Exploring the Impact of Systems Architecture and Systems Requirements on Systems Integration Complexity

Dr. Rashmi Jain[i], Anithashree Chandrasekaran, George Elias, Dr. Robert Cloutier
Stevens Institute of Technology
Rashmi.Jain@stevens.edu, Anithashree.Chandrasekaran@stevens.edu, gelias@stevens.edu, Robert.Cloutier@stevens.edu

Abstract

The need to perform faster systems integration of complex systems require the architect and design team understand how the selected architecture and design components will impact the system integration processes complexity. System integration process complexity is an outcome of the interaction between degree of feasibility and level of effort required to understand, describe, implement, manage and document the system integration process for a given system development and operational environment. This paper analyzes the cause and effect relationships between the system requirements, architecture and the system integration processes complexity. In order to address systems integration issues upfront in the design phase it is necessary to determine if the architecture and design of components, sub-systems, processes and interfaces impacts (and to what extent) system integration process complexity. This paper also defines, and analyzes the impact of the different system architecture and requirements factors on system integration process complexity. A research framework is developed to understand the cause and effect relationships between the system requirements, architecture and integration process. Finally, the paper proposes recommendations based on the causality results. These conclusions are based on research undertaken by the authors on 8 development projects in the government sector.

Keywords: System Architecture, System Integration, Integration Complexity, System Requirements

Introduction

It is necessary to determine if integration of the defined physical elements and interfaces in system architecture contributes to system integration process complexity. Today's systems are usually expected to function independently and in cooperation with the existing systems (system of systems). The cooperating systems undergo frequent changes and interoperability becomes a key factor and a major contributor to system integration process complexity. Interoperability is based on the existence of the conceptual view [Carney and Oberndoff, 2004]. The conceptual view can be embodied in requirements and architecture/design and the system architecture determines the level of interoperability. The system integration process complexity includes interoperability.

The major focus of the paper is on the cause and effect relationship between system requirements, system architecture, and system integration process complexity. The paper is based on the research that includes system engineering literature reviews, study of industry best practices and a survey of the system engineering leads for eight different development projects in a government organization. The research methodology used for this study is provided in Figure 1. This paper is organized in a similar outline, first establishing the context by defining and discussing system integration process, and its complexity classification. In order to understand, analyze and prioritize the causal factors a cause and effect relationship model was developed and relevant attributes of good requirements and architecture were identified. These requirements and architecture attributes define the activities involved in these two critical phases and enable a relatively simplified integration process. Next, a set of activities/factors of system requirements and system architecture were selected. The selection of these activities and factors were based on the assumption that these activities contribute towards better requirements and architecture and would also reduce the system integration process complexity, and improve the quality of system verification and validation. A survey was then developed to verify and validate this. Finally, this paper discusses a set of research questions and the findings to further provide a meaningful analysis of these relationships between system requirements, architecture and integration process complexity.

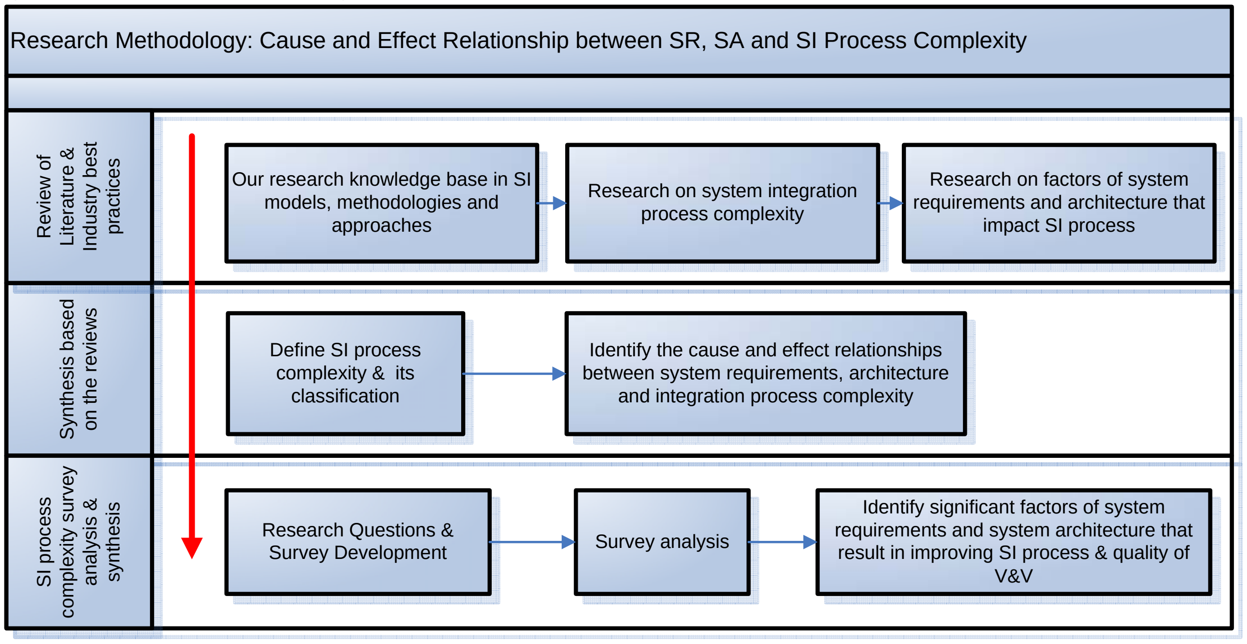

Research Methodology: Cause and Effect Relationship between SR, SA and SI Process Complexity
Review of Literature & Industry best practices
Our research knowledge base in SI models, methodologies and approaches
Research on system integration process complexity
Research on factors of system requirements and architecture that impact SI process
Synthesis based on the reviews
Define SI process complexity & its classification
Identify the cause and effect relationships between system requirements, architecture and integration process complexity
SI process complexity survey analysis & synthesis
Research Questions & Survey Development
Survey analysis
Identify significant factors of system requirements and system architecture that result in improving SI process & quality of V&V

Figure 1 Research Methodology

System Integration Process Complexity

System integration (SI) process is defined as a set of activities that transforms the stakeholder requirements into an operational system by unifying the process components and product components while ensuring compliance to the specified levels of component operations and interoperability/interdependence.

Systems integration is the process that links the systems engineering life cycle process from requirements to verification and validation and ultimately implementation of the system as shown in the Figure 2. This linking process is also called traceability, which is an integral component of an effective systems integration process.

The system architecture and design is determined by the system requirements which are originated and derived from the stakeholder requirements. Requirements specification involves translating system requirements into a formal representation of interrelated units. The output of this process serves as a blueprint of the system which guides and controls all the subsequent development activities [Dogru, et.al, 1992]. The architecture and design decomposes and allocates the system functionality into components, sometimes referred to as Hardware Configuration Items (HWCI) and Computer Software Configuration Items (CSCI), and defines the interfaces between these configuration items. The complexity of the systems integration process may be determined by the functional and physical decomposition of the system architecture.

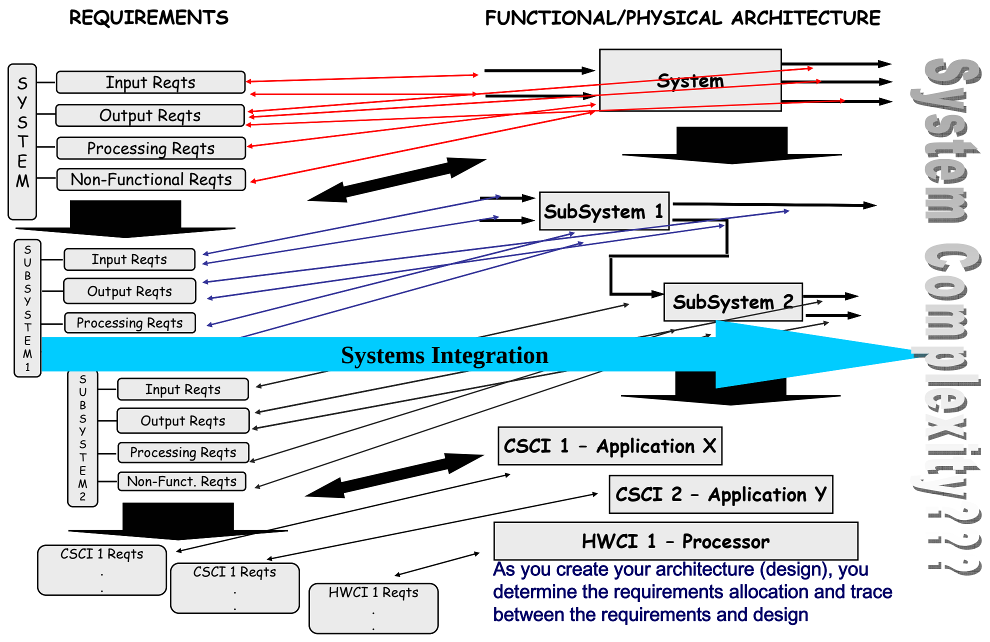


Figure 2 System Integration[ii]

Figure 2 demonstrates the process by which the functional architecture enables the requirements to be traced and defined for each of the configuration items that are a part of the physical architecture. These requirements form the basis for hardware and software detail design and development. Requirements are usually allocated to functions (within the functional architecture), that are performed by physical elements. In [Rossak and Prasad, 1991] Rossak and Prasad state that integration architectures for system of systems serve as a general pattern to define the basic layout of an integrated system. The integration architectures deal with the development of new system parts and the post-facto integration of already existing solutions by providing strategy of system decomposition, data storage, inter-process and user communication

The importance of the system integration process in the development of a system and its successful implementation can be understood and appreciated by understanding and familiarizing with these SI process activities. Despite the large investments that are made, many integration efforts fail for a number of technical, organizational and management, and migration planning reasons. The technical issues include the following [Smith, et al. 2002]:

- Legacy systems originated from

unplanned, stovepipe development or were developed as batch, single tier.

