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

Rashmi Jain, Anithashree Chandrasekaran, George Elias, and Robert Cloutier, *Member, IEEE*

***Abstract*—The need to perform faster systems integration of complex systems require the architect and design team to understand how the selected architecture and design components will impact the systems integration processes complexity (or difficulty). Systems 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 systems 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 systems 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, subsystems, processes, and interfaces impacts (and to what extent) systems integration process complexity. This paper also defines and analyzes the impact of the different system architecture and requirements factors on systems integration process complexity. A research framework is developed to understand the cause-and-effect relationships between 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 eight development projects in the government sector.**

***Index Terms*—Integration complexity, system architecture, systems integration, system requirements.**

I. Introduction

IT IS necessary to determine if integration of the defined physical elements and interfaces in system architecture contributes to systems integration (SI) process complexity. Today's systems are usually expected to function independently and in cooperation with the existing systems [system-of-systems (SoS)]. The cooperating systems undergo frequent changes and interoperability becomes a key factor and a major contributor to systems integration process complexity. Interoperability is based on the existence of the conceptual view [1]. The conceptual view can be embodied in requirements and architecture/design, and the system architecture determines the level of interoperability. The systems integration process complexity includes interoperability.

Manuscript received June 19, 2007; revised April 4, 2008.

R. Jain, A. Chandrasekaran, and R. Cloutier are with the School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ 07030 USA (e-mail: rashmi.jain@stevens.edu; anithashree.chandrasekaran@stevens.edu; robert.cloutier@stevens.edu).

G. Elias is with ITT Electronic Systems, Electronic Warfare Systems, Clifton, NJ 07014 USA, and also with Stevens Institute of Technology, Hoboken, NJ 07030 USA (e-mail: gelias@stevens.edu).

Digital Object Identifier 10.1109/JSYST.2008.924130

The major focus of this paper is on the cause-and-effect relationship between system requirements, system architecture, and systems integration process complexity. This paper is based on 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 projects examined were large government projects and the project teams approached systems engineering tasks using functional decomposition. While object oriented approaches to systems engineering is gaining interest, most systems engineering of complex systems is still performed using the more traditional structured analysis, sometimes called functional decomposition. The scope of this paper is to analyze the outcome of those projects rather to comment on the viability of the functional decomposition approach.

The research methodology used for this study is provided in Fig. 1. This paper is organized in a similar outline, first establishing the context by defining and discussing systems 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 systems integration process complexity, and improve the quality of system verification and validation (V&V). 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.

