Architecture Quality Assessment*
version 2.0

R. Hilliard M. Kurland S. Litvintchouk T. Rice S. Schwarm

August 7, 1996

Abstract

This document describes an Architecture Quality Assessment (AQA) instrument for the evaluation of system architectures. The AQA is intended to provide an objective and repeatable basis for assessing an architecture's quality in both generic and program-specific aspects. The AQA may be used to: evaluate a candidate architecture; review the technical progress of an on-going architecture development; assess a complete, delivered architecture prior to its acceptance and system implementation; or, compare two or more alternate architectures in a consistent fashion. This version is the product of a three month proof of concept phase and has been tested in limited trial use. Future versions will increase the depth and breadth of architectural coverage.

1 Introduction

1.1 Purpose

This document defines an Architecture Quality Assessment (AQA) for the evaluation of system architectures. The AQA is intended to provide an objective and repeatable basis for assessing an architecture's quality in both generic and program-specific aspects. The results of an AQA are meant to be a direct input to the decision maker (such as a PEO or PM) for architecture-related issues on major system developments. The assessment can then be used to highlight potential risks from which to formulate risk reduction strategies, or as a basis for other programmatic decisions relating to acceptance of the architecture, program schedule or cost adjustment, award fee or progress payment determinations. For example, the AQA may be used to:

- evaluate a candidate architecture;
- review the technical progress of an on-going architecture development;
- assess a complete, delivered architecture prior to its acceptance and system implementation; or,
- compare two or more alternate architectures in a consistent fashion.

Future: Add an appendix describing how to conduct each of these kinds of assessment.

The AQA consists of:

- a conceptual foundation, or frame of reference, for talking about architectures, architectural descriptions and architectural methods. This frame of reference is used to frame questions about the architecture under review;
- the identification of quality areas and factors which contribute to architectural quality;
- a set of measures, in the form of questions, to be used to assess an architecture against the identified quality factors;

- a methodology for applying the assessment in a repeatable fashion to address program-specific and general quality concerns;
- a technique for interpretation of assessment results and their correlation to architecture-related system and program risks.

1.2 Scope

The scope of the AQA is the architecture-related activity of a major system program, which includes the development and delivery of the *software system architecture*. Traditionally, the use of the term "architecture" or "system architecture" has been implicitly limited to the hardware and physical aspects of a system. We adopt the term *software system architecture* to include both the large-scale structure and behavior of the *whole* system – including those traditional architectural concerns pertaining to hardware and physical aspects of the system – as well as to emphasize the dominant role of software in today's large, complex systems, and the need to recognize that role during the architecture activity. But this term is in no way intended to limit the scope of architectural assessments to software issues.

We use the (vague) term *architecture activity* to reflect the fact that we do not assume any particular life cycle model, life cycle phases, or acquisition policies leading to those life cycle choices. The architecture activity takes place in a context determined by the system requirements and other programmatic goals and constraints; the resulting architectural description is the primary input to the overall system design and serves as the basis for system interoperability and evolution. It also effects other considerations, including development environment and approach.

The architecture activity may be carried out by the Government, or its agents (such as MITRE), by a Contractor, or by a joint team. The AQA is equally applicable in any of these cases, making no assumption as to who is involved in the architecture activity.

The AQA specifically excludes from its assessment other concerns and sources of risk not related to architecture. Excluded from this assessment are (among other topics): the overall system development process, the capabilities of the developers to carry out that process, the validity and relevance of requirements which led to the architecture, and the resulting system design and implementation.

In no way is the AQA designed to assess the capability of the architect or architectural organization. Nor is the AQA intended to be an evaluation of the developer's capability to produce the system. Other characteristics of the system, such as design quality and code quality are not intended to be addressed by this instrument but by other approaches. The AQA is not intended to be an evaluation of the developer, the processes used to develop the system, or the requirements specification.

For reasons of objectivity and repeatability, the AQA is narrowly focused on the artifacts of the architecture activity. This includes any *architectural documentation*, possibly including items such as simulations, models and analyses related to the architecture. At the same time, the AQA neither presumes nor prescribes that architectural information take any particular form – e.g., the AQA does not presume or prescribe the form or content of an architecture document "deliverable". It is anticipated that architectural documentation will take many forms. Similarly, we do not assume that such documents depict architectural descriptions with any regular or standard notations. At present, much of this information is likely to be informal, graphic or textual information. However, undocumented architectural information, such as that which might be learned from conversation or "hearsay" is not considered admissible information by this instrument. In the future, more formal means of capturing architectural expression may exist to facilitate these assessments.

Architecture development takes place against a background of program concerns arising from (among others): the mission and operational environment, user needs, and client (customer) expectations. To meaningfully assess architectural quality, these concerns must be taken into account within the evaluation and interpretation of AQA results. Prior to an AQA, an analysis establishes these concerns in a rigorous fashion, resulting in documented statements of Needs, Goals and Vision (see 1.4.1). Typically, these statements are "owned" by the client and serve as the baseline against which the architecture will be assessed. These statements will be used to identify, examine and give weight to system priorities consistent with client expectations within the AQA.

To make objective quality analysis possible, we have to distinguish the content and use of architectural information from other types of system information, such as requirements and design-level descriptions. There are no arbitrary boundaries here – they vary from program to program. Requirements often capture a mixture of user requirements and the needs of other stakeholders (e.g., maintainer's need to easily modify certain aspects of the system). System

architecture is the precursor of system design. Following the IEEE Architecture Working Group [8], we define the architecture of a system as:

> An architecture is the highest-level concept of a system in its environment.

An architecture addresses:

- The large-scale structure and behavior of the system
- Required external interfaces and their protocols
- Key decisions to meet program needs which can not be left to the developer, including:
  - Choice of key technologies and the rationale for their selection
  - Design commitments, such as the selection of an Open Systems Environment (OSE) profile
  - Degrees of freedom: the types of changes that must be anticipated
  - Legacy system integration strategy
- Basis for selected alternatives
- Identification of high risk implementation areas
- Guidance to the designers, implementors and maintainers for realizing and evolving the system
- Rendition of the system in a fashion suitable for client and other stakeholder's examination and input

System design is a further refinement of the system concept – providing implementation details to the components, their interfaces, and the connections between these components, while satisfying all governing architectural constraints. This activity of system design generates information to address:

- Traceability between system elements and requirements contained in a system specification or comparable specification vehicle
- Preliminary timing budgets for components that need to support end-to-end response time of a capability or function
- Sizing budgets of software allocated to processing platforms
- Identification of requirements that are cost drivers or are high risk
- Initial identification of the costs for building and maintaining the system, and
- Identification of an approach for testing the system

The intent of the AQA is to determine overall architectural quality and whether the architecture addresses program-specific concerns. The AQA provides a basis to identify architecture-related risks resulting from this assessment. An unfavorable assessment of the architecture is an indication that the evolving system may be considered to be at risk. Favorable ratings in all quality areas do not necessarily mean that the system design will satisfy system requirements, since system architecture and design are distinct activities – however, the likelihood of this is less than that of an unfavorable assessment.

1.3 About this Document

This document provides the complete methodology for applying the AQA. It includes all questions to be asked for an architecture, and the means for scoring and interpreting the results. The rationale for decisions made in the definition of the AQA is also provided.

The AQA is an evolving technique. This version is the product of a three month proof of concept phase. It has been subjected to limited testing through trial use. Future versions will provide increased depth and breadth of architectural coverage. Future areas for expansion are marked in this document by *Future*. Comments on this version and suggestions for future versions are encouraged and appreciated. (See Instructions for Comment Submission A.)

1.3.1 Acknowledgments

The authors which to thank Ed Green and John Maurer for their support, and especially Don Neuman for his encouragement and sponsorship of this project. David Emery provided useful, detailed comments on an earlier draft of the AQA. The authors also wish to acknowledge the contributions of the following individuals in the specialties noted: Richard W. Bentz Jr. (D062) and Jeffrey A. Clark (D062), Reliability and Availability; Charles J. Ludinsky (G04B), Communications; Donna L. Cuomo (D047) and Jane N. Mosier (G04E), Usability; David L. Baldauf (G021) and Marion C. Michaud (G021), Security; and Thomas M. Wheeler (D07S), Safety.

1.4 Methodology

This section describes the steps of an AQA, the methodology used for conducting one, the materials needed to perform that assessment, and the form of the resulting products.