- Data is specific to the applications and was not designed for sharing.
- The legacy systems were not initially designed for new quality-attribute requirements and are being affected by needs for interoperability, performance, security, and usability.
- The scope for the integration effort is not adequately defined.
- Decisions are made without performing adequate analyses (sometimes referred to as "management by magazine")

The exponential development and growth of technology and increase in user demands has resulted in increase of complexity in the system integration and development process. Complexity in this context is the degree to which a system or component has a design or implementation that is difficult to understand and verify [IEEE 610-12, 1990]. Complexity is also defined as the degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the types of data structures [Evans and Marciniak, 1987]. Any measurement of process complexity requires measurement of situation complexity, action complexity and intention complexity. [Howard, et al. 2004]

We define

System Integration process complexity as an outcome of the interaction between degree of feasibility and level of effort required to understand, describe, implement, manage and document the system integration process for a given system development and operational environment.

Some of the factors that impact the degree of feasibility and the level of effort required to understand, describe, implement, manage and document the system integration process activities are shown in Table 1. This list is based on our system integration framework and system integration process model (SI^{PM}) [Jain, et al. 2007a, 2007b].

Table 1 Factors that impact degree of feasibility and level of effort required for SI

Factors that impact	
Degree of feasibility	**Level of effort required**
Availability of the integration (including V&V) methodologies and tools	Documentation of the system requirements
Availability of the required external and internal interfaces	System Integration Planning, Control and Management
Scope of COTS requirements (interface and interoperability)	Documentation of system integration specifications (including V&V)
Scope of legacy requirements (interface and interoperability)	Familiarity and knowledge of the integration (including V&V) methodologies and tools
Adherence to the standards, regulations and guidelines	Familiarity and knowledge of the required external and internal interfaces
Planned programmatic resources for the system integration process	Knowledge of the system constraints, context, environment and scope
Level of performance metrics to be supported	Specification of performance metrics (What, How, When to measure? Who will measure?)
Level of operational/service metrics to be supported	Specification of operational/service metrics (What, How, When to measure? Who will measure?)

Factors that impact	
Degree of feasibility	**Level of effort required**
	Degree of automation of the SI process
	Number of dependencies for SI process activities (Degree of concurrency of the SI process)

Most of the research discussion on system integration complexity is in the context of the complexity associated with the entire system. Currently, system integration complexity is mainly focused on the degree of testing required and interface complexity. But there is a need to look at system integration complexity from both the process and product point of view in order to ascertain associated risks in a comprehensive manner. . To address these aspects we classify system integration process complexity as Technical, Programmatic, Configuration, Operational, and Organizational. This classification is based on how the system lifecycle and the development activities impact the system integration process. These are described below.

Technical complexity: The complexity of system integration process due to the required system capabilities and functions. The major factors of technical complexity are the integration technologies required to achieve the system capabilities and functions, feasibility and performance of the interfaces, and strategies, methodologies and tools for verification and validation of the technical effectiveness of the system and its sub-systems.

Programmatic complexity: Programmatic complexity includes system integration process complexity arising out of the variance between the planned and available resources needed to support the system integration process over its lifecycle. The major variances resulting in such programmatic complexity are allocated budget, cost associated with system integration activities, schedule, materials/equipments/tools, and skill sets.

Configuration complexity: The complexity of system integration process due to the inconsistencies and lack of control in the development of process and product. The major factors of configuration complexity are the control and management of changes, volatility and clarity of documentation and specifications, integration baselines, and integration personnel communication.

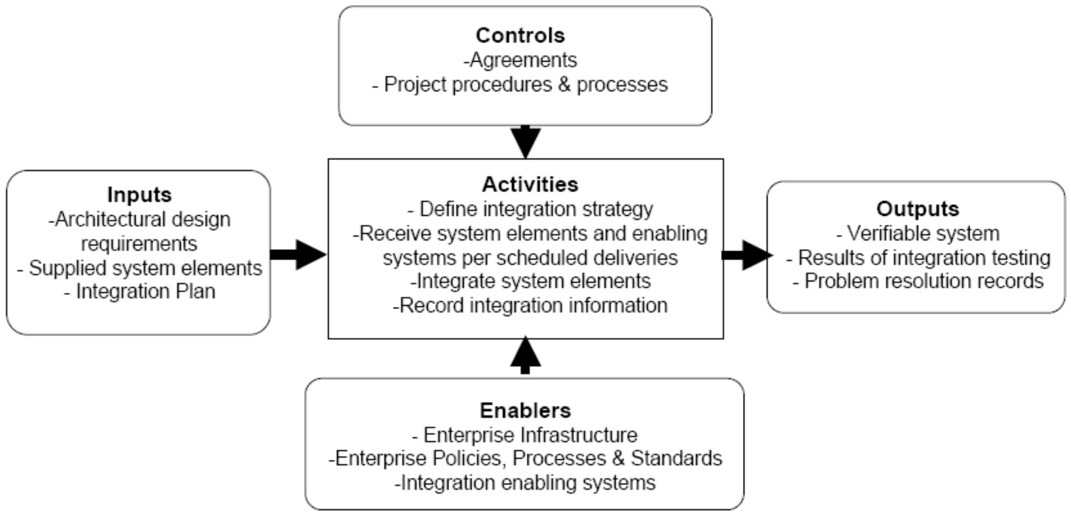
Operational complexity: This complexity results from providing and supporting the required level of operational support and system availability. The system integration operational complexity can be observed in the required level of effort to integrate, verify and validate the availability and operational support of the system and its sub-systems.

Organizational complexity: The complexity of system integration process due to the organizational strategy, compliance, processes and product line. The major factors of organizational complexity are service level agreements of sub-systems and interfaces, standards, regulations and guidelines for system integration, and legacy system integration.

Designing for integration

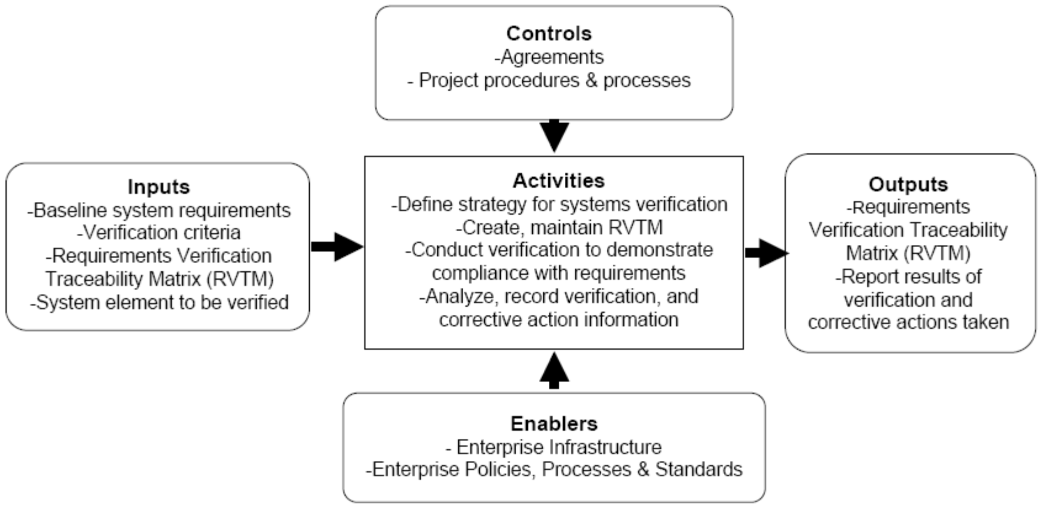
Most of the factors that result in SI process complexity are external to the context of system integration process. The system integration processes and activities receive most of its inputs, controls and triggers from system requirements and system architecture (Figure 3, Figure 4, Figure 5). This highlights the direct cause and effect relationship between the system requirements, architecture and system integration. System requirements and architecture has a direct impact on system integration.