II. Systems Integration Process Complexity

Systems Integration 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

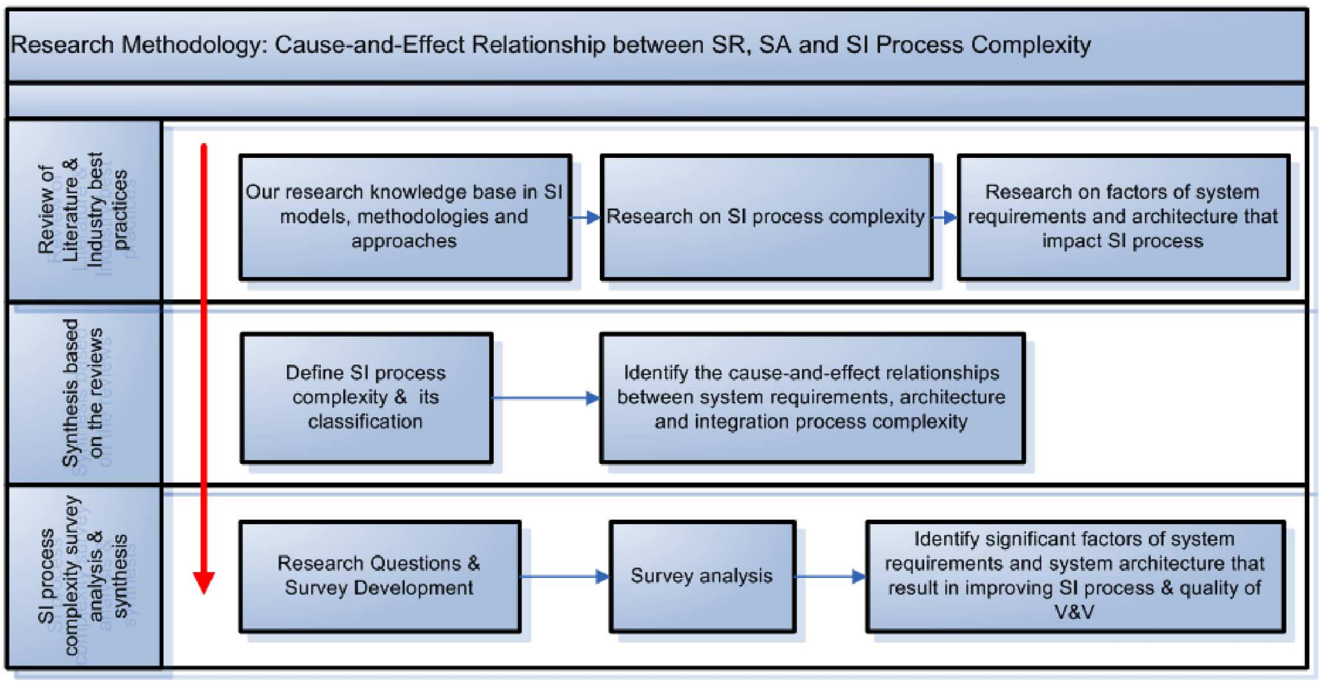

Research Methodology: Cause-and-Effect Relationship between SR, SA and SI Process Complexity
Review of Literature & Industry best practices
Our research knowledge base in SI models, methodologies and approaches
Research on SI process complexity
Research on factors of system requirements and architecture 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

Fig. 1. Research methodology.