The steps of a review are as follows:

1. Perform Analysis of Needs, Goals and Vision, when one is not available (see 1.4.1)
2. Gather relevant documents and other artifacts related to the architecture (see 1.4.2)
3. Evaluate documentation against measures and score results (see 1.4.3)
4. Interpret results and identify architecture-related risks (see 1.4.4)
5. Document results for the client (see 1.4.5)

The evaluation methodology used in the AQA follows that of the MITRE Software Quality Assessment Exercises (SQAE) [21]. The SQAE methodology identifies three levels of concerns: quality areas; quality factors and quality measures. We adopt that methodology here, without discussion.

We have identified six quality areas (see 2) which contribute to architectural quality. The quality areas are in turn addressed by eight quality factors (see 3). Each quality factor consists of a number of quality measures. Each measure takes the form of a question.

1.4.1 Analysis of Needs, Goals and Vision

> In referring to goals and visions, I have in mind a practical rather than a very principled distinction. As is usual in human affairs, it is the practical perspective that matters most. Such theoretical understanding as we have is far too thin to carry much weight.
>
> By visions, I mean the conception of a future society that animates what we actually do, a society in which a decent human being might want to live. By goals, I mean the choices and tasks that are within reach, that we will pursue one way or another guided by a vision that may be distant and hazy.
>
> – Noam Chomsky [4]

Architecture assessment as defined by the AQA takes place in the context of the *system- and program-specific concerns* for that architecture. For the purposes of this assessment, these concerns must be recorded in some consistent fashion, prior to the actual evaluation of architecture documentation. This section addresses the form of that statement of concerns, and the manner by which such a statement may be prepared, if a suitable one does not already exist.

For the purposes of the AQA, an understanding of system- and program-specific concerns are captured through the activity of *analysis of needs, goals and vision* (or "system of systems analysis" or simply "needs analysis"). Needs analysis produces three work products: a needs repository, a goals statement and a vision statement.

- The needs repository contains needs of the system. Needs may be thought of as distilled, architecture-relevant system requirements.
- The goals statement captures the client's success criteria.
- The vision statement reflects the client's long-term direction for system and its evolution.

Figure 1: Needs, Goals and Vision in Architecture Development

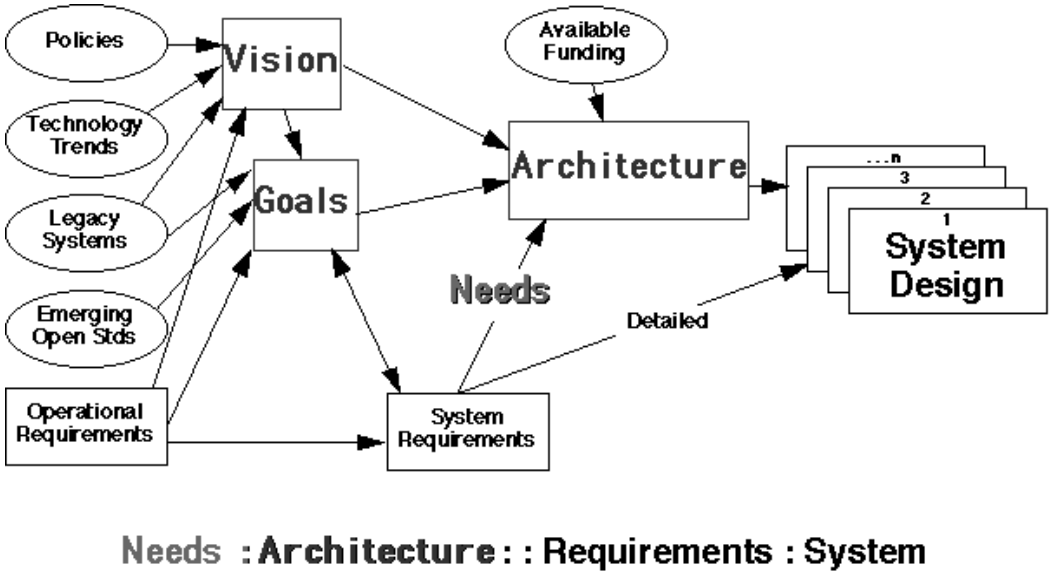

Figure 1.4.1 places these products in the context of a notional systems development. These work products flow from the client's expectations and environment and are typically articulated in collaboration with the architect. The client's environment encompasses the client's business environment (policies, legacy, and operational requirements – reflecting the mission) and the technical environment (trends and emerging standards) in which the system will be built.

The goals of the system flow from the vision statement and the needs flow from the goals. These relationships also imply a temporal precedence within the system's life cycle: needs must be addressed immediately, the vision is addressed over the long term (perhaps as much as 20 years out, but highly dependent on the life cycle of the architected system) and in the interim, the goals reflect success criteria (perhaps measured at incremental releases or other major milestones).

These work products form the foundation upon which the subject architecture will be judged, as they enumerate the client's expectations of the architecture to be assessed. Therefore, they are used to bound the problem space of the system for the purposes of the AQA. Since they are used to reflect client expectations, these products should be directly derived from/by the client. Ideally, the client "owns" these products – and often has prepared them independently of any assessment. At a minimum, in order for an AQA to be successful, the client is expected to "buy in" to these products as valid.

The vision statement requires thinking beyond the immediate objectives of the current users in acquiring and developing a new system. It should be linked to the client's long-range planning and speculation (for large systems, perhaps 10 to 20 years), to potential future users and new missions. It should not be restricted by predefined allocations of functions to organizations, systems, or persons.

The capabilities described in the vision are not prerequisites for a successful architecture; however, architectural flexibility sufficient to accommodate those capabilities is sought by the architect. The vision statement provides insight into new capabilities the system might need to provide in the future. The vision statement also provides insight into the potential future changes (advances) mission of the system, as well as anticipated technology advances. Examples of vision statements can be found in [10, 20]

The goals statement is intended to be representative of those technical and mission objectives that are relatively well defined and achievable in a predetermined time frame. The goals of a system should be consistent with – or even derived from – the vision statement. Goals are linked with the users short-range planning and are considered invariants within that time frame. Goals typically represent quantifiable, as well as qualifiable, success criteria for the program; often, they embody those concerns which will establish the *emergent properties* or "ilities" of a system and its architecture (see 3.5). A goals statement might address the behavioral characteristics of the system in terms of its

design. Goals might express where and how flexibility is desired, or where and what performance is desired. For an example goals statement, see [10].

The needs of a system define the problem space in a concise form. It is useful to think of the slogan:

Needs are to architecture as requirements are to a system.

Needs differ from requirements in several ways. Needs are more stable than requirements over the life cycle of the system and thus its architecture. Whereas detailed requirements may change as the users' day-to-day activities change, the need for that day-to-day activity does not. A need is not necessarily testable. A single need often represents the synthesis, or abstraction, of one or more individual requirements. A need captures those concerns that will drive key decisions by the architect, such as decisions pertaining to performance, technology or cost drivers. Examples of needs might include identification of a prescribed set of technical standards (a so-called "technical architecture"), user interaction timelines, distribution or policy needs.

The needs repository records the collected needs for the system in an orderly fashion. It captures the concerns of the client as well as all the stakeholders of the system. The needs repository also captures the environmental concerns of the stakeholders. These concerns may include security, development environment, legacy interaction or migration. The successful capture of the needs of program will be succinct. This is critical to keep the architect from getting bogged down in excessive detail. For example, for a system which may have as many as 5,000–10,000 requirements, it would not be unusual for the needs to number under 200.

For each need, the repository records a concise statement of that need, its source, its priority, and its classification based on quality measures or concerns. Needs originate from two types of sources. The first is any documentation that is available capturing system and operational requirements of the system. The second is the client or other contributing stakeholder with whom the need originates, such as users of the system. The prioritization communicates to the architect the relative importance of various needs when considering potential tradeoffs.

The accurate derivation of the needs for a program is critical to the definition of its architecture. A typical stage of needs analysis is to insure that the needs repository indeed "covers" the full set of requirements. Once an architecture is developed, all architectural elements may be traced back to architectural elements.

When the needs, goals and vision statements do not exist, they must be derived prior to an architecture assessment. These work products should be drafted, coordinated with the client, and authenticated by the client and key stakeholders.