System architecture also receives inputs, triggers and controls from system requirement activities. Hence there is a mediating and moderating effect of system architecture on system integration. System architecture can play a direct causer or a moderator or a mediator depending on the system context, and concept.



Controls
-Agreements
- Project procedures & processes
Inputs
-Architectural design requirements
- Supplied system elements
- Integration Plan
Activities
- Define integration strategy
-Receive system elements and enabling systems per scheduled deliveries
-Integrate system elements
-Record integration information
Outputs
- Verifiable system
- Results of integration testing
- Problem resolution records
Enablers
- Enterprise Infrastructure
-Enterprise Policies, Processes & Standards
-Integration enabling systems

Figure 3 Context Diagram for the Integration Process [SE Handbook, 2006]



Controls
-Agreements
- Project procedures & processes
Inputs
-Baseline system requirements
-Verification criteria
-Requirements Verification Traceability Matrix (RVTM)
-System element to be verified
Activities
-Define strategy for systems verification
-Create, maintain RVTM
-Conduct verification to demonstrate compliance with requirements
-Analyze, record verification, and corrective action information
Outputs
-Requirements Verification Traceability Matrix (RVTM)
-Report results of verification and corrective actions taken
Enablers
- Enterprise Infrastructure
-Enterprise Policies, Processes & Standards

Figure 4 Context Diagram for the Verification Process [SE Handbook, 2006]

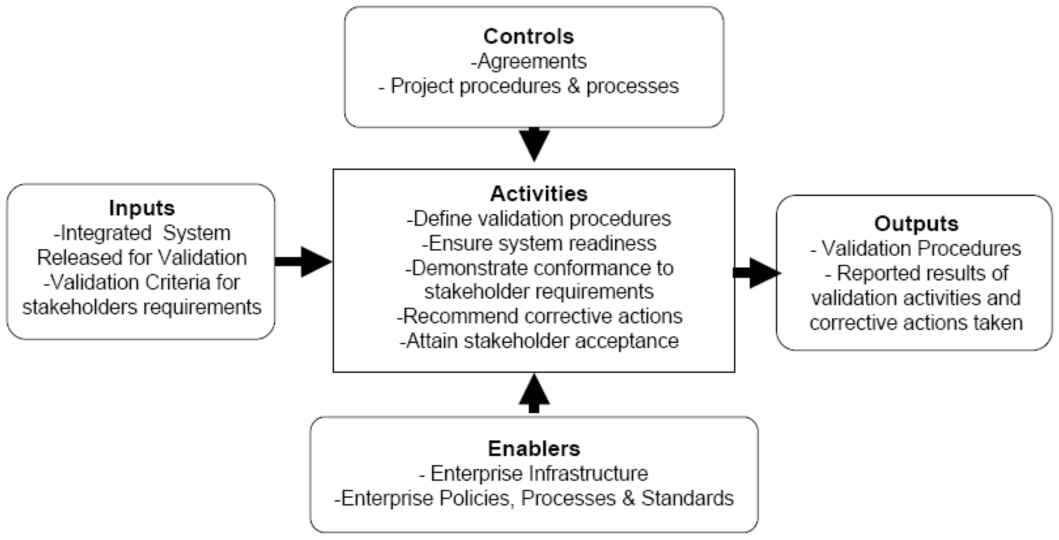

Controls
-Agreements
- Project procedures & processes
Inputs
-Integrated System Released for Validation
-Validation Criteria for stakeholders requirements
Activities
-Define validation procedures
-Ensure system readiness
-Demonstrate conformance to stakeholder requirements
-Recommend corrective actions
-Attain stakeholder acceptance
Outputs
- Validation Procedures
- Reported results of validation activities and corrective actions taken
Enablers
- Enterprise Infrastructure
-Enterprise Policies, Processes & Standards

Figure 5 Context Diagram for the Validation Process [SE Handbook, 2006]

The goodness of system architecture and requirements can be defined by some attributes. The attributes of a system or architecture are important because they are used to describe the properties of the system or the system architecture in a unique distinguishing manner [Elias and Jain, 2007]. System attributes are a key link between good system architecture and the system's ultimate development cost and schedule [McCabe and Pollen, 2004]. A system attribute denotes a characteristic or category of requirements commonly demanded of systems. Recent work suggests that system attributes offer an effective set of perspectives to evaluate architectural decisions and tradeoffs [Bass, 2003]. The set of attributes shown in Figure 6 provides standard guidelines for system requirements and system architecture. By addressing some of these attributes of system architecture and requirements some of the significant system integration issues can be contained and controlled effectively resulting in a relatively simplified system integration process. A comprehensive list of attributes of architecture can be obtained from SEI Quality Measure Taxonomy [SEI, 2006], [Clements, 2001], and [Elias and Jain, 2007].

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 a system or an architecture are used to describe the properties of the system or the SA in a unique distinguishing manner. Because attributes are measurable they are ideal for describing and monitoring many systems engineering tasks [ISO/IEC 15939, 2002]. The use of attributes to measure and evaluate systems and systems architecture is valuable in making decisions and tradeoffs [Bass, 2003]. In this paper, we focus on important attributes that contribute to System Integration Complexity (SIC). By understanding which attributes of SA affect SIC one could use this information throughout the requirements, architecting, and development phases of the system lifecycle.

SIC can be affected by multiple factors. We are primarily concerned with system architecture aspects that impact SIC such as commonality, modularity, standards-based, and reliability, maintainability, and testability (RMT) as shown in Figure 6.

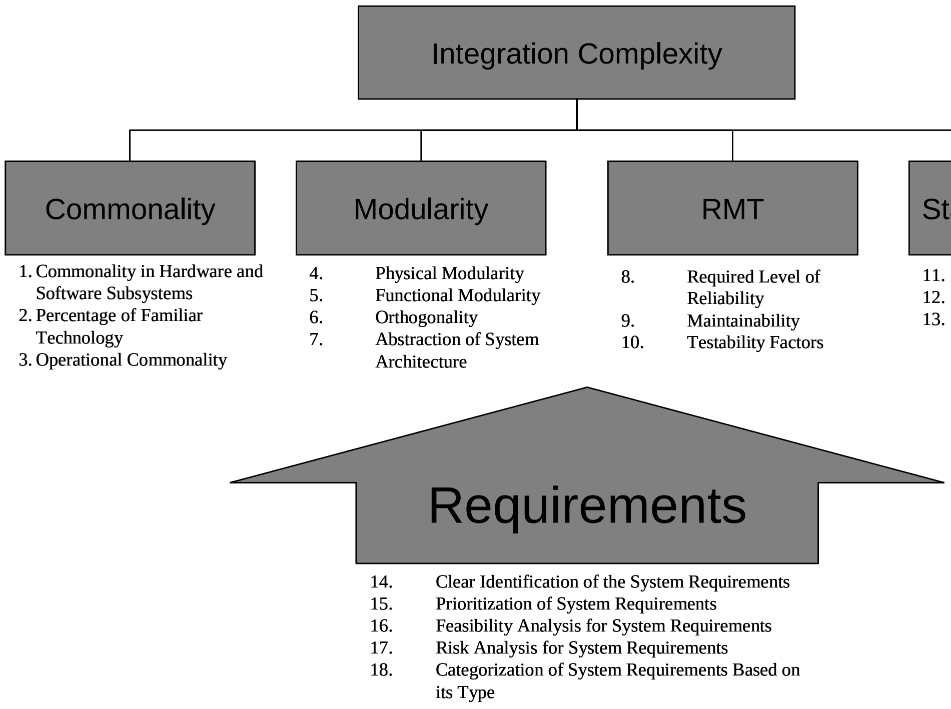

Integration Complexity
Commonality
Modularity
RMT
Standards Based
1. Commonality in Hardware and Software Subsystems
2. Percentage of Familiar Technology
3. Operational Commonality
4. Physical Modularity
5. Functional Modularity
6. Orthogonality
7. Abstraction of System Architecture
8. Required Level of Reliability
9. Maintainability
10. Testability Factors
11. Interface Openness
12. Open System Orientation
13. Consistency Orientation
Requirements
14. Clear Identification of the System Requirements
15. Prioritization of System Requirements
16. Feasibility Analysis for System Requirements
17. Risk Analysis for System Requirements
18. Categorization of System Requirements Based on its Type

Figure 6 Architectural attributes that impact SIC

Commonality refers to the systems design where a component or subsystem can be used in more than one place in the system. Commonality directly corresponds to the degree to which components and subsystems are reused within the system. Commonality is highly related to reuse when multiple systems have subsystems in common. Reusability is the degree to which a module, component, system or other work product can be used more than once [IEEE 610-12, 1990]. Operational commonality has to do with how close-related systems or subsystems are operationally similar. This directly affects the attributes of usability and maintainability. Another aspect of commonality is architecting with a view to use familiar technology. Familiarity of technology helps design more open and flexible architectures. Factors that negatively affect commonality are: the number of unique Line Replacement Units (LRU), number of unique fasteners, number of unique cables, number of unique standards implemented, number of unique software packages implemented, number of languages, number of compilers, average number of software instantiations.

