Attributes Guided System Architecture Assessment

George Elias
Stevens Institute of Technology
gelias@stevens.edu

Rashmi Jain, PhD
Stevens Institute of Technology
Rashmi.Jain@stevens.edu

Abstract: A system's architecture profoundly influences the cost and success of a system throughout the entire lifecycle. As a result, System Architecture (SA) assessment has become a field of importance among systems architecting researchers and practitioners. An assessment of a system architecture is performed to determine the architectures attributes such as completeness, integrity, consistency, usability, and compliance. The assessment results can be used to drive changes in the architecture that would increase stakeholder acceptance, decrease defects, cost, and schedule throughout the entire lifecycle, and enhance system performance and quality. Most relevant assessment methods focus on system attributes. The key to success in this approach is to understand which system attributes should be addressed and how they relate to the architecture. This paper discusses relevant system architecture assessment methods and attributes.

Why System Architecture?

System architecting is an important job of systems engineers and architects, which influences all the later phases of the systems engineering process. System architecture is key in a system's success, as well as the dominant cost driver. Normally, an architecture would be frozen during the first phase of the project where much of the project cost is already committed. Knowing as much as possible in the early stages of design would enable effective decision-making and allocation of resources, thus reducing risk. Unfortunately, SA issues are often most evident during system integration. Consideration of key integration attributes early in the development cycle has impact on integration complexity (Jain 2008). Additionally, fixing architecture issues during system integration is far more costly than in beginning phases of design. Using System Architecture Assessment in the early system design phases is essential for the development of successful systems.

System Architecture is a logical construct for defining and controlling the interfaces and the integration of all the components of the system (Zachman 1987). A system architecture is classically comprised of a set of functions or module-subsystems; which are intended to cooperate for achieving the required functionality. Systems are typically (or ideally) developed considering a set of requirements and constraints, which satisfy the objective(s) of various stakeholders. Although systems have varying requirements and constraints there are common stakeholder concerns such as system appropriateness, integration capability, purpose, environment, functional capability and external interfaces (Concerns Survey). There are four major perspectives (4Ps) that influence SA quality: Product (Architecture), Process (Architecture Creation), People (Stakeholders), and Project Characteristics (e.g. cost, schedule, quality attributes) (Balci 2008). Systems architects need to address these concerns as well as others in their architectures.

During the development of a system architecture, many decisions are non-trivial because of competing and conflicting stakeholder requirements and constraints (Jain et. al. 2009). Once an

architecture baseline is established, architecture design elements are increasingly difficult to reverse because of the inherent investment in the previous architecture design. Architecture changes may affect many or all of the system functions and/or components. For this reason, effective architecture assessment is a cost and schedule saving technique.

Assessing Architectures

System architectures are assessed to determine the level of system attributes such as completeness, integrity, consistency, usability, and compliance. The assessment results can be used to drive changes in the architecture that would increase stakeholder acceptance, decrease defects, costs and schedule throughout the entire lifecycle, and enhance system performance and quality. Alternatively, architecture assessment can be used to trade-off competing system architectures.

Architecture Assessment Methods

Currently, there are various assessment techniques that focus on different aspects of the system architecture. We have included in Table 1 a selected set of common methods currently available that use attributes as a basis for guiding and analysing system architecture:

Table 1: Summary of System Architecture Assessment Methods

Method	Technique	Focus
Architecture Trade-off Analysis Method (ATAM)	Survey	Quality Attributes
Cost Benefit Analysis Method (CBAM)	Survey	Cost
Software Architecture Analysis Method (SAAM)	Scenario	Quality Attributes

Architecture Trade-off Analysis Method (ATAM) is a scenario-based architecture method for assessing quality attributes such as: modifiability, portability, extensibility, and integrate ability. ATAM is focused on how well the architecture satisfies particular quality goals. Besides the assessment of quality attributes which are also in SAAM, ATAM explores the quality attributes interaction and their interdependencies (Kazman 2000).

Cost Benefit Analysis Method (CBAM) is an extension version of ATAM. CBAM is an architecture centric method for analyzing the costs, benefits and schedule of the architecture. This method provides a means for measuring the economic aspects of architectural design, such as Return on Investment. Since CBAM is built on architecture assessment methods SAAM and ATAM, the method inherits their benefits (Nord 2004).