1.4.2 Architecture-Related Documentation

The team performing the AQA needs to gather the relevant architectural documentation for assessment of the subject architecture. These materials may include written documents or other artifacts such as mathematical models, formal or executable specifications, or simulations. Ideally, the architectural documentation should be identified in concert with the architect, and should be focused on architectural information. There is no standard or required set of documents in which architectural information resides, so this step is crucial in obtaining appropriate information for the AQA. Relevant items include:

- Vision statement, outlining basic needs of users both today and projected into the future; and how architecture meets those needs today and is planned to meet those future needs as well.
- Goals statement, outlining the client's criteria for success.
- Needs repository, identifying the needs for each class of stakeholders. The needs should be prioritized, both as an absolute list across all classes of stakeholders, and as prioritized sublists of needs for each separate class of stakeholders.
- Identification and description of system stakeholders.
- Concept of operations for the system (CONOPS).
- Enterprise models of the projected operational environment for the system.
- Background information on application domain and any basic technologies to be employed by the architecture (e.g., COTS; special algorithms).

- Life cycle model for the system, including development, maintenance, and projected logistics support.
- Identification of the notations and representations that will be used to represent the architecture to all of its stakeholders. The description of the approaches used for development, maintenance and logistics support should specify in some detail the roles for the various classes of stakeholders who are involved in these activities.
- Description of the architectural method employed, including any special or unusual descriptive notations for the architecture.
- Descriptions or other models of the subject architecture, using the identified notations.
- Rationale for the architecture. Explanation of the choices made in the architecture, and the reasons behind those choices. Two ways (there are likely others) that adequate rationale can be provided for part or all of the architecture are: comparison against "straw man" architectural alternatives; analysis of how well the architecture performs in various scenario(s) of interest to the different classes of stakeholders.
- Suggested Figures of Merit (FOMs) for the architecture. These are metrics or measures of the architecture's effectiveness over time as perceived by the various classes of stakeholders. The FOMs should be selected for their ability to be assessed objectively (and, hopefully, quantitatively).

Once an set of architectural documentation is identified and agreed upon, the AQA team records the identification of this set, and begins its study of the architecture, reading and analyzing the collected documentation.

1.4.3 Evaluation and Scoring

When the AQA team is sufficiently familiar with the architectural documentation, the actual assessment can begin. This involves evaluating each of the measures below (3).

There are two types of questions in the AQA: *iterators* and *discretes*. Iterators are questions whose answers take the form of a list. The results of an iterator are then typically used to instantiate one or more other questions. For example:

> What stakeholders does the architecture technique identify? *(List.)*
> *For each stakeholder (listed above):*
> *Ask one or more questions regarding that stakeholder ...*

To make the scoring and interpretation of results repeatable, all questions (other than iterators) are *discretes* – responses to these questions range over a discrete set of values show in the table below.

Value	Description
IDEAL	Subject is explicitly addressed, and receives best possible treatment.
GOOD	Sound treatment of the subject relative to expectations.
MARGINAL	Treatment of subject meets minimal expectations.
UNACCEPTABLE	Treatment of subject does not meet minimal expectations.
INCOMPLETE	Treatment of subject exhibits a level of detail which is insufficient to make a judgment.
NON-APPLICABLE	This measure is not applicable to the subject of interest.

Table 1: AQA Scored Values

The results of individual questions will be aggregated in several ways to yield aggregate results for each factor, area, and to identify probable architecture-related risks (next section). In the aggregate – at the factor and area level – scoring will be:

ACCEPTABLE, UNACCEPTABLE, or INCOMPLETE. where ACCEPTABLE subsumes IDEAL, GOOD and MARGINAL and there are no NON-APPLICABLE results at the area and factor level. For analysis purposes, each of the discretes is coded as a numerical value. We provisionally adopt the scheme used by the SQAE, where IDEAL, GOOD, MARGINAL, and UNACCEPTABLE are coded as: 1, .75, .25, and 0, respectively. INCOMPLETE and NON-APPLICABLE are also coded as 0, but are tallied differently.

1.4.4 Interpretation of Results

Once all applicable questions have been answered, the results are may be weighted and aggregated to yield values for each factor, area, and risk. The matrices governing these aggregations are presented below.

The rules for aggregation and weighting will initially be hypothesized – as data is gathered from multiple AQAs, these rules will be tuned.

1.4.5 Reporting of Results

There are three primary products of the AQA review:

- Executive Summary
- Detailed Evaluation and Interpretation
- Open Issues and Questions

The *Executive Summary* provides an overview of the overall quality results and any outstanding issues, relative to the system's needs analysis. The *Detailed Evaluation* provides the scores at all levels of aggregation, the weightings used, and the identified risks. In some cases, clients want to take the results of the AQA back to the architect to provide feedback, or to highlight unresolved issues in the architecture. For this purpose, *Open Issues and Questions* are documented in a form that may be submitted to the architect for clarification.

1.5 Assumptions

This section documents assumptions made during the construction of the AQA which apply to its execution.

- The AQA evaluation methodology and question set **shall** be publicly known and openly available to the architects, and generally, to the organizations whose architectures are being evaluated. The vision and goals statements, and needs repository, against which the architecture is to be evaluated, **shall** likewise be available.
- All questions **shall** be answered from existing architectural documentation. Interviews, hearsay, or other undocumented forms of communication shall not be taken into consideration.
- The AQA **shall** be conducted by evaluators who are independent of the program being evaluated. There **shall** be a separate reporting chain for the AQA evaluator team to the architecture's client (or whoever commissioned the AQA) which is independent of project management.
- An AQA **shall** require minimal preparation on the part of personnel of the program, for Government, MITRE or Contractor personnel on the program; architecturally-relevant materials should be available via the ordinary Data Accession List (or equivalent).
- The evaluators **shall** be individuals with strong systems engineering experience, and familiarity with modern architectural principles and relevant system specialties.

1.6 Architecture Terminology

Because architectural methods and descriptions are in their infancy, there is a lack of standardized terminology and concepts applicable to architecture. Yet, for purposes of objective, repeatable evaluation, we need some uniform and neutral of talking about architectures to frame questions in the AQA. Therefore, for the purposes of architecture assessment, we adopt the terminology of an architecture *metamodel*, originally developed by MITRE for Army SBIS [9].

Future: Should a widely accepted terminological framework for architecture become available in the future (such as IEEE's *Recommended Practice for Architectural Description*), this AQA should consider its adoption.

This section outlines the basic concepts and terminology to be used in posing questions in the AQA.

1. An architecture is documented as an *architectural description*.
2. An architectural description consists of one or more *views*.
3. Each view presents the system from a well-defined vantage point to address a specific set of concerns.
4. Each view is documented in terms of a number of *architectural elements*.
5. An architectural element may be a component, connection or constraint.
6. A *component* depicts a major element of a view.
7. A *connection* depicts a relationship between components.
8. A *constraint* depicts depicts a condition/law which an element must satisfy/obey. A constraint acts upon a component or connection.
9. A *feature* is a computational element of a component; e.g., a function or port.[1]
10. An *interface* specifies a component of a system. An interface should specify both the features a component provides to other components or to the environment, as well as those features the component requires from other components or from the environment. [This is extremely important for embedded systems–we usually specify how components respond to requests, rarely do we specify what they do to other components or to the environment–side effects–in the course of satisfying those requests.]
11. Interfaces may either be *safe interfaces* (stateless) or be *communication interfaces* (have state).
12. A component is *shared safely* among other components if either it is safe, or all these other components are directly connected to it.
13. A connection between interface features is *correct* iff the implied transfer of information is consistent with all constraints on the features.
14. *Connectability* of interface features is an easily checked property of their features necessary for correct connection.
15. An architectural *commitment* is a "propert[y] of the system design which [detailed] designers are not at liberty to change." [3]
16. An architectural *obligation* is an "aspect[] of a design to which no commitment is made ... [is] the subject of obligations that lower levels of a design must address." [3]

2 Architecture Quality Areas

We have identified six quality areas related to architecture and their interrelationships.

[1] The definitions of "feature" and those related to interfaces, sharing and connectability are based on RAPIDE [17].

2.1 Understandability

An architecture is useful only insofar as it is expressible to others. Architecture understandability is a prerequisite to each of the other quality areas – if an architecture can not be understood, it can not be readily implemented, maintained or evolved. We assess understandability in terms of the conceptual foundations underlying the architecture, the techniques used to create it, the description used to record and analyze it and the qualities of that resulting description. Understandability is largely generic rather than program specific.

