quality attributes that pose the biggest challenge, or biggest risk if not satisfied. Thus, the first use of the table in a review is to examine the priorities shown in Column H (Project Prioritization) to see if they are in agreement with the formally described driving requirements. Those priorities should not all be "High." Architecting inevitably involves tradeoffs, so it may be necessary to sacrifice a "medium" or "low" QA in order to achieve a "high" QA. There should be a range of priorities so that reviewers can see how some tradeoffs will be made.

Reviewers will then use the QA table to probe into architectural details with respect to Column D (Requirement), Column F (Evidence of/Verification) and Column G (Tactic to achieve). In places where an ADD lacks convincing evidence (Column F), discussion in the review will reveal whether it is a weakness in the architecture or in its documentation.

5. FUTURE WORK

As a test run of the QA Table across two of the primary use cases, SARB members will use the table to assess existing software architecture(s). As a first step, the SARB will ask the original architect(s) to complete the "Project Prioritization" and "Project intended variation" columns as originally intended and then provide the "Evidence of/verification" information. The goals are threefold: (1) To validate text in the "Requirement", "Evidence of/verification", and "Tactic to achieve" columns; (2) To mature the document with additional tactics or types of evidence; and (3) To provide objective feedback to the architects on how well the original intents were satisfied.

6. SUMMARY

This paper describes a table of Quality Attributes that was developed by NASA's Software Architecture Review Board as an aid to flight missions. The QA Table is intended for use by flight software architects to help them consider and determine which attributes are important to their mission. This table serves as a reference for FSW architects to ensure that they have considered all relevant QAs. With the "Project specified" columns filled out by project teams and relevant stakeholders, this table then serves as a set of requirements and a guide for designers and implementers. Additionally, this table also can serve as an aid to flight software architect reviewers, allowing them to assess the architecture by examining the priorities that the FSW architect and project team have selected for a mission, as well as the trades that went into making these architectural decisions. Note that this QA table is expected to be a living document with additional "Aspects of", "Requirements", "Rationale", and other columns to be documented as software technology evolves.

The QA table is currently available for all NASA missions, and can be accessed on the NASA Engineering Network SARB Community of Practice Website at URL https://nen.nasa.gov/web/software/sarb . The authors have started the process for an open release of the QA table and expect a release in a few months.

7. ACKNOWLEDGEMENTS

The authors of this paper acknowledge the following persons who worked as a team to develop the QA table: Ken Costello, NASA Independent Verification and Validation Facility; Michael Madden, NASA Langley Research Center; Alex Murray, Jet Propulsion Laboratory, California Institute of Technology; Darrel Raines, NASA Johnson Space Center; John Weir, NASA Marshall Space Flight Center; Kathryn Weiss, Jet Propulsion Laboratory, California Institute of Technology; and Dr. William G. Antypas, Jr, Senior Scientist with CRL Technologies Inc. The authors also thank Michael Aguilar, NASA Technical Fellow for Software, for sponsoring the SARB and the work described in this paper, and Professor David Garlan, Carnegie Mellon University, for his thorough review and insightful comments and recommendations that improved the quality of our product.

Part of the research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] Dvorak, (2009), "NASA Study on Flight Software Complexity", [Online]. Available: http://www.nasa.gov/offices/oce/documents/FSWC_study.html

[2] Collins, B (2013), "Space Universal Modular Architecture (SUMO): CCSDS Spring Plenary" [Online]. Available: http://cwe.ccsds.org/sois/docs/SOIS-APP/Meeting%20Materials/2013/Spring/SUMO%20CCSDS%20Spring%20Plenary.pdf

[3] Len Bass, Paul Clements and Rick Kazman. Software Architecture in Practice, 2nd Edition, Chapter 5, Addison Wesley, 2003.

[4] Software Systems Engineering by Andrew P. Sage and James D. Palmer, 1990, Wiley Interscience

[5] Wikipedia (2015) "List of system quality attributes" [Online]. Available: https://en.wikipedia.org/wiki/List_of_system_quality_attributes

[6] Wikipedia (2015) "Non-functional requirement" [Online]. Available: https://en.wikipedia.org/wiki/Non-functional_requirement

[7] Len Bass, Paul Clements and Rick Kazman. Software Architecture in Practice, 2nd Edition, Chapter 4, Addison Wesley, 2003

[8] Merriam-Webster's Collegiate Dictionary (11th ed.). (2005). Springfield, MA: Merriam-Webster.

[9] Firesmith, Software Engineering Institute; QUASAR: A Method for the QUality Assessment of Software-Intensive System Architectures; CMU/SEI-2006-HB-001; July 2006.

BIOGRAPHIES

Jonathan Wilmot *is an aerospace Computer Engineer at NASA's Goddard Space Flight Center in the Flight Software Branch. He has over 30 years of aerospace software experience. Following several years developing software for commercial and defense avionics, he joined NASA in 1991 as a lead software engineer on the Small Explorer series of spacecraft. After varied roles on over 12 spacecraft, he now serves as a software systems architect with NASA's Core Flight System (cFS) reusable software framework project. He is also serves as a Deputy Software Technical Discipline Lead with the NASA Engineering & Safety Center, and as Chair of the Spacecraft Onboard Interface Services Working Group within Consultative Committee for Space Data Systems*

(CCSDS) international standards organization. He received his BS in Software Engineering at the University of Maryland College Park.

Lorraine Fesq *is a Principal Engineer in the Engineering Development Office at NASA's Jet Propulsion Laboratory. She has over 30 years of aerospace experience that spans industry, government and academia, has worked all mission phases of spacecraft development. She has received a NASA Public Service Medal for her work on the Chandra X-ray Observatory, and a NASA Exceptional Achievement Medal for advancing the Fault Management discipline within NASA. Lorraine taught in the Aeronautics/Astronautics department at MIT while researching model-based diagnostic techniques. She organized both of NASA's Fault Management Workshops, which brought together Fault Management practitioners and experts from NASA, DoD, industry, and academia to share insights and to expose and address systemic challenges. Lorraine led a NASA-wide assessment and advisory team to review the Constellation Program and to recommend improvements to the program's Fault Management plans, designs, and organizational structure. Lorraine is the Lead for NASA's FM Community of Practice, is the co-Lead for NASA's Software Architecture Review Board, and serves as a Deputy Software Technical Discipline Lead with the NASA Engineering & Safety Center. Lorraine received her B.A. in Mathematics from Rutgers University and her M.S. and Ph.D. in Computer Science from the University of California, Los Angeles.*

Daniel Dvorak *is a Principal Engineer in the Engineering Development Office at NASA's Jet Propulsion Laboratory, California Institute of Technology, where his interests span software architecture, autonomous control, fault management and model-based systems engineering. Dan co-leads the NASA Software Architecture Review Board, under which this quality attribute table was developed. Dan holds a BS in electrical engineering from Rose-Hulman Institute of Technology, an MS in computer engineering from Stanford University, and a Ph.D. in computer science from The University of Texas at Austin.*