Modularity is the degree to which a system is structured as a configuration of smaller, self-contained units with well-defined interfaces and interactions (i.e., independently testable), moderating design complexity and enhancing its clarity, and enabling design and functional flexibility and variety for the system as a whole [McCabe and Pollen, 2004], [Open Systems Joint Task Force, 2005]. Modularity is usually associated with open systems design, but in this case we are considering open systems design to be a design standard, while modularity is a good design practice on its own. Nevertheless, both of these concepts are covered regardless of where they are placed in this structure. Abstracting the system architecture enables the architect to review several alternative solutions before choosing the one that will be implemented. Differentiating the layers of functionality and the structure that is required to support through abstraction simplifies the choice in many cases. Abstraction also facilitates reusability and portability by preserving the boundaries of the different layers [Jorgensen and Philpott, 2002].

Standards based systems are designed and architected based on open standards. Open systems employ a system design philosophy which provides for interoperability and portability. In addition to using an open systems approach, the architecting or implementing organization may choose to document their own design standards. Organizational design standards create consistency in the way systems are architected and designed by an organization. The downside to designing to standards is that they may impose constraints that become problematic in relation to technology issues, or may not even be possible in some cases. While designing to standards may not always be possible or desirable, it is always important to consider design standards.

Reliability is often a very important factor in the design of a system. Reliability has two classic definitions that apply to systems: 1) The duration or probability of failure-free performance under stated conditions, and 2) The probability that an item can perform its intended function for a specified interval under stated conditions. (For non-redundant items, this is equivalent to definition (1). For redundant items, this is equivalent to mission reliability [MILSTD-1388-1A, 1983].

Unfortunately, when systems are not designed with maintainability in mind it is usually left out. Maintainability is the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [IEEE 610-12, 1990]. Methods to measure maintainability are mean time to repair (MTTR), maximum fault group size, etc.

Testability is the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE 610-12, 1990]. Examples of issues that affect testability: number of LRUs covered by BIT (Built In Test), logging/recording capability, ability to create system state at time of system failure, online testing, ability to be operational during external testing, ease of access to external test points, automated input/stimulation insertion).

Requirements are the life-blood of systems development. Every man-made system starts with requirements regardless of whether they are implicit, explicit, good or bad. The first phase of the systems engineering process is driven by collecting, composing and analyzing systems requirements. These requirements drive the success or failure of the system throughout the lifecycle. The decisions made through detailed design commit approximately 80% of the cost of the system [Buede, 2000]. Therefore, requirements are a very important driver of the systems engineering process and require substantial attention. Requirements that are generated as a result of stakeholder concerns, and written in the context of the key attributes about to be listed, increase that likelihood of system success.

Based on our research, we identified the following 18 activities that characterize the impact of system requirements and architecture related attributes on systems integration complexity.

1. Commonality in hardware and software subsystems (such as number of unique Line Replacement Units (LRU), number of unique fasteners, number of unique cables, number of unique standards implemented, number of unique software packages implemented, number of languages, number of compilers,

average number of software instantiations)
2. Percentage of familiar technology used in the system of interest
3. Operational commonality (such as percentage of operational functions automated, number of unique Skill codes required, estimated operational training time, estimated maintenance training time)
4. Physical modularity (such as ease of system element upgrade, ease of operating system upgrade)
5. Functional modularity (such as ease of adding new functionality, ease of upgrade existing functionality)
6. Orthogonality (examples are functional requirements fragmented across multiple processing elements and interfaces, throughput requirements across interfaces, common specifications identified)
7. Abstraction of system architecture
8. Required level of reliability (factors such as fault tolerance, percentage of system loading (processor loading, memory loading, network loading, etc…))
9. Maintainability in terms of expected MTTR (mean time to repair), maximum fault group size, accessibility and required system operational during maintenance
10. Testability factors (Examples: number of LRUs covered by BIT (BIT Coverage), reproducibility of errors, logging/recording capability, ability to create system state at time of system failure, online testing, ability to be operational during external testing, ease of access to external test points, automated input/stimulation insertion)
11. Interface openness (such as number of Interface Standards/Interfaces, multiple vendors, multiple business domains, standard maturity)
12. Open system orientation (hardware and software standards)
13. Consistency orientation (common guidelines for implementing diagnostics and performance monitoring and fault localization)
14. Clear identification of the system requirements (uniquely identified (i.e., number, name tag, mnemonic, hypertext) and reflect linkages and relationships)
15. Prioritization of system requirements (based on stakeholders' input and criticality analysis)
16. Feasibility analysis for system requirements
17. Risk analysis of system requirements
18. Categorization of system requirements based on its type (such as input, output, reliability, availability, security, environmental, performance, interface, testing)

Some of the significant sources of the causality analysis include [Verma and Johannesen, 2000], Architecture Tradeoffs Analysis Method (SEI ATAM), IBM Architecture Evaluation Methodology (IBM AEM), [Bachmann, et al. 2000], [Bachmann, et al. 2002], [Kazman, et al. 2003], [Bachmann, et al. 2003], [IEEE 1233, 1998], [IEEE 830, 1998], INCOSE SE Handbook v3 [SE Handbook, 2006]. These system architecture and requirements activities when implemented can result in certain clearly observable patterns in system architecture and requirements [Cloutier, 2006], [Cloutier and Verma, 2007]. By studying and mining similar systems, common patterns are found that go well beyond software patterns in common use today. These patterns can form the basis for entire subsystems of a complex system. The Perform C2 (command and control) is an example of such a pattern [Cloutier and Verma, 2006]. In this system architecture pattern, the smaller plan, detect, control, and engage patterns are assembled into a pattern language called Perform C2. One of the motivators documented for the use of system patterns is the value of managing complexity. When a pattern such as the Perform C2 is implemented and integrated into a system, the integration complexities are better understood, and can be leverage in subsequent implementation and integration efforts if the same pattern is used again. The same can be said for patterns that evolve when developing embedded system in compatible languages or common platform, using fasteners of the same specification within and among the subsystems.

Each of the identified system requirements and architecture activity and factor help address at least one category of system integration process complexity. The impact of these activities on the complexity category is a result of the dependencies between the complexity factors and the activities. A mapping was created to provide an overview of how each activity/factor impact the system integration process complexity categories. The mapping is shown in Table 2.

Table 2 System Architecture and Requirements Cause and Effect on System Integration Process Complexity

System Architecture and Requirements Factors	Technical complexity	Programmatic complexity	Configuration complexity	Operational complexity	Organizational complexity
Commonality in hardware and software subsystems	X				
Percentage of familiar technology		X	X		X
Operational commonality		X		X	X
Physical modularity	X				
Functional modularity	X				
Orthogonality	X		X		
Abstraction of system architecture		X	X		X
Required level of reliability				X	
Required level of maintainability				X	
Testability		X			
Interface openness				X	X
Open system orientation		X	X		X
Consistency orientation	X		X		
Clear identification of the system requirements	X	X		X	X
Prioritization of system requirements	X		X		
System requirements feasibility analysis	X	X		X	
System requirements risk analysis	X	X		X	X
Categorization of system requirements	X	X	X	X	X

Formulating hypothetical SI complexity causal relationships

Through the course of this work, research questions were constructed to assist in the understanding of how and to what extent the identified system architecture and requirements factors impact the system integration process. By analyzing these research questions we arrive at some recommended practices to handle the cause and effect relationship between system architecture and requirements that would simplify the system integration process and reduce the dependencies between these development phases. Addressing some of the system integration issues during the system requirements and architecture phase provides a strong foundation for the integration phase and help manage the associated criticality of these issues. When the foundation for the system integration activities and processes is initiated early in the lifecycle, feedback and iterations during development helps to refine and improve system integration strategies, planning and quality.