2.2 Feasibility

The foremost question to be asked about an architecture pertains to its *feasibility* – Does the architecture provide a sufficient basis upon which to develop a system which is an instance of that architecture and satisfies program needs. Feasibility is a prerequisite to the system's implementability, maintainability and evolvability. If the architecture does not exhibit feasibility, the system may not be readily or economically implemented and user or client needs may not be met. In the AQA, feasibility is restricted to the architecture – the AQA does not address the feasibility (capability) of the development team to implement the architecture, or the feasibility of its approach.

2.3 Openness

An architecture exhibits *openness* if it describes a system which can operate in an open system environment. An open system is built to exploit public interfaces and services, exhibiting minimal dependence on proprietary systems or APIs. Such interfaces allow smooth upgrade of system components conforming to those interfaces, increased portability to new platforms, and minimal machine dependencies.

Note: Openness also contributes to simplicity because standardly defined interfaces routinely have well-defined behaviors.

2.4 Evolvability

An architecture is *evolvable* if it exhibits a degree of changeability to meet new user or client needs, and may be adapted to new, unanticipated missions, while preserving the integrity of the original architecture. Evolvability is measured against the vision statement. Prerequisites: Feasibility and Maintainability.

2.5 Maintainability

Once a system is built, an architecture has implications for the *maintainability* of that system. One ingredient of that maintainability is the maintainability of the architecture itself.

2.6 Client Satisfaction

The quality areas above (Understandability, Feasibility, Openness, Evolvability and Maintainability) are general characteristics pertaining to the quality of a system's architecture. For a given system, they may be assessed independently of that system's mission, requirements, etc.

However, a key ingredient of the quality of an architecture must be its ability to yield a system which meets its requirements in the context of its mission. These concerns are (necessarily) project- and system-specific. The purpose of this area, *Client Satisfaction*, is to provide a point for the collection of these project- and system-specific concerns.

As with the other quality areas, Client Satisfaction is analyzed through a number of contributing factors. To do so, we must "tune" the contributing factors to project- and system-specific concerns. This requires input from the program. We do this in the form of a needs analysis, , prior to AQA. The needs analysis determines overarching needs, goals and vision for the system. The needs analysis is then used to:

- Determine relevance and applicability of measures (specific questions) to the system under assessment.
- Determine thresholds and prioritization of measures and factors to the system under assessment.
- Inventory system-specific items for assessment.

3 Quality Factors

This section identifies the *quality factors* which contribute to the quality areas of the previous section. For each factor, a brief description is presented, followed by a set of *measures*, used to evaluate the factor. The correlation of factors to areas is described below (see 4).

3.1 Conceptual Foundations

This factor addresses whether the architecture is being developed against a well-defined set of concepts, or *conceptual foundation*. The purpose of this factor is to determine the well-foundedness and intellectual accessibility of this frame of reference – not necessarily its appropriateness for this system. It investigates, for example, whether terms and concepts pertaining to software systems architecture, like "architecture" and related concepts are well-defined. For a discussion of relevant terminology and concepts, see [12].

Note: This section is somewhat abstract by itself. An example here would be helpful to clarify how we are using the term "conceptual foundations". Consider the Air Force's CCPDS-R: TRW developed NAS (Network Architecture Services), an infrastructure framework to support the development of message-based command and control systems. The NAS conceptual framework established a set of architectural constructs: sockets, circuits, processes, tasks, etc., for the architect to use. Underlying this, were the theoretical foundations based on Ada's tasking model. Other humorous examples of what happens when there is a mismatch between foundations and the desired architecture: such as a certain program's Object View consisting of data and functions; and Danny Cohen's "global clock" for SDI – which happily ignored relativistic effects in a distributed system.

Key Entities: conceptual foundations, terms and concepts, intended stakeholders

1. What, if any, conceptual foundations underlie the development of the subject architecture? (*List.*)
2. Are the conceptual foundations identified and documented?
3. Do the conceptual foundations include a definition of "architecture"?
4. Are relevant terms and concepts identified and defined as a part of the conceptual foundations?
5. Are the definitions of relevant terms and concepts self-contained so that they are comprehensible and accessible to others? (Published, etc.)
6. Do the conceptual foundations provide, at a minimum, analogues for each of the terms defined above (see 1.6)? (*Map subject terms and concepts to those of the metamodel.)*
7. Is the purpose of an architecture well-defined within the conceptual foundations?
8. Is the scope of an architecture well-defined within the conceptual foundations?
9. Are the audiences and intended stakeholders for an architecture clearly identified within the conceptual foundations? *(List the intended stakeholders and audiences.)*
10. *For each audience and stakeholder*: Do the conceptual foundations identify those needs which the architecture is intended to address?
11. Are the conceptual foundations well-founded? I.e., is there a cited theory, practice or other discipline upon which the architectural foundations are based?
12. Is there a reference model depicting relation of architecture to the rest of the system?
13. Are the foundations documented? Are they expressed such that they may be objectively analyzed?
14. Do the conceptual foundations support any formal model of system architecture such that resulting architectural descriptions may be reasoned about or analyzed?
15. Are the conceptual foundations appropriate for the domain of application of the subject system?

16. Are the conceptual foundations appropriate for the class of systems of which this is an instance? (real-time, embedded, distributed, dynamic, legacy, MIS, ...)
17. Is there a documented rationale for the conceptual foundations?
18. Are there precedents for use of these foundations for related or similar systems?
19. Were alternative foundations considered?
20. Is there a demonstrable commitment to the foundations from the architect's organization?
21. Have the implications and ramification of the foundations been analyzed? Have trade-offs been considered?

3.2 Architecture Documentation

This factor addresses the quality of the *documentation set* (including actual documents and any other artifacts, such as simulations, mathematical models, executable prototypes, etc.) of the architecture, relative to the conception of that architecture, and anticipated usage of the documentation by the client, system developers, and other stakeholders. The intent of this factor is to establish that a conception of the architecture is reflected in actual documentation in a usable form.

Key Entities: Architectural documentation

1. Is the architecture documented?
2. What architecture documentation is available? (*List.*)
3. Is the architecture documentation consistent with the conceptual foundations?
4. Does the architecture documentation meet the purpose, scope of architecture as defined by the conceptual foundations?
5. Is it feasible to identify the relevance of the documentation for the various stakeholders?
6. Is the architecture documentation accessible to relevant audiences and stakeholders?
7. Does it clearly identify architectural assumptions and decisions?
8. Is the architecture expressed in a manner which is appropriate to its audiences?
9. Is the format of documentation appropriate to the size and scope of the architecture?
10. Is the architecture documentation available in a machine-readable format?
11. Does documentation provide an overview of the architecture?
12. Does documentation provide guidance to its contents and to other relevant documentation?

3.3 Architecture Description Techniques

This factor deals with the quality of means (notations and conventions) by which architectural information is expressed within the architecture documentation including any notations for representing architectural constructs.

Note: This section draws upon the SEI's Descriptive framework for Architecture Description Languages [14] and Hilliard's comments on an early draft of same.

Future: Revisit this factor to consider understandability vs. expressivity, elegance vs. obscurity. At present, the best score would go to a highly expressive, unconstrained notation – e.g., English.

Key Entities: Architectural elements (i.e., components, connections, constraints, and views), notations

1. How are architectural elements represented notationally? (*List.*)
2. What architectural elements are identified for capturing architectural information? (*List.*)

3. Is the set of architectural elements well-defined?
4. Is the syntax of architectural elements well-defined?
5. Is the semantics of architectural elements well-defined?
6. Are the rules for representing architectural elements documented?
7. Does technique support meaningful naming?
8. Does the technique address naming of architectural elements, the form and scope of names?
9. Are architectural elements typed?
10. Can architectural element types be defined by the user (architect)?
11. Can architectural elements be associated with additional information? (such as function, realization)
12. Are there guides or handbooks for architects and other audiences describing the architectural elements and their usage?
13. Does the technique permit the expression of various architectural styles? (e.g., pipe-and-filter, event-driven, client-server, blackboard)
14. Does the technique place limits on the styles it may be used to express?
15. Does technique allow the expression of various architectural topologies? (e.g., layered, object-oriented, message-passing, transactional, rule-based)
16. Does technique place limits on the expression of architectural topologies? (such as hierarchy, DAG, bag)
17. Can technique be used to capture other, known architectural concepts?
18. Does technique provide mechanisms which permit the expression of deferred architectural decisions?
19. Does architectural technique have provisions for inter-view cross references?
20. Does architectural technique have provisions for presentation control? (such as text vs. graphics, report generation)
21. Does architectural technique facilitate ease of change of architectural descriptions?
22. Does technique have constructs to effectively manage architectural complexity (e.g., views, packages or modules)?
23. Is the architecture technique ergonomically engineered for human?
24. Is technique well-known?
25. Does technique support means to distinguish architectural information from design?
26. Does the architectural technique allow expression of commitments?
27. Does the architectural technique allow expression of obligations?