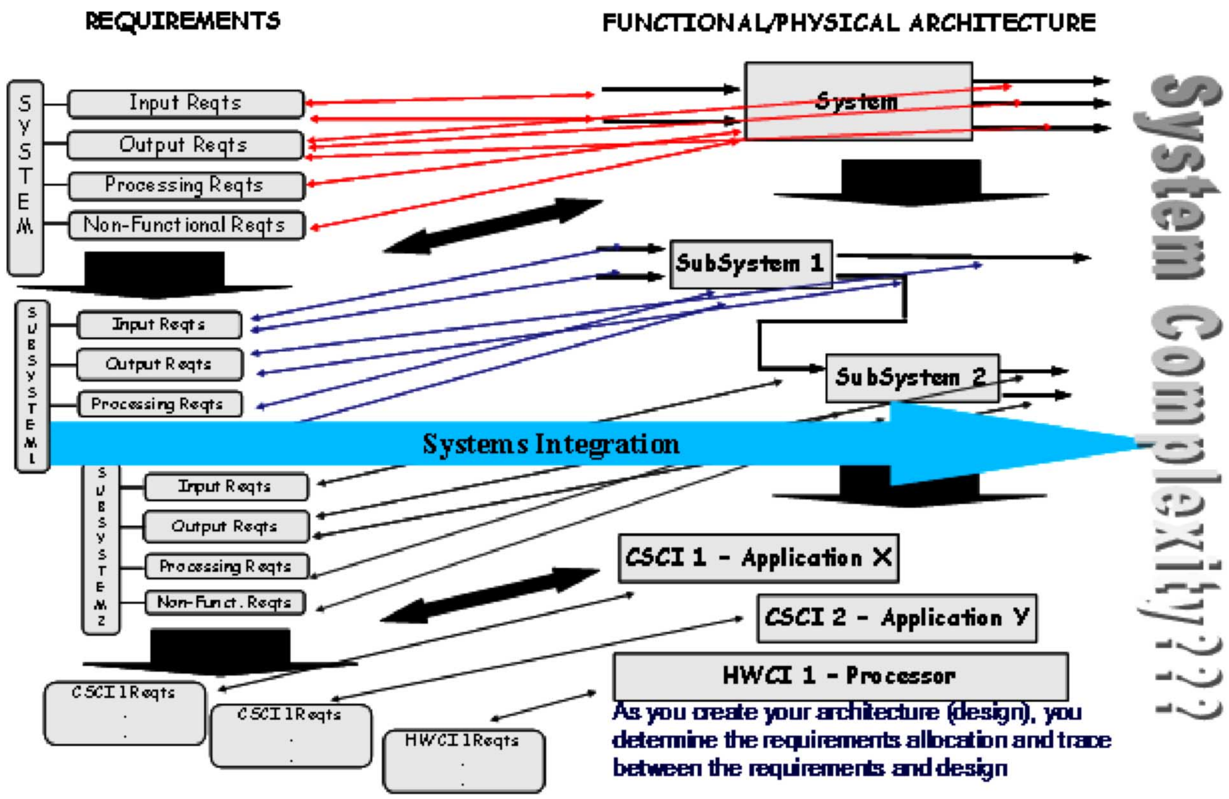


Fig. 2. Systems integration [3].

and validation and ultimately implementation of the system as shown in the Fig. 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 [2]. 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.

Fig. 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 [4], Rossak and Prasad state that integration architectures for SoS 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 systems integration process in the development of a system and its successful implementation can be understood and appreciated by understanding and familiarizing with these systems integration process activities. Despite the large investments that are made, many integration efforts fail for a number of technical, organizational, managerial, and migration planning reasons. The technical issues include the following [5]:

- 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;
- legacy systems were not initially designed for new quality-attribute requirements and are being affected by needs for interoperability, performance, security, and usability;
- 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 systems 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 [6]. 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 [7]. Any measurement of process complexity requires measurement of situation complexity, action complexity, and intention complexity [8].

We define as follows.

Systems 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 systems integration process for a given system development and operational environment.

TABLE I
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	Systems integration Planning, Control and Management
Scope of Commercial Off The Shelf (COTS) requirements (interface and interoperability)	Documentation of systems 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 systems 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?)
	Degree of automation of the SI process
	Number of dependencies for SI process activities (Degree of concurrency of the SI process)

Some of the factors that impact the degree of feasibility and the level of effort required to understand, describe, implement, manage, and document the systems integration process activities are shown in Table I. This list is based on our systems integration framework and systems integration process model (SI^{PM}) [9], [10].

Most of the research discussions on systems integration complexity are in the context of the complexity associated with the entire system. Currently, systems integration complexity is mainly focused on the degree of testing required and interface complexity. But there is a need to look at systems 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 systems integration process complexity as technical, programmatic, configuration,