The two factors (dependent variables) considered for this research study were the system integration complexity (system integration process complexity) and quality of system verification and validation (V&V). The most important phase of system integration is the system verification and validation. V&V are the phases of system engineering where the required functionality comes together with all the interfaces completed. Hence studying the quality of V&V along with the SI complexity is important and provides a comprehensive view of system integration. The independent variables are the system architecture and requirements factors discussed in the previous section. We formed the following research questions to understand the cause and effect relationships between system architecture and requirements with system integration process complexity.

1. Null Hypothesis: There is a cause and effect relationship between system requirements, system architecture and system integration process complexity.
2. Null Hypothesis: There is a cause and effect relationship between system requirements, system architecture and quality of V & V.
3. What are the three most important factors of system requirements and architecture that could reduce system integration complexity?
4. What are the three most important factors of system requirements and architecture that could improve quality of system verification and validation?
5. Does system architecture impact the complexity of system integration? Hypothesis: System architecture improvements can reduce system integration complexity.
6. What are the three most important architectural factors for system integration complexity?
7. Does system architecture impact the quality of system verification and validation? Hypothesis: System architecture improvements can improve the quality of system verification and validation.
8. What are the three most important architectural factors for quality of verification and validation?
9. Does the level system reliability and maintainability impact the complexity of system integration and the quality of verification and validation? Hypothesis: Higher levels of reliability and maintainability in system design reduce the complexity of system integration and increase the quality of verification and validation.

10. Does an improved requirement engineering process results in reduced system integration complexity and better quality of verification and validation?
11. Does familiarity of technology lead to reduced system integration complexity and better quality of verification and validation?

A survey questionnaire addressing each of these identified system architecture and requirements factors was developed. The survey questions are designed to verify if there is a cause and effect relationship between each of the identified 18 factors on system integration complexity and quality of system verification and validation and also measure their impacts. The survey questions were also designed to capture the comments (thoughts and inputs) of the participants on each of these 36 cause-and-effect relationships. The questionnaire with 36 questions was piloted to the SE personnel at a government organization to obtain their feedback on the survey clarity and terminologies used. After the completion of the pilot, data was collected on 8 different development projects. The survey responses are shown in Table 3, and Table 4. The impact of the causal factors of requirements engineering on system integration complexity is shown in Figure 7, and Figure 8. The impact of the causal factors of system architecture on system integration complexity is shown in Figure 9, and Figure 10. The impact of the causal factors of requirements engineering on quality of verification and validation is shown in Figure 11, and Figure 12. The impact of the causal factors of system architecture on quality of verification and validation is shown in Figure 13, and Figure 14.

Table 3 System Integration Process Complexity: Survey Results

#	SI is improved as a result of	Not Improved (%)	Improved (%)			
			Partially	Moderately	Significantly	Exceptionally
1	Categorization of system requirements based on its type	25	13	50	0	13
2	Clear identification of the system requirements	0	0	38	25	38
3	Commonality in hardware and software subsystems	0	0	63	25	13
4	Decrease in abstraction of the system architecture	67	17	17	0	0
5	Decrease in expected maintainability	71	14	14	0	0
6	Decrease in interface openness	29	29	14	29	0
7	Decrease in level of required reliability	43	29	14	14	0
8	Decrease in orthogonality	0	14	29	57	0
9	Decrease in testability factors	33	17	33	17	0
10	Increase in consistency orientation	0	29	43	29	0
11	Increase in functional modularity	0	0	43	43	14
12	Increase in open system orientation	25	25	13	38	0
13	Increase in operational commonality	13	25	50	13	0
14	Increase in percentage of familiar technology	13	13	25	50	0
15	Increase in physical modularity	0	0	57	29	14
16	Prioritization of system requirements	0	13	38	25	25
17	System requirement feasibility analysis	25	13	38	13	13
18	System requirement risk analysis	13	13	38	25	13

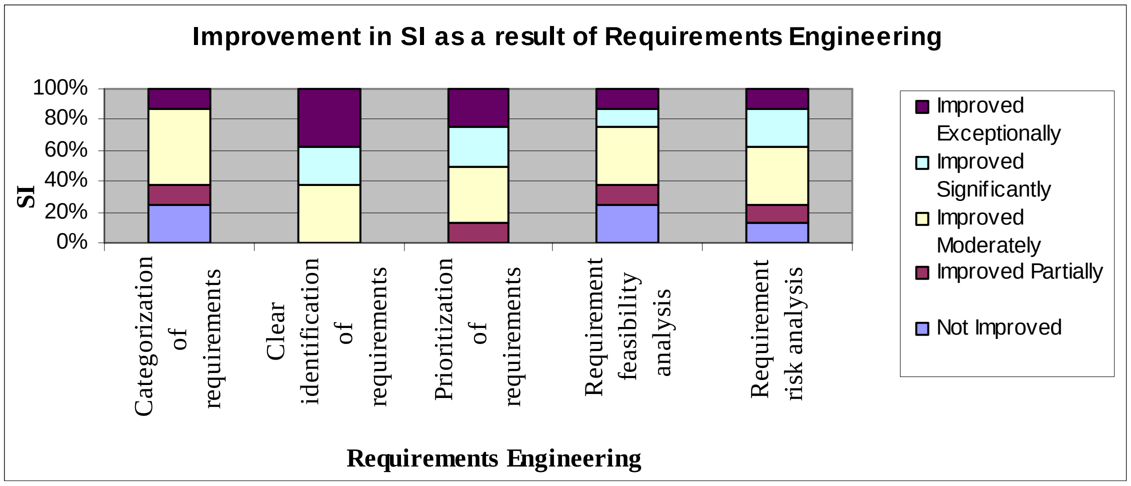


Figure 7 SI process improvements based on specific Requirements Engineering activities

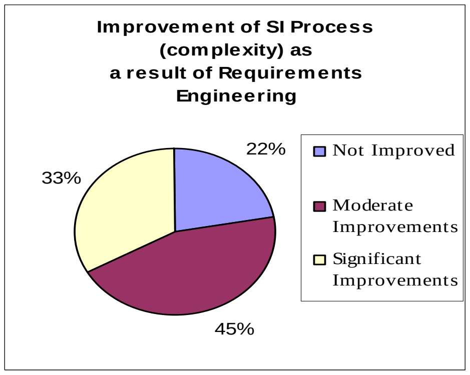


Figure 8 SI process improvements based on Requirements Engineering

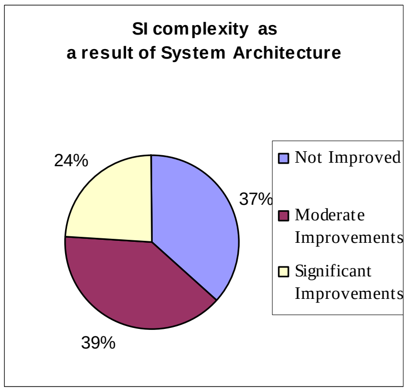


Figure 9 SI process improvements based on System Architecture

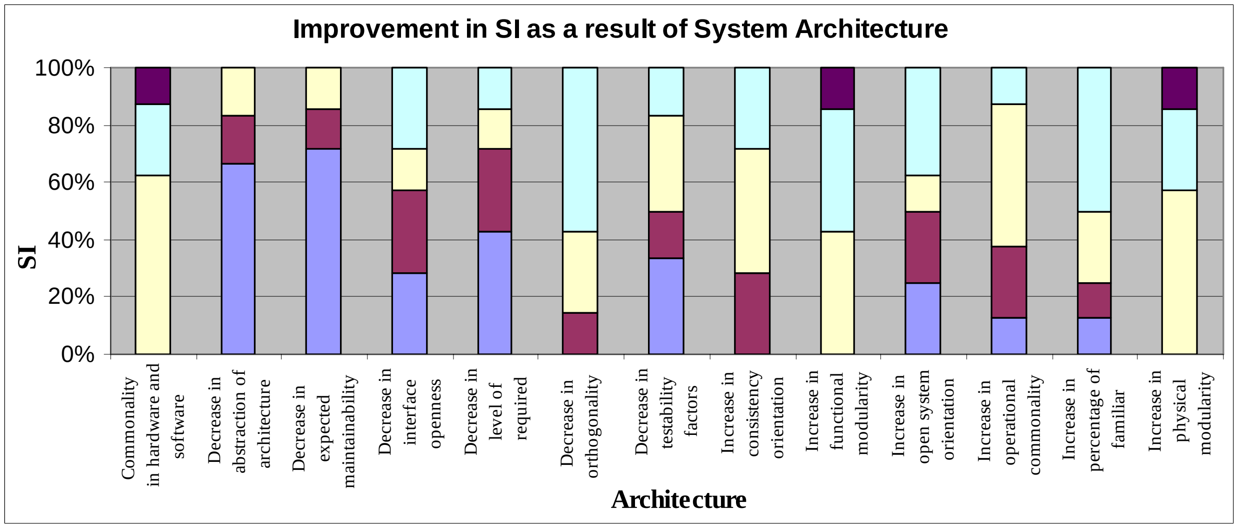


Figure 10 SI process improvements based on specific System Architecture activities [Legend same as Figure 7]