3.4 Analyzability

This factor addresses whether the architecture description, as developed using the notations and techniques of the previous section, may be analyzed for consistency, completeness and other characteristics. It further addresses whether these analyses may be supported by procedures or automated tools. Analyzability contributes to: (1) the quality of the architectural documentation, insofar as analysis improves its consistency and completeness; and (2) the quality of the resulting system, to the extent that analysis of the architecture enables the architect to anticipate future system properties.

Key Entities: analysis techniques, analysis tools

1. What does one need to know to analyze an architecture, given the conceptual foundations and their theoretical underpinnings?
2. What kinds of analyses are defined by the method to be applied to an architecture description? (*List.*)
3. Is each applicable or relevant emergent property (3.5) addressed by at least one analysis technique?
4. What additional kinds of analyses may be applied to an architecture description? (*List.*)
5. Does representation have a machine-readable, storable form?

For each analysis technique listed above:

1. Does the technique support "mechanical" (systematic) checking of architectural descriptions?
2. Is there a formal model underlying this checking?
3. Does the technique allow analysis of dependencies between elements?
4. Does the technique allow analysis of the behavioral consequences architectural constraints?
5. Does the technique allow analysis of the behavioral consequences of architectural decisions?
6. Using the technique, can quantitative measures (performance, accuracy, precision, ...) be associated with architectural elements?
7. Does the technique allow for aggregating quantitative measures and developing budgets of same? (such as resource utilization and "flow")
8. How well does the technique "play with" other analysis techniques?

3.4.1 Accountability

Completeness Analysis.

1. Does the technique define one or more criteria for completeness of the architectural description?
2. Are there pre-defined rules for checking completeness?
3. Are there provisions for the user (i.e., architect, or other stakeholder) to define additional rules for completeness checking and perform them?

Consistency Analysis.

1. Does the technique define one or more criteria for the consistency of an architectural description?
2. Are there pre-defined rules for checking consistency of the architectural description?
3. Are there provisions for the user (i.e., architect, or other stakeholder) to define additional rules for consistency checking?

Extra-Architectural Links. This section measures "traceability" between architectural documentation and other artifacts.
Future: Add a picture to clarify the notional document types and their relationships: needs repository, goals and vision statements, handbook of architectural technique, architectural documentation, architectural description, requirements specifications, etc.

1. Does the architectural technique have provisions for recording the history of and changes to the architectural description?
2. Does the architectural technique have provisions for capturing assumptions made in the architectural description?
3. Does the architectural technique have provisions for tracing architectural decisions to needs?
4. Does the architectural technique define how architectural elements may be linked to needs?
5. Does the architectural technique define criteria of consistency between architectural descriptions and information in other documents?
6. Are there rules for checking consistency between architectural description and other documents?

3.4.2 Conformance

1. Does architecture as documented conform to the conceptual foundation's definition of architecture?
2. Does the architectural description allow the resulting design to be tested or verified for conformance?
3. Are there precedents where this architecture has led to a conforming implementation?
4. Do conceptual foundations provide a way for an architecture to address operational, technical and system concerns?

Future: Conformance to architectural frameworks or guidelines (such as DoD's Joint Technical Architecture [6] or the X/Open Architectural Framework [22].

3.4.3 Tool Support

1. Are there tools to support each of the analysis techniques listed above?
2. Are there tools to support the creation of architectural models?
3. What tools support the creation, maintenance and analysis of architectural models? (*List.*)

3.5 Emergent Properties

Complex systems often are meant to exhibit desired properties or characteristics which can not be localized to specific components or connections. Such properties are called *emergent*. Many of the desirable properties of engineered systems are emergent, rather than local – typically the "ilities" cited in requirements or other sources. To achieve these properties in the delivered system, that system's architecture must take some cognizance of these properties.

Not every architecture needs to give equal weight to all of these properties – their relevance or weighting for a given architecture will vary with stakeholder needs and priorities. This section serves as a checklist of these properties, and provides a set of measures for each – to relate their coverage in the architectural documentation against stakeholders' concerns.

1. Based on the needs, goals and vision statements for the system, what are the critical emergent properties which should be addressed by the target architecture? (*List.*)
2. Is each critical emergent property addressed by a suitable analysis techniques?

3.5.1 Security

Security aspects of a software system's architecture entail consideration of three distinct areas: *confidentiality* which assumes that information is not disclosed to unauthorized entities or processes, *integrity* those characteristics which ensure that resources operate correctly and data in the system is accurate, and the *denial of service* which is the result of any action or series of actions that prevent any part of a telecommunication or automated information systems from functioning [1].

Security becomes a concern when classified or unclassified but sensitive data is to be processed in the target system. Other concerns which may need to be addressed by the assessment include whether or not the architecture addresses multiple clearances and additional access restrictions predicated on categories and compartments. If security is a concern then the following questions apply.

1. Does the architecture identify each expected user type and their clearances?
2. Does the architecture identify the components comprising the trusted computing base (e.g., servers, PCs, routers, gateways, cryptographic support, operating systems, database management systems)?
3. Does the architecture address the classification levels of the data that is to be handled by each element?
4. Does the architecture identify the environment where the elements will reside? (i.e., whether or not components and connections will be contained in a protected environment)
5. Does the architecture identify how each user type will access the system (e.g., protocols such as FTP, Telnet) and the potential vulnerabilities arising from this access?
6. Does the architecture identify the type of access (C,R,U,D) that will be needed for each user type?
7. Does the architecture address the external systems and classification levels of the data to be exchanged with each system?
8. Does the architecture address the type of mechanisms that will be used to exchange data with the external systems (communication protocols such as Internet, FTP)?
9. Is the architecture capable of being expanded to accommodate future security goals? (e.g., the ORD may indicate that the threshold is System High and the objective is to support a multi-level secure mode of operation)
10. If encryption is needed for data flowing through unprotected environment, are the components and connections that will support encryption identified?
11. If encryption is needed, is the encryption type identified in the architecture? (E.g., Class I, Classified information or Class II, Sensitive but Unclassified)
12. If encryption is a need, does the architecture address key management?
13. Does the architecture identify the critical data to which integrity mechanisms will be applied?
14. Does the architecture identify the components and connections that will support the implementation of data integrity?

3.5.2 Dependability

Dependability is defined to be the "quality of service provided by a particular system". Dependability includes reliability, availability, and safety. The following definitions are assumed in the remainder of this section [19]:

1. *Availability* is defined to be the conditional probability that a system is operating correctly (without failure) and is available to perform its functions at an instant of time.
2. *Reliability* is defined to be conditional probability that a system performs correctly (without failure) throughout an interval of time given that the system was performing correctly at the beginning of the interval of time.

3. *Failure* is the deviation of the delivered service from the specified service where the service specification is an agreed description of the expected service [2].

4. *Safety* is the freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property or damage to the environment [7].

5. *Hazard* is a state or set of conditions of a system that, together with other conditions in the environment of the system, will lead inevitably to an accident [16].

Reliability and Availability. During the architectural phase of system development, a number of strategies will be examined for meeting reliability and availability needs. These alternative strategies and the resulting trade-offs should be captured in the architecture documentation.

In most circumstances, systems will evidence either a reliability need (e.g., for critical missions such as aircraft flight control) or an availability need (usually associated with a command and control-like missions). Seldom will it be necessary to evaluate an architecture against both the reliability and availability needs.

There is a strong relationship between mechanisms that support fault tolerance and these two emergent properties. Therefore, the AQA has includes measures to address the identification of single point failures and components that support fault tolerance.

1. Does the architecture identify component numbers and types (e.g., PCs, workstations, mainframes, bridges, routers, gateways, operating systems, data stores)?

2. Is there an initial prediction of availability for each of the major architecture alternatives?

3. Does the architecture address the critical failures and their relationship to the components and connections?

4. Does the architecture address the critical failures and the resultant impacts on the architecture?

5. Does the architecture identify single points of failure and redundancies?

6. Is there an initial prediction of reliability for the each of the primary architecture alternatives?

7. Does the architecture identify how the system will handle off-nominal conditions (e.g., loads that exceed the stated need)?