Software Architecture Analysis Method (SAAM) is the first universally accepted scenario-based software architecture analysis method. It was created to assess the architectures' modifiability. SAAM assesses the various qualities of software architectures like modifiability, flexibility, maintainability, scalability, extensibility, and functionality. The scenario generation process is based on the stakeholders' inputs. There is no guidance for the measurement of the architectural attributes that are to be assessed. SAAM is a stepwise and mature method for performing architecture assessment (Kazman 1994).

In addition to the system architecture assessment methods listed in Table 1, which all originate

from the software domain. Traditional systems engineering uses various techniques for architecture assessment. Three of these methods are: objectives hierarchy (Buede 2000), QFD (Quality Functional Deployment) (i Six Sigma 2005), and Pugh Matrices. All three of these methods use the stakeholder requirements to make trade-off decisions (Buede 2000).

Objectives hierarchy, also known as a directed tree, is a tool that is used to make decisions on alternatives. The objectives hierarchy organizes the breakdown of a set of values to better define an objective. These subdivided values are then weighted (adding up to 100%) according to the voice of the customer (Buede 2000).

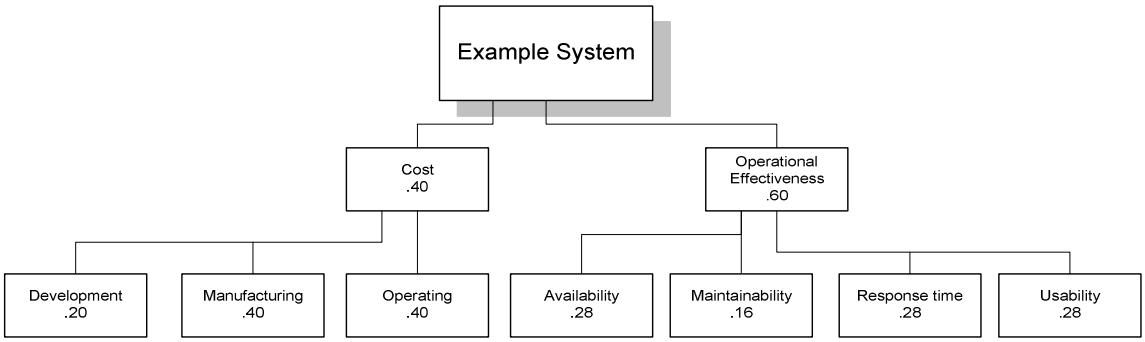

Figure 1: Example objectives hierarchy

QFD is a powerful tool for transforming the voice of the customer in to design quality, but can also be used to assess architectures. Figure 2 shows an example House of Quality found in the public domain which is an output of the QFD process. There are numerous documented processes on how to perform QFD – some of which may not even include a House of Quality, but most organizations that use QFD techniques have their own proprietary processes.