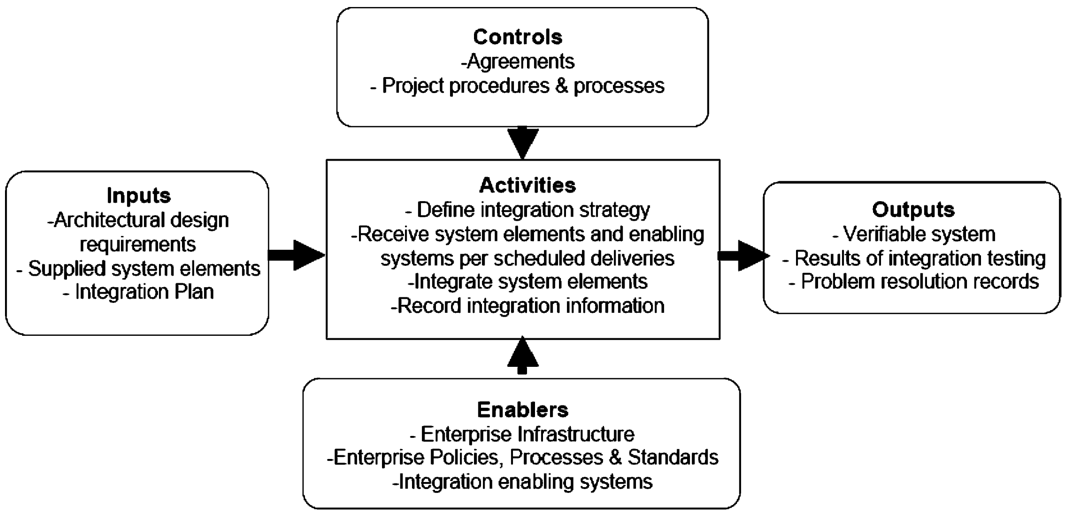

Controls
-Agreements
- Project procedures & processes
Inputs
-Architectural design requirements
- Supplied system elements
- Integration Plan
Activities
- Define integration strategy
-Receive system elements and enabling systems per scheduled deliveries
-Integrate system elements
-Record integration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

Fig. 3. Context diagram for the integration process [11].

operational, and organizational. This classification is based on how the system life cycle and the development activities impact the systems integration process. These are described in the following.

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

Configuration Complexity: The complexity of systems 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 systems 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 subsystems.

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

III. Designing For Integration

Most of the factors that result in systems integration process complexity are external to the context of systems integration process. The systems integration processes and activities receive most of its inputs, controls and triggers from system requirements and system architecture (Figs. 3–5). This highlights the direct cause-and-effect relationship between the system requirements, architecture, and systems integration. System requirements and architecture has a direct impact on systems 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 systems integration. System architecture can play a direct causer or a moderator or a mediator depending on the system context and concept.

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 [12]. System attributes are a key link between good system architecture and the system's ultimate development cost and schedule [13]. 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 [14]. The set of attributes shown in Fig. 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 systems integration issues can be contained and controlled effectively resulting in a relatively simplified systems integration process. A comprehensive list of attributes of architecture can be obtained from SEI Quality Measure Taxonomy [12], [15], [16].

An attribute is a "property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means [17]." The attributes of a system or architecture are used to describe the properties of the system or

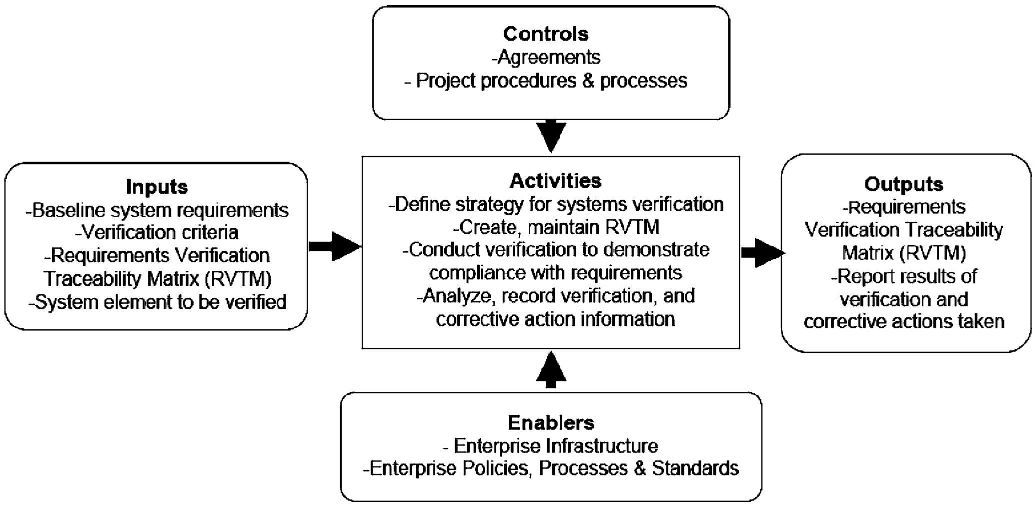

Controls
-Agreements
- Project procedures & processes
Inputs
-Baseline system requirements
-Verification criteria
-Requirements Verification Traceability Matrix (RVTM)
-System element to be verified
Activities
-Define strategy for systems verification
-Create, maintain RVTM
-Conduct verification to demonstrate 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

Fig. 4. Context diagram for the verification process [11].