Table 4 Quality of Verification and Validation: Survey Results

#	Verification and Validation is improved as a result of	Not Improved (%)	Improved (%)			
			Partially	Moderately	Significantly	Exceptionally
1	Categorization of system requirements based on its type	25	13	38	13	13
2	Clear identification of the system requirements	0	0	38	25	38
3	Commonality in hardware and software subsystems	0	0	75	25	0
4	Decrease in abstraction of the system architecture	50	33	0	0	17
5	Decrease in expected maintainability	43	29	14	0	14
6	Decrease in interface openness	33	17	33	17	0
7	Decrease in level of required reliability	43	14	14	14	14
8	Decrease in orthogonality	17	0	17	50	17
9	Decrease in testability factors	33	0	50	0	17
10	Increase in consistency orientation	14	29	29	29	0
11	Increase in functional modularity	17	17	33	17	17
12	Increase in open system orientation	25	38	13	25	0
13	Increase in operational commonality	0	13	50	25	13
14	Increase in percentage of familiar technology	13	13	38	25	13
15	Increase in physical modularity	0	17	33	17	33
17	Prioritization of system requirements	13	25	25	13	25
18	System requirement feasibility analysis	25	25	25	25	0

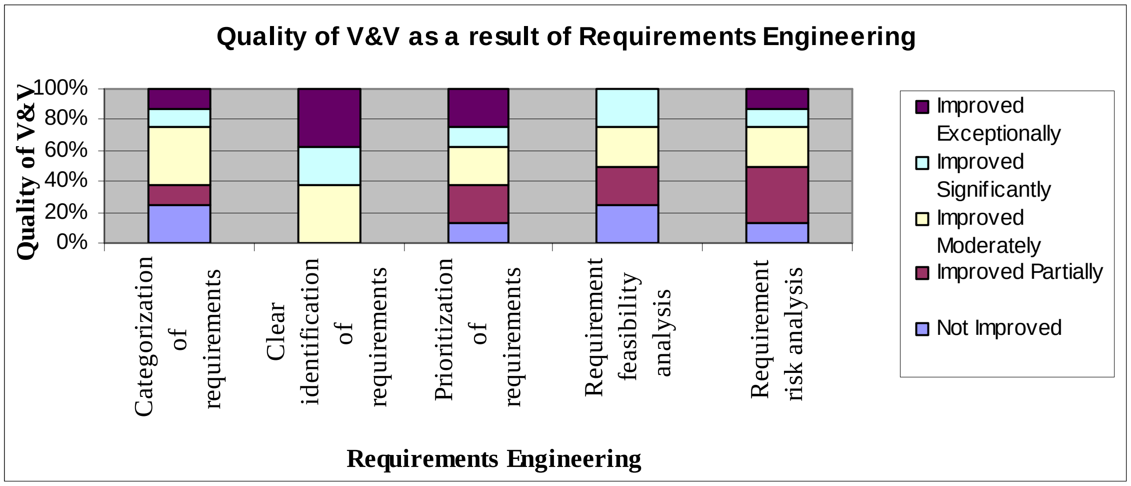


Figure 11 V&V quality improvements based on specific Requirements Engineering activities

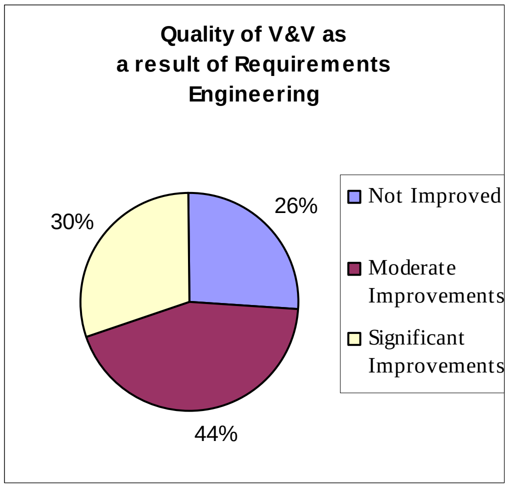


Figure 12 V&V quality improvements based on Requirements Engineering

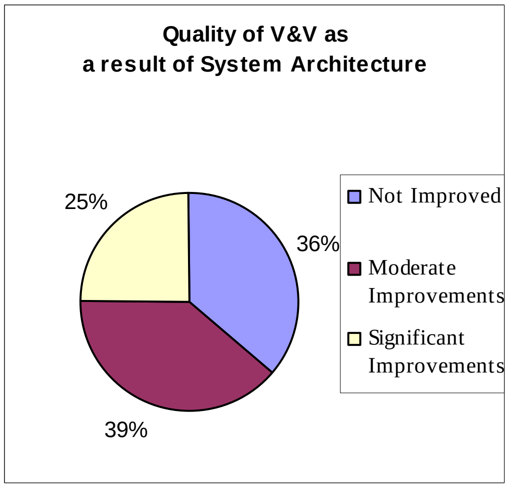


Figure 13 V&V quality improvements based on System Architecture

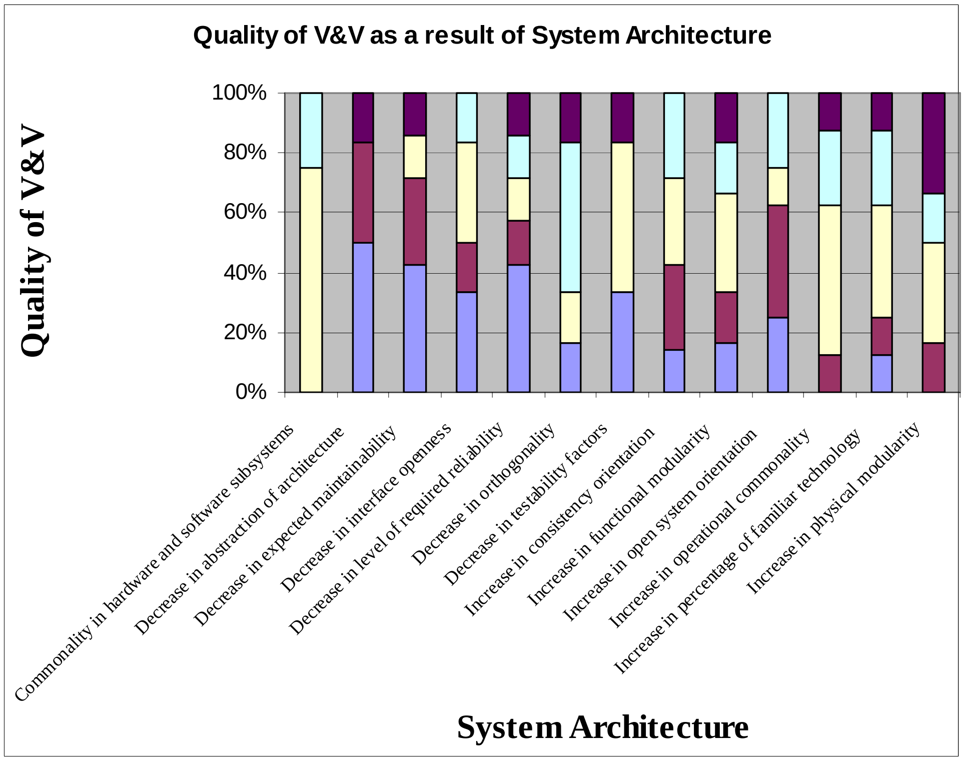


Figure 14 V&V quality improvements based on specific System Architecture activities
[Legend same as Figure 11]

Significant Cause and Effect Relationships

The results of the survey were analyzed based on the research questions. The following sections provide a consolidated view of this analysis and synthesis. The significant cause and effect relationships between system requirements, system architecture, integration process complexity, and quality of V&V are shown in the fishbone charts shown in Figure 15 and Figure 16.