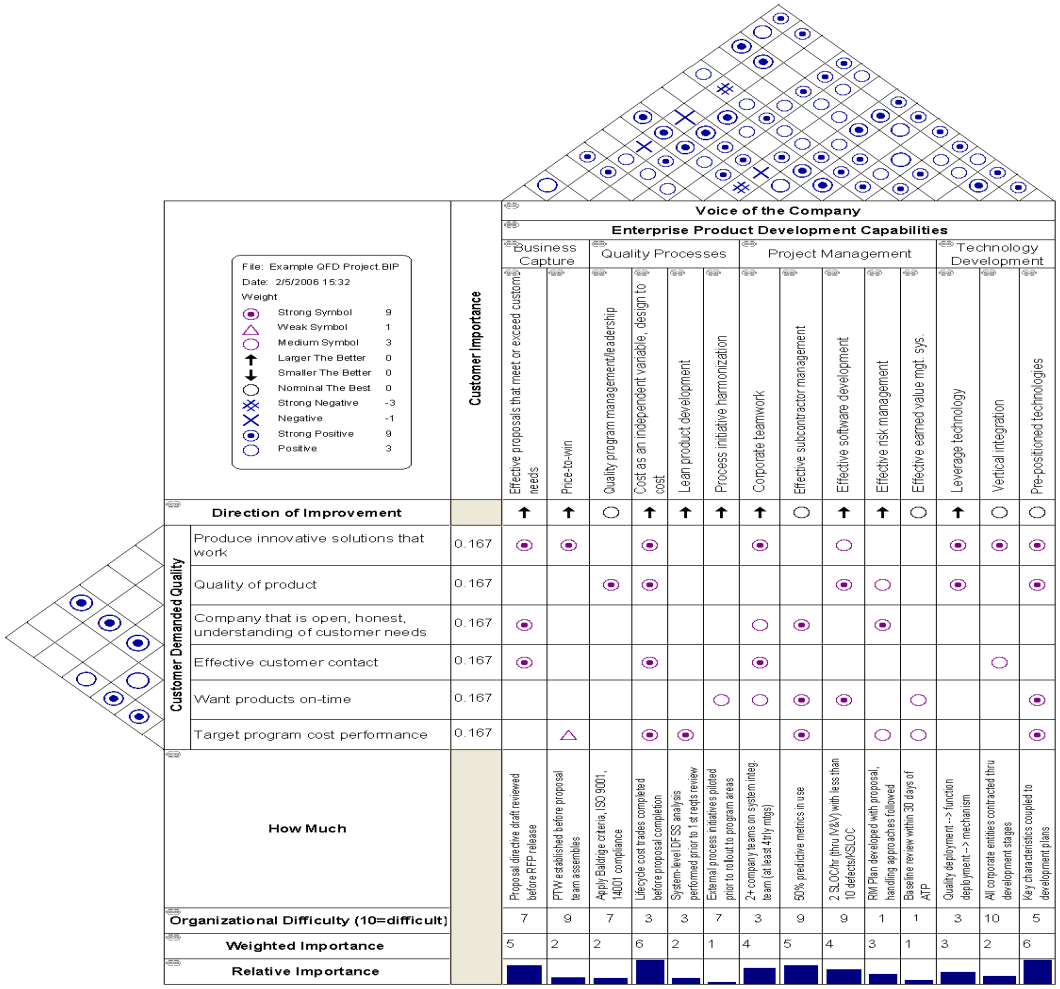

Figure 2: Example House of Quality

Pugh matrices are designed to be a simple means to make complex decisions concerning competing designs. Pugh Matrices are developed by listing competing concepts across the top and listing selection criteria along the side, then rating each concept by the criteria. Pugh matrices can be used in conjunction with QFD and other architecture assessment methodologies. An example Pugh matrix is shown in Figure 2.

	DESIGN ALTERNATIVES					
CRITERIA	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6
A	+	-	s	+	s	-
B	+	-	+	s	-	-
C	-	+	s	+	-	+
D	-	+	s	s	+	s
C	+	-	+	-	s	s
∑+	3	2	2	2	1	1
∑-	2	3	0	1	2	2
∑s	0	0	3	2	2	2

Key:
s = Satifies criteria
+ = Exceeds criteria
- = Does not meet criteria

Figure 2: Example Pugh design concept selection matrix

Architecture Assessment Attributes

According to ISO/IEC 15939:2002, an attribute is a "property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means (ISO/IEC 15939 2002)." The attributes of system architecture are important because they are used during architecture assessment to describe various aspects of the architecture. Because attributes are measurable, they are great for assessing, monitoring, and tracking system architecture (ISO/IEC 15939 2002). Attributes are normally derived from objectives or other attributes and are described by either subjective or objective measures (Keeney 1993). The use of attributes to measure and evaluate systems and systems architecture is valuable in making decisions and tradeoffs (Bass 1998).

Classification of Attributes

The effective use of attributes requires an understanding of how they relate to systems architecture and the instantiated system. There are many different types of attributes: directly-composable, architecture, derived, usage-dependent, context-dependent (Crnkovic 2004), and proxy (Keeney 2003), functional, and non-functional. These attribute classifications can be used to better understand the common behaviours of groups of attributes.

A *directly-composable* attribute of a system "is a function of and only of the same attribute" of the system's components (Crnkovic 2004). An example of a directly-composable attribute is weight. The weight of a system is additive so that the weights of the system's components can be added to calculate the weight of the system of interest (assuming all the weights are measured identically and in the same units). Directly-composable attributes are common, but usually only able to be estimated during architecture assessments.