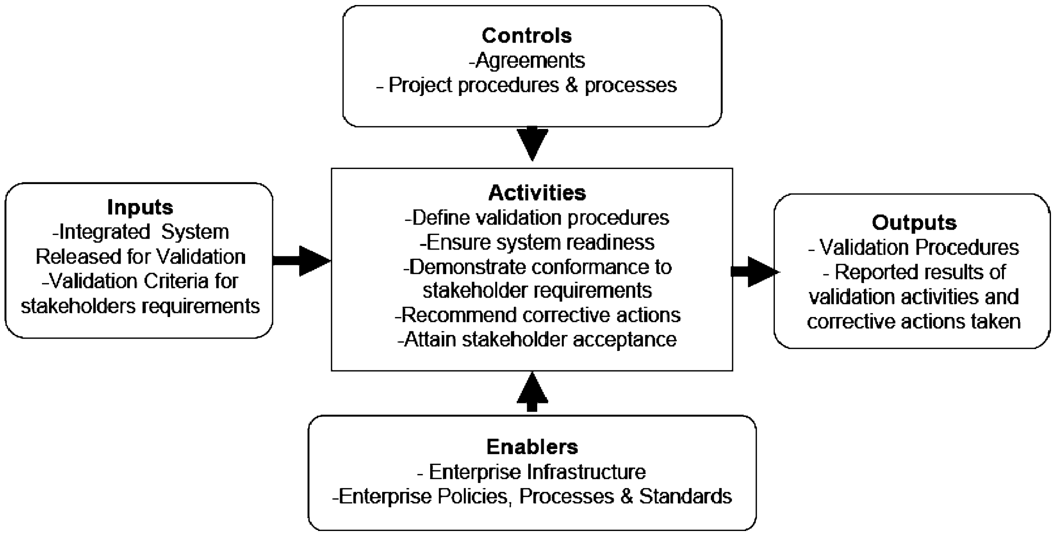

Controls
-Agreements
- Project procedures & processes
Inputs
-Integrated System Released for Validation
-Validation Criteria for stakeholders requirements
Activities
-Define validation procedures
-Ensure system readiness
-Demonstrate conformance to stakeholder requirements
-Recommend corrective actions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

Fig. 5. Context diagram for the validation process [11].

the system architecture in a unique distinguishing manner. Attributes are ideal for describing and monitoring many systems engineering tasks because they are measurable [17]. The use of attributes to measure and evaluate systems and systems architecture is valuable in making decisions and tradeoffs [14]. In this paper, we focus on important attributes that contribute to systems integration process complexity. By understanding which attributes of system architecture affect systems integration complexity one could use this information throughout the requirements, architecting, and development phases of the system life cycle.

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

Commonality refers to the system 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 [6]. 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 replaceable 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, and 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), moder-