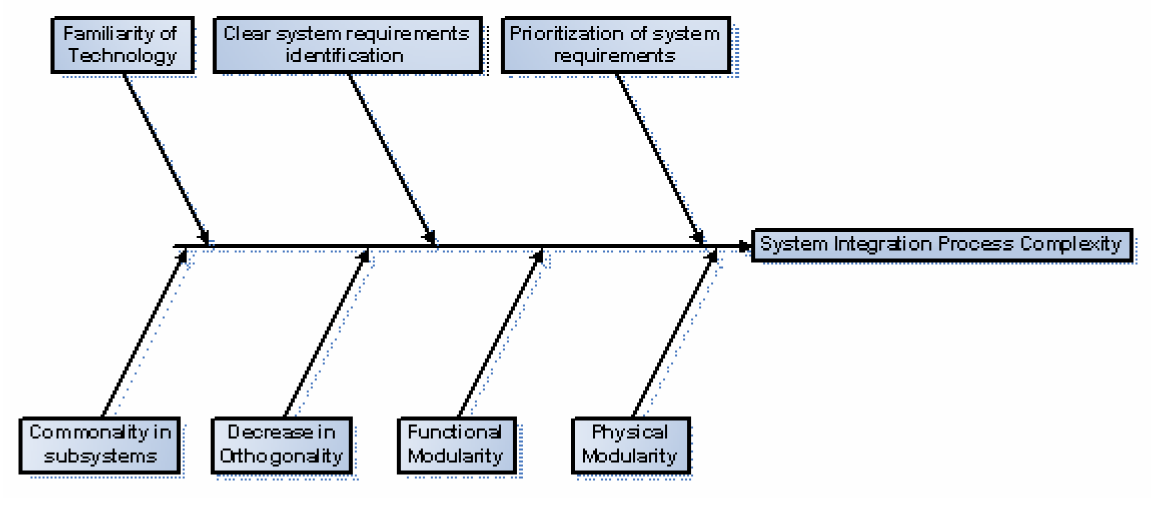

Familiarity of Technology
Clear system requirements identification
Prioritization of system requirements
System Integration Process Complexity
Commonality in subsystems
Decrease in Orthogonality
Functional Modularity
Physical Modularity

Figure 15 Major Factors Impacting SI process complexity

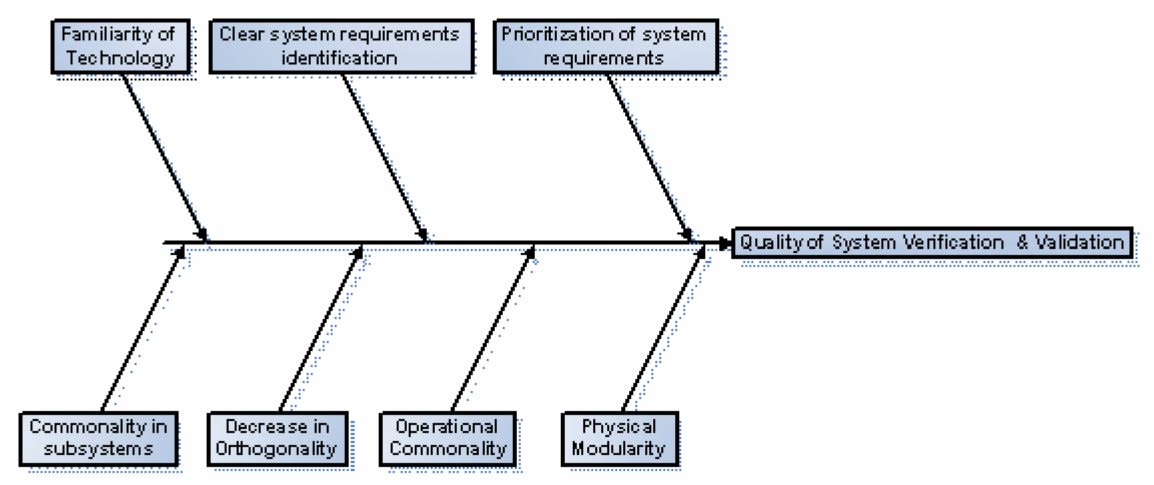

Familiarity of Technology
Clear system requirements identification
Prioritization of system requirements
Quality of System Verification & Validation
Commonality in subsystems
Decrease in Orthogonality
Operational Commonality
Physical Modularity

Figure 16 Major Factors Impacting Quality of System Verification & Validation

Impacts of Requirements Engineering on SIC and Quality of V&V

The results of the survey confirmed the belief that an improved requirement engineering (RE) process results in reduced system integration complexity and a higher quality requirements verification and validation. The improved RE process will result in significant improvements by reducing system integration complexity and improving the quality of V&V. The significant system requirements factors that result in reducing the system integration complexity and in improving the quality of verification and validation are discussed below.

<u>Clear identification of system requirements:</u>

There has been a good amount of research and improvements in the field of requirements engineering. The major focus area of these research, methods and tools is the ambiguity and volatility associated with the system requirements. Subject + Verb + Modifier format is commonly used for clear system requirements. Traceability and testability of requirements have to be concentrated during requirements elicitation and analysis. A number of tools on the market today help system engineers trace requirements and specify the associated test requirements. But the traceability in most of these tools ends during the requirements phase. Even though there is a good traceability between the originating and derived requirements, the benefits are limited when there is no traceability across all phases of development. Stakeholder involvement throughout the system development is a key to identify clear system requirements and transition them effectively across the system lifecycle.

By performing this activity the system capabilities and functions can be clearly identified. This activity impacts every phase of system integration (derive integration requirements, develop integration architecture, plan integration, implement based on the architecture and plan, and verify and validate the system and its interfaces) and significantly impacts the technical complexity of integration. By identifying clear requirements, the cost overruns and schedule slippage can be avoided due to the requirements ambiguity.

Therefore, requirements traceability also impacts programmatic complexity. By identifying clear operational support and availability requirements and other constraining requirements the risks associated feasibility and effort required can be analyzed up in the lifecycle and can be mitigated. The result is lower integration operational complexity and organizational complexity.

<u>Prioritization of system requirements:</u>

During system development concentrating on value added activities is very important. In order to provide the value added activities the system requirements need to be prioritized earlier in the lifecycle. One of the key success criteria for evolutionary or iterative development is prioritization of system requirements. The effort required for integration, verification and validation can be planned and executed effectively by performing this activity early in the lifecycle with stakeholder participation. By prioritizing and concentrating on core functionalities/ capabilities first and building upon them can also help in configuration management. This activity helps reduce both the technical and configuration complexity of integration.

Impacts of Systems Architecture on SIC and Quality of V&V

The results of the survey suggest that there will be a significant improvement in the system integration complexity and the quality of V&V by adopting some of the suggested system architecture practices. Current systems engineering researchers focus on architectures that are integration friendly. The mode of system development has shifted away from being manufacturing-centric to being integration-centric due to the fact that use significant percentage of COTS, strategic outsourcing, value based capabilities development, and need for supply chain excellence. Another architecture trend is the migration toward service oriented architectures (Service Oriented Architectures focused around supportability of operational scenarios) and event driven architecture which are even more integration friendly [Kumar, et. al., 2005], [Krishnamoorthy, et. al., 2005] The factors of system integration process complexity are addressed earlier in the lifecycle and addressed as a part of the architecture by adopting these integration friendly architecture methodologies. Some of the system architecture factors that impact system integration process complexity evolve when these architecture methodologies are adopted. Future research based on these research findings has been proposed at Stevens Institute of

Technology, and should result in practices, methods, and tools that will aid in creating system architectures which utilize patterns at the system and subsystem levels, and displaying high degrees of modularity.

Good architecture should exhibit characteristics such as Time Sensitivity, Context Sensitivity, and Stakeholder Sensitivity. Based on the survey results the factors of system architecture that impacted system integration process complexity and quality of system verification and validation the most and results in at least partial improvement are

Commonality in hardware and software subsystems: This system architecture factor emerged to be an important factor impacting both system integration process complexity and quality of system verification and validation. Commonality in subsystems helps reduce the effort required for integration, verification and validation. The degree of unique interfaces, platforms, technology used, and protocols are reduced resulting familiarity of subsystems. This reduces the technical complexity associated with system integration process.

Increase in physical modularity: Quality of verification and validation can have at least moderate improvements and System integration process complexity can be at least partially reduced by increasing the physical modularity in system architecture. The physical and functional modularity in architecture facilitates both modernization and replacements of legacy systems. This significantly reduces the complexity associated with the legacy system integration. Modularity also results in higher levels of maintainability and support. This reduces the technical complexity associated with system integration process.

Increase in functional modularity: System integration complexity can be reduced significantly by increasing functional modularity in system architecture. The increase in functional modularity in system architecture is a major contributor to rapidity in system development by adding ease of building upon or upgrading the existing functionalities. This factor is a key in evolutionary and iterative system developments. The risk associated with system architecture during rapid system development would also be reduced by addressing functional modularity. This reduces the technical complexity associated with system integration process.

Increase in operational commonality: Quality of verification and validation can be improved significantly by increasing the operational commonality. Commonality is the extent to which the system is made up of common hardware and software components, utilizes familiar technologies, and is automated reducing the effort of training and maintenance. Operational commonality in system architecture results in operational requirements that can be easily verified and validated. Automation also reduces the effort involved in V&V.