Architecture-related attributes of a system "is any attribute that can be determined as a function" of its architecture (Crnkovic 2004). Some examples of architecture related attributes are functionality, security, and modifiability (Clements 2001). These attributes are easily assessed in architecture and are most affected by architecture issues.

A *derived attribute* is an attribute of a system that is composed from other attributes of that system (Crnkovic 2004). Dependability is composed from the attributes of availability, reliability, safety,

integrity, and maintainability (Avizienis 2004). Architecture attributes can be organized in a hierarchical manner, which increases the understanding of how system properties are related. A derived attribute would appear as a parent of the attributes it is derived from. The organization of the attribute hierarchy has value because it reveals how attributes are defined through interaction that would have otherwise remained hidden (Elias 2007).

A *usage-dependent attribute* of a system is dependent on how the system is used (Crnkovic 2004). An example of a usage dependant attribute is security because the security of the system may depend on how it is used. These attributes would be most easily measured through use-cases or similar techniques were user interactions are explicitly analyzed.

A *context-dependant attribute* is dependant on the environment that the system and system components exist in (Crnkovic 2004). All systems reside within a context. The context that a system resides in affects some attributes like availability. In the case of communication systems, environmental factors often affect the availability of the communication system to be used (i.e, the load on a communication line can prevent a handset from transmitting a signal). Context-dependant attributes are similar to usage-dependent attributes in that they are best assessed through use-cases or similar techniques. However, the main difference is whether the change in the attribute is caused through usage or external factors.

Proxy attributes indirectly describe the objective or attribute from which they are decomposed (Keeney 1993). This is an issue that faces many systems when the attributes are derived from the voice of the customer. Stakeholders often want systems to solve a problem that cannot be directly measured. For instance, take the example of a fire department whose objective is to minimize the loss of life and property. This can be decomposed into sub-objectives or attributes, which would be equally immeasurable. A solution to this would be to employ a proxy attribute: response time, which is objectively measurable and is understood to be correlated with the minimization of the loss of life and property (Keeney 1993). Even though response time doesn't directly measure the loss of life and property it provides an objective measure that can be used.

When classifying attributes, it is important to recognize that some attributes are related. These attributes are often related concepts that come about due to the use of language. This can be seen when considering the use of simplicity and complexity as attributes, which are antonyms. Since attributes are highly dependant on language, it is further important to define them carefully and specify how to measure them (ISO/IEC 15939 2002). When creating a set of attributes it is important that the set is complete, operational (usable), decomposable, non-redundant, and concise (Keeney 1993).

Using Attributes Taxonomies for System Behavior

This section outlines a set of selected important attributes for architecture evaluation. The authors have reviewed the existing taxonomies and have briefly outlined some of the relevant attributes for discussion and comparison purposes. Our review of these selected attributes is summarised in Table 2.

ISO/IEC 9126 describes quality software as having the characteristics of functionality, maintainability, usability, efficiency, reliability, and portability (ISO/IEC 9126 1995). By definition, these characteristics are attributes that together create a model for describing software. It should be further noted that this model further derives the sub-characteristics with more attributes; however these attributes are not well defined within this model (ISO/IEC 9126 1995).

Another model for describing software through attributes is the McCall Model. The attributes described in the McCall Model by category are product revision: maintainability, flexibility and testability; product transition: portability, reusability, interoperability; product operations: correctness, reliability, efficiency, integrity, and usability (Cavano 1978). The majority of the attributes in the McCall Model are addressed in ISO/IEC 9126 – or covered by similar attributes. However, no other model divides attributes into these categories.

The Boehm Quality Model along with the McCall Model and ISO/IEC 9126 1995 represent three of the most referenced models for software quality attributes. The Boehm Quality Model uses a tiered approach that derives attributes from parent attributes. Further, the lowest set of derived attributes are intended to have corresponding measures in order to measure the software under evaluation (Boehm 1978). Further work by Boehm in 1984, reports that the attributes for evaluating software specifications are different than those used for the software. These verification and validation attributes are: completeness, consistency, feasibility, and testability (Boehm 1984).