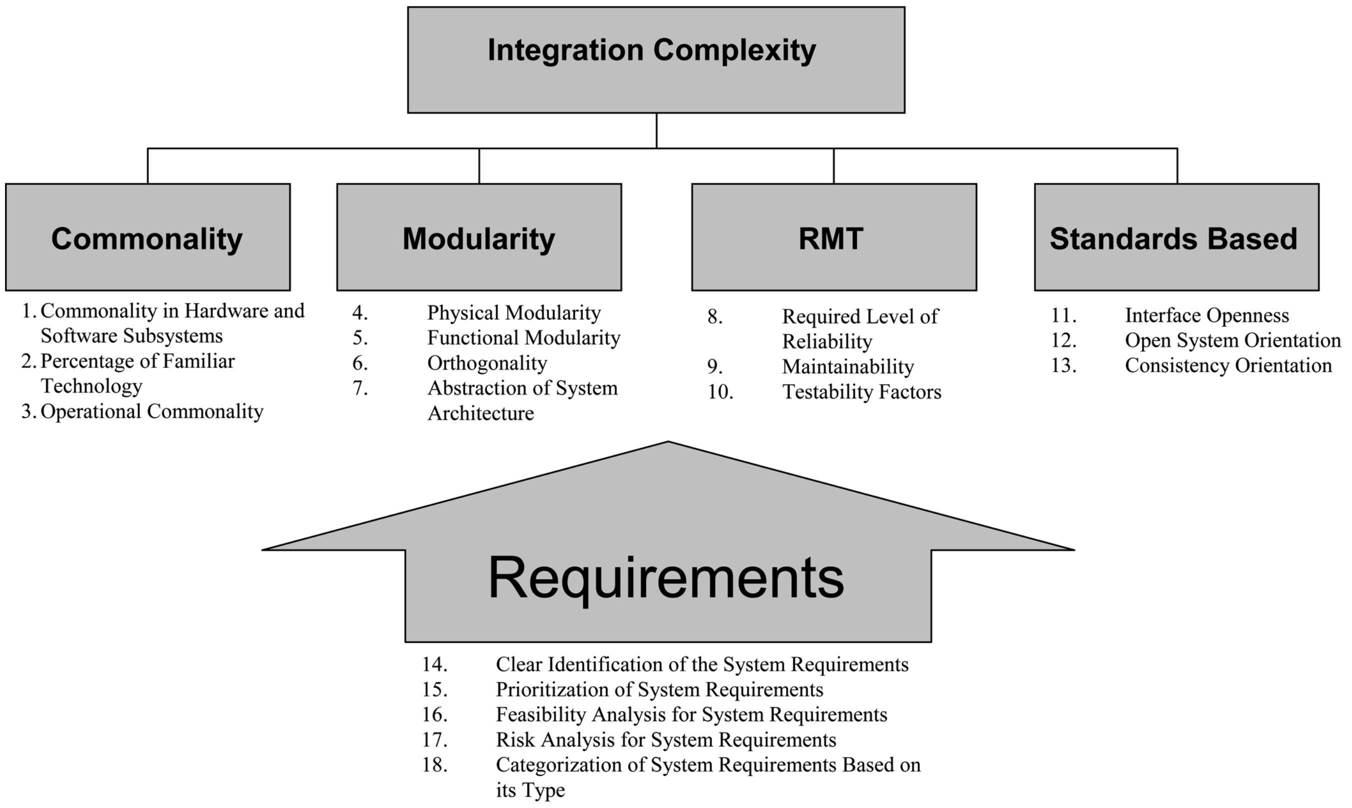

Integration Complexity
Commonality
Modularity
RMT
Standards Based
1. Commonality in Hardware and Software Subsystems
2. Percentage of Familiar Technology
3. Operational Commonality
4. Physical Modularity
5. Functional Modularity
6. Orthogonality
7. Abstraction of System Architecture
8. Required Level 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

Fig. 6. Architectural attributes that impact systems integration complexity.

ating design complexity and enhancing its clarity, and enabling design and functional flexibility and variety for the system as a whole [13], [18]. 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 [19].

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 [20].

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 [6]. 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 [6]. Examples of issues that affect testability: number of LRUs covered by built-in test (BIT), 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 test points, and 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 life cycle. The decisions made through detailed design commit approximately 80% of the cost of the system [21]. 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 LRUs), 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, and 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, and estimated maintenance training time);
4) physical modularity (such as ease of system element upgrade and ease of operating system upgrade);
5) functional modularity (such as ease of adding new functionality and ease of upgrade existing functionality);
6) orthogonality (examples are functional requirements fragmented across multiple processing elements and interfaces, throughput requirements across interfaces, and common specifications identified);
7) abstraction of system architecture;
8) required level of reliability (factors such as fault tolerance, and percentage of system loading [processor loading, memory loading, network loading, etc.)];
9) maintainability in terms of expected MTTR, 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, and automated input/stimulation insertion);
11) interface openness (such as number of interface standards/ interfaces, multiple vendors, multiple business domains, and 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, and testing).

Some of the significant sources of the causality analysis include [22], Architecture Tradeoffs Analysis Method (SEI ATAM), IBM Architecture Evaluation Methodology (IBM AEM), [23]–[28], and INCOSE SE Handbook v3 [11]. These system architecture and requirements activities when implemented can result in certain clearly observable patterns in system architecture and requirements [29], [30]. 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 [31]. 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 leveraged 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 systems 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 systems integration process complexity categories. The mapping is shown in Table II.