Safety. For those systems where there is some risk with respect to loss of life, injury, environmental harm, or loss of equipment, the architecture should address safety.

1. Does the architecture identify the components and connections that address system safety?

2. Does the architecture documentation identify the hazards to be addressed by these elements?

3. Are there any hazards that have not been addressed by the architecture documentation?

4. Does the architecture introduce new hazards? Have they been recognized and addressed?

3.5.3 Usability

For interactive systems, the means by which the various end users of the system will interact with the system should be addressed by the architecture. *Usability* is the "effort required to learn, operate, prepare input, and interpret output of the system" [18].

Future: Usability is one important case of operability, for which the stakeholders are the end users of the system, other cases will be dealt with in future versions.

1. Does the architecture address a user interface concept?

2. Does the architecture enforce a common "look and feel" that will help ensure that there is a consistent user interface for appropriate classes of users?

3. Does the architecture identify the elements (e.g., toolkits, special input/output devices) that support the user interface?

4. Does the architecture support the usability goals of client?

5. Does the architecture support customization and extension of the user interface by users and others?

3.5.4 Interoperability

Interoperability may be defined as [13]:

1. The ability of systems, units or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together, or

2. The condition achieved among communications-electronics systems or items of communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them and/or their users.

1. Does the architecture identify the external systems that are to interoperate with the subject system?

2. Does the architecture identify the protocols that will be used to support the interoperations between the subject system and the external systems?

3. Does the architecture identify the data elements (and their attributes) that are to be exchanged between the subject system and the external systems?

4. Does the architecture identify the services that are to be provided by the external systems in order to support the operations at the subject system?

5. Does the architecture identify the services that are to be provided by the subject system in order to support the operations at the various external systems?

6. Does the architecture identify any behavioral or other constraints on interoperability of data and services?

3.5.5 Physical Characteristics

Physical characteristics which include component footprint (the dimensions of a component), weight, and power (e.g., the need for UPS, the cycles per second) constraints have become less of an architectural driver. Technological advances in electronics (e.g., miniaturization, less power consumption) have led to the decreased emphasis in power weight, and footprint as architectural drivers.
Future: This section will be the subject of further elaboration as needed for specific systems. It is expected that for most systems only a subset of the following questions will be applicable.

1. Does the architecture address weight needs?

2. Does the architecture address footprint needs?

3. Does the architecture address set up and tear down needs?

4. Does the architecture address power needs?

5. Does the architecture address environmental and climatic concerns (such as temperature, humidity, and sand)?

3.5.6 Survivability (Future)

Survivability is the ability of the system to provide continuous and adequate performance of critical services and functions even after a successful attack. An attack may be a result of nuclear weapons effect, jamming, terrorism, conventional weapons, terrorism, and information warfare.

3.5.7 Performance Characteristics

Performance characteristics which include mean response time, average throughput and the variance of both of these characteristics as seen by the users, will often influence the architecture. To support the analysis of the architecture with respect to performance characteristics, the following definitions are assumed:

1. A *hard deadline* is time constraint which cannot be violated without a critical system failure.
2. A *soft deadline* can tolerate some violation of a time constraint without causing a critical failure.

The following measures are used to assess whether the architecture has potential risks in meeting performance needs.

1. Does the architecture documentation identify the hard and soft deadlines and loading constraints?
2. Does the architecture documentation identify the key external events that are related to the soft and hard deadlines?
3. Does the architecture identify the components that are involved in scheduling common resources (e.g., processes, tasks, network bandwidth)?
4. Does the architecture identify the temporal relationship between the components and the hard and soft deadlines?
5. Does the architecture documentation identify the scheduling strategies that will be used to meet constraints?
6. Does the architecture address priority inversions?
7. Does the architecture identify the granularity for the clocks that will support scheduling?
8. For distributed systems that requires synchronized clocks, does the architecture address the synchronization of clocks?

3.5.8 Modes of Operation (Future)

A number of target systems may have the need to support two or more modes of operation and different states.

1. Does the architecture address the modes of operations?
2. Does the architecture address the transitions between states?

3.5.9 Affordability (Future)

Affordability is the condition that the lifecycle cost associated with the system is within the expected costs of the client.

1. Does the architecture contain sufficient information in order to perform a preliminary cost for acquiring and maintaining the selected system?
2. Is the preliminary cost for acquiring the system and maintaining the system within the client's budget?
3. Does the architecture documentation contain sufficient information with respect to alternative architectures in order to obtain preliminary estimates of the costs associated with these alternative systems?

3.5.10 Supportability (Future)

Supportability is the ability for the architecture to support the client's logistical needs. For AQA, this assessment must be considered to be preliminary and will be qualitative.

3.6 Patterns and Idioms

This factor assesses the quality of the architecture with respect to its identification and use of organizing principles such as computational models, programming and design idioms, and "patterns". The use of such principles demonstrates: (1) awareness of system concerns; (2) intent to address them; and, (3) provision of solutions compatible with other constraints. This factor contributes to understandability and feasibility.

A *pattern* is defined as a solution to a particular system development problem, in a particular context. For example, many embedded systems require hard deadline scheduling of tasks. Possible architectural patterns applicable to this problem include periodic (cyclic executive) scheduling and rate monotonic scheduling. Ideally, a pattern should be capable of being defined and described separately from any system in which it is employed. One could attempt to "reverse engineer" such patterns out of actual architectures in which they have been used, but this is both inefficient and possibly inaccurate – since a pattern may need to be tailored for some specific aspects of a particular architecture.

Key Entities: a description of a pattern may include all of the following [5]:

1. The *Name* of the pattern.
2. The *Problem* the pattern is trying to solve.
3. The particular *Context* in which the pattern solves a problem. For example, the use of an enterprise-wide data model probably makes sense in a problem context where distributed data management is a concern – the architecture for an air-to-air missile is probably not an appropriate context for this pattern.
4. *Forces*, or *Tradeoffs*, which explain how the use of the pattern trades off among various properties of the architecture in order to improve some of its properties, possibly over others. Force is the explanation of the potential tension among certain potential properties; use of the pattern represents a conscious tradeoff in this tension. For example, the use of a "cyclic executive" approach to hard deadline scheduling emphasizes simplicity and prior expertise, over Evolvability and openness of the resulting architecture. Alternative scheduling policies might make different tradeoffs.
5. The *Solution* describes the structure and behavior of the result, or how to build that result.
6. *Examples* and *Visual Analogies*: These help explain the pattern.
7. *Force Resolution* or *Resulting Context*: This explains what forces (issues and properties) the pattern leaves *unresolved*, and what other patterns might be applied to resolve these remaining issues. If not resolved by the architecture, these may become obligations to designers. For the example of an enterprise-wide data model, issues left unresolved include resolution of "impedance mismatches" among components that support data models different from the enterprise-wide model. Patterns like the use of "wrappers" or "translators" can reconcile such different models with the enterprise-wide model.
8. The *Rationale* provides a justification for the pattern. A justification can be theoretical (e.g., the mathematical theory of rate monotonic scheduling), or practical (e.g., prior case studies in which the pattern was successfully employed).

Programming or design idioms may be thought of as one type of pattern Solution that is expressed in a particular programming language or design language. A pattern may *characterize* a programming or design idiom, along with descriptions of how, when or why to use that particular idiom [15].

We analogize architectural idioms as representing the Solution parts of corresponding architectural patterns. Note that whether a particular pattern should be considered as "architectural" or as "programming/design" is not always clear-cut, and that this distinction may shift over time. As programming languages improved, programmers became increasingly capable of dealing with many patterns and idioms formerly considered "architectural" (e.g., those dealing with multitasking). Unfortunately, we currently lack appropriate, robust formalisms for defining architectural idioms. Thus for the present, most architectural idioms will still need to be represented in an ad hoc manner, using colloquial English, mathematics, diagrams, etc., as appropriate to the particular pattern.

For the reasons just discussed the assessment of architecture-related patterns is less "cut-and-dried" than other factors in this assessment. The steps in patterns assessment are as follows:

1. Identify the key system problems that the architecture must solve. (*List.*)

2. Discover and inventory the predominant architectural patterns, and associate the patterns to system problems. (*List patterns and their associated problems.*)

3. Assess the discovered patterns with respect to the pattern attributes given above.

The concept of "architectural pattern" is related to the concept of emergent or non-local properties of a system (see 3.5). As noted above, a pattern solves an architectural problem in a way that may improve certain properties of the system even if other properties are unimproved or even made worse.