Not all software attributes can be applied to software architecture (Clements 2001). In the Architecture Tradeoff Analysis Method, the following attributes are evaluated: performance, reliability, availability, security, modifiability, portability, functionality, variability, subsetability and conceptual integrity (Kazman 2000). The attributes that can be evaluated by architecture are a subset of those for the system. In particular, attributes that are usage dependent or system environment context attributes may not easily be addressed by the architecture. The attributes used in ATAM together derive the attribute of suitability which is the main attribute evaluated (Clements 2001).

Avizienis, and others, note that important system attributes are: functionality, performance, dependability, security, usability, manageability, and adaptability. They further break down dependability as composed from the attributes of availability, reliability, safety, integrity, and maintainability (Avizienis 2004). Their hierarchy is used to comprehensively describe the attributes derived from dependability, and how, through an understanding, of these attributes, systems can be more dependable.

The Open Systems Joint Task Force states that it is 'never' inappropriate to take an open systems approach. However, they do concede that it may be inappropriate for a system to have an open systems implementation (Open Systems Joint Task Force 2005). Modularity and commonality are key attributes included in open systems concepts. Modularity and commonality, is also not for every system. Some systems have size and performance requirements that do not lend themselves to modularity (Marshall 1997). Not all attributes are relevant in every system domain (Dasgupta 1997); nevertheless, the consideration of these attributes is important to system success. Understanding which properties a system should not have is as important as which properties a system should have.

The SEI Quality Measure Taxonomy (SEI 2006) is one of the most complete taxonomies on the topic in existence. They consulted the work of Boehm, Barbacci, and others to compose this taxonomy. This list was originally created for software-systems; however it is relevant work for a SA attribute taxonomy and has been heavily referenced lately in this context.

Table 2: Summary of Attributes Taxonomies

4C's Model	McCall Quality Model	ISO/IEC 9126	Boehm Quality Model	Open Systems	SEI Quality Measure Taxonomy	Barbacci and Others	Clemence and others
Context: -System Purpose -System Scope and Environment -System Design -Internal System Interfaces -System Appropriateness (in fulfilling it's mission)	Product Revision - maintainability, flexibility, testability	Functionality- suitability, accuracy interoperability, compliance, security	Portability – device independence, completeness	Modular - cohesive, encapsulation, self contained, high binding	Needs Satisfaction - effectiveness, responsiveness, correctness, verifiability	Performance - latency, throughput, capacity	Availability
Characteristics: -System Constraints -Technical Feasibility of the System -Functional System Capabilities -Non-Functional System Capabilities -System Performance -System Design Documentation System Testability System Ergonomics (including aesthetics)	Product Transition - portability, reusability, interoperability	Reliability - maturity, recovery, fault tolerance	Reliability – completeness, accuracy, consistency	Standard Interfaces - interchangeability, interconnection, compatibility	Performance Measures - dependability, efficiency/resource utilization, usability, human engineering, fidelity	Dependability - availability, reliability, maintainability, safety, confidentiality, integrity	Security
Constraints: -System Compliance (Technology, Standards, and Regulations) -External System Interfaces -System Reliability -System Integratability -System Usability -System Adaptability and Scalability Security and Safety of the System	Product Operations:- correctness, reliability, efficiency, integrity, usability	Usability- learnability, understandability, operability	Efficiency – device efficiency, accessibility	interoperability	Maintenance Measures - maintainability, understandability	Security - confidentiality, integrity, availability	Modifiability
Contractuals: -System Configuration Management -System Deployment -System Sustainment -System Retirement -System Maintainability -System and Sub-Systems Reusability -System Programmatics (Cost, Schedule, Risk, ROI, TCO, etc) -Service Level Agreements on the System		Efficiency - time behavior, resource behavior	Human Engineering - accessibility, communicativeness,	portability	Adaptive Measures - interoperability, portability, scalability, reusability	Safety - iteration complexity, coupling strength,	Portability
		Maintainability - stability, analyzability, changeability, testability	Testability – accessibility, communicativeness, structuredness, self descriptiveness		Organizational Measures - cost of ownership, productivity		Functionality
		Portability - installability, replaceability, adaptability	Understandability – structuredness, self descriptiveness, conciseness, legibility				Variability
			Modifiability – structuredness, augmentability				Subsetability
							Conceptual Integrity

Clements and others list attributes to be used with software-intensive system evaluation methods like Architecture Tradeoff Method, Software Architecture Analysis Method, and Active Reviews of Intermediate designs (Clements 2001). They conclude that attributes can be used both early and late in the architecture lifecycle. Although the evaluation would be looking to answer different question, these attributes are still valid (Clements 2001). In addition, they provide the attributes of performance, reliability, availability, security, modifiability, portability, functionality, variability, subset ability, and conceptual integrity (Clements 2001). These attributes are listed without a taxonomy, and cite the evaluation methods previously listed for their usage.