Decrease in orthogonality: This factor of system architecture results in improvements in both quality of verification and validation and system integration process complexity. Orthogonality of system architecture can be reduced by performing one-to-one functional mapping. This factor reduces both the technical and configuration complexities of system integration process.

Increase in percentage of familiar technology: By having familiar technologies in system architecture quality of V&V and system integration process can be improved moderately. This can be achieved by adopting good product line architectures and nested layer architecture. This factor reduces the programmatic and configuration complexities of system integration process.

Functional modularity as a characteristic of system architecture plays an important role in reducing system integration complexity. However, it does not seem to improve system verification and validation. Also we can observe that operational commonality plays an important role in improving system verification and validation but not in reducing system integration complexity. These differences could be attributed to the nature of the tasks involved in system integration and system verification and validation. Modularity is the extent to which the system is made up of well defined, functionally non-overlapping, modular elements with well documented interfaces allowing updates to or replacements of a portion of the system without affecting the remainder of the system. Functional modularity in system architecture results in standard functional requirements fragmented across multiple processing elements and interfaces. It also enables adding new functionality with minimum disruption. By doing so system integration is simplified.

The results also indicate that the most of the factors that impact system integration complexity also impact quality of system verification and validation. By adopting these activities in system requirements and architecture critical issues of system development are addressed early in the lifecycle. This provides more bandwidth for mitigating the risks associated with these issues. By doing so adverse consequences of these risks are understood and addressed resulting in a more likely successful system development and operation. Normally there are no system operational effectiveness and total cost of ownership tradeoffs being performed during these activities. Therefore, one might decide there is no need to perform optimization of system operational effectiveness and total cost of ownership when adopting these activities. However, the results presented in this paper indicate these activities could result in reduced total cost of ownership and improved system operational effectiveness.

Summary

The specific nature of application-domain plays a critical role in the impact of the system architecture and system requirements related factors on system integration process complexity. The findings of this study were based on the survey of one government organization. They indicate that attention to key architecture and requirements attributes during the early development activities can have significant impact during systems integration, while resulting in reduced systems integration complexity. Further research to extend the study across domains and industry sectors is required to generalize the findings.

References

1. Bachmann, F., Bass, L., Chastek, G., Donohoe, P., Peruzzi, F., "The Architecture Based Design Method", Software Engineering Institute, Carnegie Mellon University (SEI-CMU), 2000.
2. Bachmann, F., Bass, L., Klein, M., "Deriving Architectural Tactics: A Step Toward Methodical Architectural Design", SEI-CMU, 2003.
3. Bachmann, F., Bass, L., Klein, M., "Illuminating the Fundamental Contributors to Software Architecture Quality", SEI-CMU, 2002.
4. Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison-Wesley, 2003.
5. Buede, D., "The Engineering Design of Systems", Wiley Series in Systems Engineering, 2000.
6. Carney, D., Oberndoff, P., "Integration and Interoperability Models for System of Systems", Carnegie Mellon University, 2004.
7. Clements, P., Kazman, R., Klein, M., "Evaluating Software Architectures:

Methods and Case Studies", Addison-Wesley, 2001.

8. Cloutier, R., "Applicability of Patterns to Architecting Complex Systems", Doctoral Dissertation, Stevens Institute of Technology, 2006.

9. Cloutier, R., Verma, D., "Applying Pattern Concepts to Systems (Enterprise) Architecture", Journal of Enterprise Architecture, May 2006.

10. Cloutier, R., Verma, D., "Applying the concept of patterns to systems architecture", Systems Engineering 10(2): 138-154, 2007.

11. Dogru A., Delcambr, S., Bayrak, C., Christiansen, M., Tanik, M., "The development of an integrated system design environment", Proceedings of the second international conference on systems integration, ICSI, 1992, pp. 691-698.

12. Elias, G., Jain, R., "Exploring Attributes for Systems Architecture Evaluation", CSER, 2007.

13. Evans, M. W., Marciniak, J., "Software Quality Assurance and Management", John Wiley & Sons, 1987.

14. Howard, N., Rolland, C., Qusaibaty, A., "Process complexity: Towards a theory of intent-oriented process design", INFOS, 2004.

15. IEEE 1233, "IEEE guide for developing system requirements specifications", IEEE, 1998.

16. IEEE 610-12, "IEEE Standard Glossary of Software Engineering Terminology", IEEE Computer Society, 1990.

17. IEEE 830, "IEEE recommended practice for software requirements specifications", IEEE, 1998.

18. ISO/IEC 15939, "Software engineering – Software Measurement Process", 2002.

19. Jain, R., Chandrasekaran, A., Erol, O., "A System Integration Process Model (SI^{PM})", Working Paper, Stevens Institute of Technology, 2007a.

20. Jain R., Erol, O., Chandrasekaran, A., "Proposing a System Integration Readiness Framework", Working Paper, Stevens Institute of Technology, 2007b.

21. Jorgensen, R., Philpott, I., "Architectural Abstractions", INCOSE Symposium, 2002.

22. Kazman, R.., Nord, R.., Klein, M., "A Life-Cycle View of Architecture Analysis and Design Methods", SEI-CMU, 2003.

23. Krishnamoorthy, V., Unni, N.K., Niranjan, V., "Event-driven service-oriented architecture for an agile and scalable network management system", IEEE Conference on Next Generation Web Services Practices, August 2005.

24. Kumar Harikumar, A., Lee, R., Chia-Chu C., Hae-Sool Y., "An event driven architecture for application integration using Web services", IEEE Conference on Information Reuse and Integration, August, 2005.

25. McCabe, R., Pollen, M., "Evaluating Architectures With System Attributes", Software Productivity Consortium, 2004.

26. Mil-Std-1388-1A, "Logistics Support Analysis", DOD, Pars. 20, April 1983.

27. Open Systems Joint Task Force, Open Systems Joint Task Force, http://www.acq.osd.mil/osjtf/, 2005.

28. Rossak, W., Prasad, S., "Integration Architectures – a framework for systems integration decisions", Proceedings of the IEEE international conference on systems, man, cybernetics, 1991, pp. 545-550.

29. Smith, D., O'Brien, L., Kontogiannis, K., Barbacci, M., "The Architect: Enterprise Integration", SEI News, Vol. 5, No. 4, 2002.

30. Software Engineering Institute (SEI), Carnegie Mellon, "Quality Measures Taxonomy", http://www.sei.cmu.edu/str/taxonomies/

qm_tax_body.html, 2006.

31. Systems Engineering Handbook v3, INCOSE, 2006.

32. Verma, D., Johannesen, L.H., “Supportability assessment and evaluation during system architecture development”, INCOSE Symposium, Minneapolis, MN, July 2000, pp. 699–706.

Biography

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. Dr. Jain is currently the Head of Education and Research for International Council of Systems Engineering (INCOSE). Her teaching and research interests include systems integration, systems architecture and design, and rapid systems engineering. Dr. Jain is 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.

Anithashree Chandrasekaran is a Doctoral Candidate in the School of Systems and Enterprise at Stevens Institute of Technology. Her research interests includes Rapid Systems Development and its processes, Development process reengineering, Risk Management and Modeling, System Integration, System Design and Architecture. She obtained her B.E. in Electrical and Electronics Engineering from P.S.G. College of Technology, India. She obtained her M.S. in Systems Engineering from Stevens Institute of Technology. She is the president of Stevens INCOSE student chapter.

George M. Elias is a Systems Engineering Doctoral Candidate at Stevens Institute of Technology. Mr. Elias is a Systems Engineer 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.

Robert Cloutier (Rob) has over 20 years experience in systems engineering, software engineering, and project management in both commercial and defense industries. His research interests include Systems Engineering Patterns and modeling complex systems with UML/SysML, Reference Architectures, and agent based technology applied to systems engineering. Rob received his Ph.D. in Systems Engineering from Stevens Institute of Technology, and also holds an M.B.A. from Eastern University, and a B.S. from the United States Naval Academy. He is an Adjunct Professor for Eastern University and chairs the Rowan University Electrical and Computer Engineering Department Industry Advisory Board. Rob belongs to the International Council on Systems Engineering (INCOSE), is a member of the Technical Leadership Team. He is also an active member of the Association of Enterprise Architects, and IEEE. Finally, he and his wife raise puppies for The Seeing Eye to serve as guide dogs to the blind.

[i] Dr. Rashmi Jain is the primary author of this paper. All questions and comments should be addressed to her

[ii] Adapted from SYS 625 course notes, SEEM, Stevens Institute of Technology, 2005