To evaluate each pattern used in a particular architecture, one would start by attempting to identify all of the above attributes for that pattern. Inability to do so might detract from the credibility of that pattern. Having identified the attributes, the next step would be to evaluate them. For example, how credible is the Rationale for each pattern?

3.6.1 Inventory of Pattern Categories

Architectural patterns may be organized into several broad categories. Two important categories of patterns are those that deal with issues of "execution time" (including events and behavioral abstractions) and "space" (including decentralization and distribution). A third category of patterns attempts to trade off such time-oriented properties against such space-oriented properties. Patterns of information management and communications (key problems in C3I) may be found in any of these three categories, depending on the particular "forces" and tradeoffs the patterns are intended to deal with. Other useful patterns may attempt to deal with any of the emergent properties enumerated (see 3.5).

Future: This is an open set, which we have only begun to populate.

- Note that there are other aspects of temporality besides execution time. For example, each architecture, each system, and each of their components and technologies has a finite lifetime. While issues of the lifetime and evolution of the architecture will be considered as part of this overall effort, defining appropriate "patterns of evolution" will be left as a subject for future research.

- Add category to address legacy-system related concerns.

- Incorporate Maier's principles for systems-of-systems (collective phenomena).

Patterns of Communication. If needs dictate that processing will be allocated to two or more processing elements to be interconnected by some type of communications media, then the architecture documentation should address the communication entities that comprise part of the target system architecture. *Communication entities* are defined to be the components and connections that provide for the control and direction for the information flows of the subject systems. These communication entities may be partitioned into two distinct categories, entities which have been dictated to the system architect by the Needs or entities that will be selected by the system architect.

1. Is the informational flow (information processing flows based on what the system is supposed to do) depicted in the architecture documentation?

2. Does the architecture identify the components that are involved in controlling, facilitating, or directing the information?

3. Does the architecture reflect the communication entities identified in the Needs?

4. Does the architecture identify the constraints for the communication entities (e.g., bandwidth, reliability, cost, and distances)

5. Does the architecture identify if the communication entities have the capacity to handle the expected information flow?

6. Does the architecture indicate if the communication entities can interoperate with one another??

7. Does the architecture indicate if non-standard entities (non-COTS) being used to support communications?

8. Does the architecture documentation identify the communication infrastructures where the system will be deployed? (In some cases, some systems may be deployed overseas and may make use of the host country's communication infrastructure. There is a possibility that the infrastructure may not be adequate to support the expected information flow)

9. Has sufficient alternative communication paths (distinct sets of communication entities) been identified in the architecture to assure that the architecture has adequately addressed the threat (e.g., HF communication and alternatives)?

10. Does the architecture address system control network management (e.g., mechanisms to handle changes in information flow control)?

Patterns of Execution Time. Patterns of execution time may include solutions for problems of:

- Coordination among architectural components
- Synchronization with respect to one or more time bases (clocks)
- Granularity of time
- Reaction to significant events and the processing needed to cope with such events
- Identification of which deadlines are "hard" vs. "soft"
- Dealing with "hard" deadlines
- Dealing with "soft" deadlines
- Identification of relative priorities for processing
- Identification of bounded delays vs. unbounded delays
- Development of adequate schedules for dispatching of tasks, processes, etc.

Patterns of Space. Patterns of space may involve solutions for problems of:

- Satisfying the decentralization requirements (e.g., redundant "hot standby") and distribution requirements (e.g., defense system with geographically distributed weapon sites for "area defense")
- Tradeoffs among centralized control vs. decentralized control
- If distributed processing is **not** a requirement (as it often is!), then considerations of when to employ distributed processing (as a design choice) and when not to
- Identifying how the information of the system is to be modeled and managed (e.g., ERA model, relational, object-based, object-oriented)
- If more than one such information model is to be used in the system, then identifying how "impedance mismatches" among these models are to be resolved
- If different information models are to be used over the lifetime of the system, then identifying how migration from one model to another is to take place. (e.g., migrating from Ada 83 to Ada 95; migrating from relational database management to object-oriented database management)
- Characterizing the information content of the system, and partitioning the information content among the components of the system
- Identifying which elements (data and/or processes) must be kept consistent (coordinated) across the system
- Identifying which elements must be replicated (for improved performance or reliability)

- Identifying which components are producers vs. consumers of data
- Identifying which components are reactive vs. pro-active
- Identifying which aspects of information flow are "pull" and which are "push"

Patterns of Space vs. Execution Time. These patterns may involve solutions for problems of:

- The tradeoff between consistency of data (or coordination of processing) and the need to achieve high performance. (For example, it may be necessary to tolerate some inconsistency in order to achieve adequate performance [11].)
- How information is to be *routed* among different processing elements in the system. This may involve tradeoffs among bandwidth, real time performance, reliability, etc.

3.6.2 Quality of Patterns

1. Is there a patterns solution to each critical stakeholder need?
2. Is the expressed architectural style appropriate to the present system?
3. Is the expressed topology appropriate to the present system?
4. Is architectural technique suitable for descriptions of the size and complexity of the system?
5. Which views does pattern apply to?
6. Are pattern's Context and Forces accountable to needs?
7. Are choices of patterns made consistent with needs for architecture and other pattern choices.
8. Check pattern's force resolution relative to completeness.
9. Are patterns expressed in a form appropriate to designers (e.g., design rules)?

3.7 Quality of Description

This section provides measures for the assessment of particular elements of the architectural description. Quality of description includes: simplicity, descriptive consistency, and descriptive completeness which relate to the characteristics of the resulting architectural description – not the system itself.

Key Entities: concept/term, view, component, connection, constraint, pattern

3.7.1 General

1. Does architecture identify architectural decisions which have been made (commitments)?
2. Does architecture identify architectural decisions which have yet to be made? (open issues)
3. Does architecture identify decisions which have been deferred to designers (obligations)?
4. Is the rationale for architectural decisions provided? Were trade-offs, alternatives considered and documented?
5. Is each architecture-relevant addressed by the architecture description?
6. Have consistency checks been performed and recorded for architectural description against other documents?
7. Are there redundant elements?

3.7.2 Views

1. What views are evident in the architecture description? (*List.*)

For each view:

1. Does the view represent a well-defined purpose and an easily stated viewpoint?
2. Does the view identify the major concerns it is intended to address?
3. Does the view identify the major stakeholder classes it is intended to address?
4. Does the view address all concerns of the stakeholder classes it is intended to address?
5. Does the view elements of a few related element types?

3.7.3 Between Views

1. Are components between views linked to show relevant relations (e.g., identity, part-whole or other implementation relations)?
2. Do components exist in different views with the same names?
3. Are identically named components in a defined relationship with each other (redundant, coincidental, linked)?

3.7.4 Components

For each component:

1. Does the component have a well-defined purpose, and identified role within the present view?
2. Does component include the expression of the decisions that it embodies?
3. Does component have an associated rationale for decisions it embodies?
4. Is the component expressed at an appropriate level of granularity, relative to the concerns addressed by the view?
5. Is the component expressed at an appropriate level of granularity to support the patterns it plays a role in?
6. Is the component expressed at an appropriate level of granularity to meet stakeholders' stated needs?
7. Are those components which must be kept coordinated (or synchronized) clearly identified?
8. Are those components which must be replicated clearly identified, and a rationale provided for each such replication?
9. Are those components which can operate concurrently clearly identified?
10. Are those components that may be shared safely among other components clearly identified? (See section 1.6.)
11. Does each component clearly identify the features it uses from its own interface? (**Note**: A component must not use features or functionality that are not included in its own interface. On the other hand, a component is not obligated to use every "required" feature in its interface. However, if it does not than the interface may be over-specified.)
12. Is component substitution possible based on interface conformance? (Note: If the interfaces are specified properly, than the functionality of the architecture should not be affected by such a change. However, some emergent properties (cf. 3.5) might be affected by a such a change.)
13. Are component side effects clearly identified? (See section 1.6 for a working definition of "side effect".)

3.7.5 Interfaces

1. Does the component have one or more defined interfaces? (*List.*)

For each interface:

1. Does interface include an expression of its roles?
2. Does the interface clearly specify the features it provides to other components and/or to the environment?
3. Does the interface clearly specify the features it requires from other components and/or from the environment?
4. Are those interfaces that are "safe" (stateless) clearly identified? (See section 1.6.)
5. Which other components (internal or environmental) might be affected by a change to this interface (as changes to either **provides** or **requires**)?