Barbacci and others in their 1995 technical report list many important quality attributes (Barbacci 1995). These quality attributes are: performance, dependability, security, and safety (Barbacci 1995) - far from a complete list. Nevertheless, this work provides further insight into the four attributes listed by creating a generic taxonomy for each, and providing concerns, factors, and methods of use for each.

Jain and others in 2009 proposed the new attribute framework "4Cs" based on stakeholder concerns. The 4Cs describes the various categories of stakeholder architecture concerns: context, constraints, contractuals, and characteristics. The 4Cs are mapped to twenty-eight architecture attributes which help can be used to understand stakeholder information needs related to architecture (Jain et. al. 2009). Comprehensive architecture assessments must take a holistic approach to addressing all stakeholder needs, otherwise risking low stakeholder acceptance of the system.

System evaluation across a number of different domains shows that there are both technical and non-technical attributes to be considered. Early concept selection methods are important and of interest to government agencies and commercial industry across multiple domains.

Attribute Trade-offs

Any system has a delicate balance of system attributes to achieve its mission. Tipping the scales in any direction that destroys that balance may result in a failure of the system to fulfil its intended purpose. By using a balance beam analogy, we can see how getting the stakeholder(s) to describe that delicate balance is important (Buede 2000). All quality and value requirements should be derived through the stakeholder requirements.

While some attributes enhance each other, others conflict. These conflicts can be also seen in conflicting requirements. When the requirements ask for a high performance, robust, and economical architecture there are inherent conflicts.

Many projects are complex and dynamic, so making decisions can be difficult. There are many ways to deal with decision-making when faced with complex multivariate problems (Buede 2000). These techniques range from subjective to objective and simple to complex. In systems engineering, it is well established that the simple solution that fits the requirements is best (Maier 2002). This also holds true for analysis methods; the simplest technique that fits the requirements is most desirable. Analysis of multivariate attributes can be addressed simply (Buede 2000), as shown here, or in much more complex manners. The best techniques take into account the relevant stakeholder preferences. Two classic methods for dealing with system architecture tradeoffs are objectives hierarchy and Pugh. Both of these methods use the stakeholder requirements to make trade-off decisions (Buede 2000).

Attributes can be traded-off between each other or even within each other depending on the stakeholder requirements. Consider the case where both extreme performance and robustness is required for a real-time life critical system. Both performance and robustness will need to be bounded by scarcity of resources (i.e, Budget & Schedule).

Certain attributes are inherently difficult to trade-off. A classic example is the trade-off between network security, cost, and capabilities. The ultimate in network security and the lowest cost is to disconnect access to the outside world. This is often an unreasonable solution, and defeats the purpose of network access. On the other hand, the cost of a network intrusion is difficult to

quantify, so justifying expenditures for costly and restrictive security is daunting. Without clear data on the cost or risk of a network intrusion, constructing a cost-benefit analysis is impossible without using expert judgement.

Conclusions and Direction for Further Research

The above discussion is based on the authors' research on identifying and exploring architecture attributes for credible architecture evaluation. Most of the discussions on architecture assessment are focused on heuristics and subjective judgments of experienced team members. However, with the increasing size and complexity and pressures of 'time-to-market' it is becoming impossible to make architectural investment decisions based on just 'guesstimates'. The current literature still does not offer empirically validated architecture assessment methods. However, significant amount of discussion has been generated on architecture attributes. Further work needs to be done on validating the interaction and impact of architecture attributes on quality of architecture.

References

Avizienis, Laprie, Randell and Landwehr. January 2004. *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, v.1 n.1, p.11-33.

Barbacci and others. December 1995. *Technical Report: Quality Attributes*, Software Engineering Institute.

Bass, Clements, and Kazman. 1998. Software Architecture in Practice, Addison-Wesley.

Boehm, B. 1984. Verifying and Validating Software Requirements and Design Specifications, IEEE.