3.7.6 Connections

For each connection:

1. Does the connection have a well-defined role within the present view?
2. Is the protocol provided by the connection specified?
3. Is the type of the connection specified?
4. Does the connection identify the interfaces (of components) which it uses or assumes?
5. Does the connection include the expression of the decisions that it embodies?
6. Does the connection have an associated rationale for decisions it embodies?
7. Is the connection expressed at an appropriate level of granularity, relative to the concerns addressed by the view?
8. Is the connection expressed at an appropriate level of granularity to support the patterns it plays a role in?
9. Is the connection expressed at an appropriate level of granularity to meet stakeholders' stated needs?
10. Are those connections which must be kept coordinated (or synchronized) clearly identified?
11. Are behavioral (e.g., concurrency) and other constraints on the connection clearly identified?
12. Is connection substitution possible based on protocol conformance?
13. Are any connection side effects clearly identified?
14. Is it clear whether each connection between interfaces is correct?

3.7.7 Constraints

1. Are constraints identified within the view?

For each constraint:

1. Is the constraint specified in sufficient detail for the view?
2. Is the constraint source identified?
3. Are the architectural elements governed by the constraint identified?

3.8 Management Context

This factor addresses measures in the larger context in which architecture development is carried out. This includes life cycle issues, and the procedures and methods used to develop and maintain architectural documentation and descriptions. Whereas the previous factors introduced above are blatantly product-oriented, the present factor, has a process aspect to it. The intent of this factor is to assess whether the architecture under assessment has been developed using some architectural method which is well-defined and documented. The reason for this is not for the sake of the process itself, but to ascertain that the method provides a basis for improving the resulting product.

Key entities: Architectural methods and procedures

1. Is architecture being developed using a defined method for specifying architecture?
2. Is the architecture method documented?
3. Is documentation stable?
4. Is documentation mature?
5. Is there a well-defined review procedure for architectural documentation and descriptions?
6. Is there a well-defined architecture maintenance procedure?
7. Is the architecture documentation under version management and control?
8. Is architecture documentation under Quality Assurance control?
9. Is the method consistent with the conceptual foundations?
10. Is there a relationship between the architecture activity and other life cycle processes?
11. Are there well-defined relationships between Architecture activity and other familiar processes such as system planning, acquisition and engineering activities?
12. Are the relationships between architectural document and other products (requirements analyses, system specifications, system design document, software requirements specifications, trade studies) explicitly stated and are they well-defined?
13. Is there a clear separation of concerns between issues addressed by architecture and those of other phases?
14. Is the architectural description consistent with the documented architectural method?
15. Is configuration management defined for the architecture documentation?
16. Is granularity of configuration management adequate to control changes to views and elements?
17. Is the architecture description stable?

4 Relation of Factors to Quality Areas

Future: This section will describe the contribution of each of the quality factors to each of the quality areas, summarized in table . The particular contributions and weighting of factors to areas will be partially determined by client's needs statement. **Note**: For the proof of concept phase, the weightings of each group of measures in the factors above will be the same (1).

Factors x Areas	Understandability	Feasibility	Openness	Evolvability	Maintainability	Client Satisfaction
Conceptual Foundations	1	1	1	1	1	1
Architecture Documentation	1	1	1	1	1	1
Architecture Description Techniques	1	1	1	1	1	1
Analyzability	1	1	1	1	1	1
Emergent Properties	1	1	1	1	1	1
Patterns and Idioms	1	1	1	1	1	1
Quality of Description	1	1	1	1	1	1
Management Context	1	1	1	1	1	1

Table 2: Contributions of Factors to Areas

5 Architecture-Related Risks

Future: This section will delineate architecture-related risks to a program. Risks will be organized around each of the quality areas identified above. Symptoms of those risks will be derived based upon the quality factors.

Examples of architecture-related risks (organized by quality areas):

- Understandability
  - Architecture documentation does not effectively communicate to client or other stakeholders.
  - Architecture documentation does not effectively communicate design basis to designers and implementors.
- Feasibility
  - Architecture documentation does not provide sufficient information to design and implement system.
  - Architecture has not addressed key system modes of operation or mission needs.
- Openness
  - System cannot readily interoperate with external systems.
  - Upgrades and element replacement is not possible.
- Evolvability
  - System is not flexible for new missions.
  - System is not readily enhanced for new functionality.
  - System cannot be upgraded to new technologies.
- Maintainability
  - Architectural principles not sufficiently articulated that they may be maintained during modifications to the system.
- Client Satisfaction
  - Architecture development is not integrated with overall program concerns.
  - System cannot meet required or desired capabilities.

References

[1] *Air Force Systems Security Memorandum Command, Control, Communications, and Computer Systems Security Glossary*, January 1993.

[2] Algirdas Avizienis. Dependable computing: From concepts to design diversity. *Proceedings of the IEEE*, 74(5), May 1986.

[3] A. Burns and M. Lister. A framework for building dependable systems. *The Computer Journal*, 34(2), 1991.

[4] Noam A. Chomsky. *Powers and Prospects: Reflections on Human Nature and the Social Order*. South End Press, 1996.

[5] Jim Coplien. Software design patterns: Common questions and answers.

[6] Joint technical architecture. http://www.itsi.disa.mil/jta, 1996.

[7] *Military Standard System Safety Program Requirements*, January 1993.

[8] Walter J. Ellis, Richard F. Hilliard, Peter T. Poon, David Rayford, Thomas F. Saunders, Basil Sherlund, and Ronald L. Wade. Toward a recommended practice for architectural description. In *Proceedings of 2nd IEEE International Conference on Engineering of Complex Computer Systems, Montreal, Quebec, Canada, October 21–25, 1996*, 1996.

[9] David E. Emery and Richard F. Hilliard. "Architecture," methods and open issues. In D. Garlan, editor, *Proceedings of the First International Workshop on Architectures for Software Systems*, Seattle, WA, 1995. Published as CMU–CS–TR–95–151.

[10] David E. Emery, Richard F. Hilliard, and Timothy B. Rice. Experiences applying a practical architectural method. In Alfred Strohmeier, editor, *Reliable Software Technologies – Ada-Europe '96*, number 1088 in Lecture Notes in Computer Science. Springer, 1996.

[11] The epsilon-serializability (ESR) home page, 1996.

[12] Richard F. Hilliard. Dimensions of architectural thinking. MII, 1995.

[13] *DoD Dictionary of Military and Associated Terms*, joint publication 1-02 edition, March 1994.

[14] Paul Kogut and Paul Clements. Features of architecture representation languages. http://www.stars.reston.unisysgsg.com/arch/sei-feature-model-draft.ps, 1995.

[15] Doug Lea. Patterns-discussion faq.

[16] Nancy G. Levenson. *Software Safety and Computers*. Addison-Wesley, 1995.

[17] David C. Luckham, James Vera, and Sigurd Meldal. Three concepts of system architecture. Technical Report CSL-TR-95-674, Stanford University, July 1995.

[18] J. McCall, P. Richards, and G. Walters. Factors in software quality. Technical report, ????, 1977.

[19] D. K. Pradhan. *Fault Tolerant Computer System Design*. Prentice Hall, 1996.

[20] Thomas F. Saunders, Barry Horowitz, and Matt L. Mleziva. New process for acquiring software architecture. Technical Report M 92B0000126, The MITRE Corporation, 1992.

[21] Software quality assessment methodology. http://avatar.mitre.org:8000/Top_Page.html.

[22] X/open architectural framework. http://www.xopen.org/public/arch/, 1995.

A Instructions for Comment Submission

Comments on the Architecture Quality Assessment (AQA) should be sent to Rich Hilliard (rh@mitre.org). Comments should use the following format:

!topic Title summarizing comment
!reference AQA(version)-section.subsection(paragraph)
!from Author Name
!keywords keywords related to topic
!discussion
text of discussion

where *version* is the version ID of the present document, *section* is the section number (1 .. 5), likewise for *subsection*, and *paragraph* is the paragraph number within the lowest level heading. The **!keywords** line is optional.

Multiple comments per e-mail message are acceptable. Please use a descriptive Subject line in your e-mail message.

When correcting typographical errors or making minor wording suggestions, please put the correction directly as the topic of the comment; use square brackets [ ] to indicate text to be omitted and curly braces to indicate text to be added, and provide enough context to make the nature of the suggestion self-evident or put additional information in the body of the comment, for example:

!topic [c]Character
!topic it[']s meaning is not defined