Boehm, B. and others. 1978. Characteristics of Software Quality, North-Holland Publishing Company.

Buede, D. 2000. *The Engineering Design of Systems*, Wiley Series in Systems Engineering, John Wiley & Sons, Inc. New York.

Cavano and McCall. Nov. 1978. *A Framework for the Measurement of Software Quality*. Proceedings of the Software Quality and Assurance Workshop, San Diego, CA, 133-139, ACM SIGMETRICS and SIGSOFT.

Clements, Kazman, and Klein. 2001. *Evaluating Software Architectures*: Methods and Case Studies, Addison-Wesley.

Crnkovic and Larsson. June, 2004. *Classification of quality attributes for predictability in component-based systems*, DSN 2004 Workshop on Architecting Dependable Systems, IEEE, Florence, Italy.

Dasgupta, A. et al. 1997. Perspectives to Understand Risks in the Electronic Industry, IEEE.

Elias, G., Jain R. 2007. *Exploring Attributes for Systems Architecture Evaluation, Conference on Systems Engineering Research*, Stevens Institute of Technology, Hoboken, NJ.

i Six Sigma. 2005. *i Six Sigma - Six Sigma Quality Resources for Achieving Six Sigma Results*, http://www.isigma.com.

ISO/IEC 15939. 2002. *Software engineering – Software Measurement Process*.

ISO/IEC 9126. 1991. *Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their use*.

Jain, R.; Chandrasekaran, A.; Elias, G.; Cloutier, R. June 2008. *Exploring the Impact of Systems Architecture and Systems Requirements on Systems Integration Complexity*, Systems Journal, IEEE, Volume 2, Issue 2, Page(s): 209 – 223.

Jain, R.; Chandrasekaran, A.; Elias, G. May 2009. *System Architecture Concerns: A Stakeholder's Perspective*, The Journal of Enterprise Architecture, Volume 5, Number 2, Page(s): 41 – 66.

Kazman, R.; Klein, M.; & Clements, P. 2000. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Keeney, R. and Raiffa, H., 1976. *Decisions with Multiple Objectives*. Wiley, New York.

Maier, M.W. and Rechtin, E. 2002. *The Art of Systems Architecting Second Edition*, CRC Press, New York.

Marshall, Russell. 1997. *Holonic Product Design Workbook*." Loughborough University.

Nord, R.; Barbacci, M.; Clements, P.; Kazman, R.; Klein, M.; O'Brien, L.; & Tomayko, J. 2004. *Integrating the Architecture Tradeoff Analysis Method with the Cost Benefit Analysis Method* (CMU/SEI-2003-TN-038) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Osman Balci and William F. Ormsby. 2008. *Network-Centric Military System Architecture Assessment Methodology*, International Journal of System of Systems Engineering 1, 1-2, 271-292.

Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb. May 1994. *SAAM: A Method for Analyzing the Properties Software Architectures*, Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, pp. 81-90.

Software Engineering Institute (SEI). 2006. Carnegie Mellon, *Quality Measures Taxonomy*, http://www.sei.cmu.edu/str/taxonomies/qm_tax_body.html.

Zachman J., "A Framework for Information Systems Architecture", IBM Systems Journal, Vol. 26, No. 3, 1987.

BIOGRAPHY

George M. Elias is a Systems Engineering Doctoral Candidate at Stevens Institute of Technology. Mr. Elias is a Program Manager at ITT Electronic Systems in Clifton, New Jersey. Mr. Elias has an Undergraduate degree in Information Systems from Rutgers and New Jersey Institute of Technology and a Masters in Computer Science from Stevens Institute of Technology. Mr. Elias has research interests in Systems Architecture and Systems Engineering Attributes.

Dr. Rashmi Jain is Associate Professor of Systems Engineering at Stevens Institute of Technology. Dr. Jain has over 15 years of experience of working on socio-economic and information technology (IT) systems. Over the course of her career she has been involved in leading the

implementation of large and complex systems engineering and integration projects. Her teaching and research interests include systems integration, systems architecture and design, and rapid systems engineering. Dr. Jain is currently the Head of Education and Research of INCOSE. In this role she is leading the development of a reference Systems Engineering curriculum. She holds Ph.D. and M.S. degrees in Technology Management from Stevens Institute of Technology.