IV. Formulating Hypothetical Systems Integration 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 systems 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 systems integration process and reduce the dependencies between these development phases. Addressing some of the systems 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 systems integration activities and processes is initiated early in the life cycle, feedback and iterations during development helps to refine and improve systems integration strategies, planning and quality.

The two factors (dependent variables) considered for this research study were the systems integration complexity (systems

TABLE II
SYSTEM ARCHITECTURE AND REQUIREMENTS CAUSE AND EFFECT ON SI 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

integration process complexity) and quality of system verification and validation. The most important phase of systems integration is the system verification and validation. Verification and validation are the phases of system engineering where the required functionality comes together with all the interfaces completed. Hence, studying the quality of verification and validation along with the systems integration complexity is important and provides a comprehensive view of systems integration. The independent variables are the system architecture and requirements factors discussed in Section III. We formed the following research questions to understand the cause-and-effect relationships between system architecture and requirements with systems integration process complexity.

1) *Null Hypothesis:* There is a cause-and-effect relationship between system requirements, system architecture, and systems 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 systems 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 systems integration? Hypothesis: System architecture improvements can reduce systems integration complexity.
6) What are the three most important architectural factors for systems 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 of system reliability and maintainability impact the complexity of systems integration and the quality of verification and validation? Hypothesis: Higher levels of reliability and maintainability in system design reduce the complexity of systems integration and increase the quality of verification and validation.
10) Does an improved requirement engineering process results in reduced systems integration complexity and better quality of verification and validation?
11) Does familiarity of technology lead to reduced systems integration complexity and better quality of verification and validation?

A survey questionnaire addressing each of these identified system architecture and requirements factors was developed.

TABLE III
SI 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

TABLE IV
QUALITY OF V&V: 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

The survey questions are designed to verify if there is a cause-and-effect relationship between each of the identified 18 factors on systems 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 system engineering 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 eight different development projects. The survey responses are shown in Tables III and IV.

Fig. 7 illustrates improvements at five different levels in the systems integration process complexity on the vertical axis as a result of the requirements engineering factors listed on the horizontal axis. For example, clarity in identification of require-

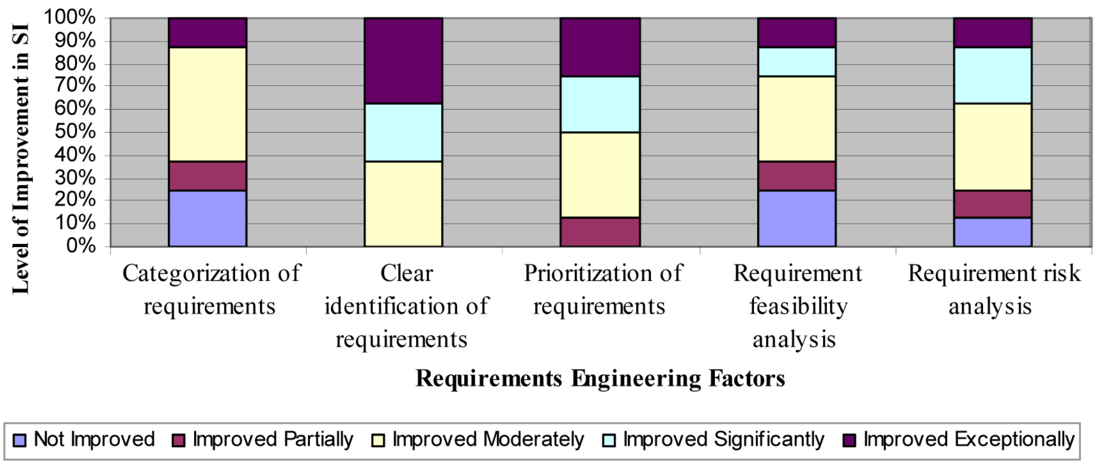


Fig. 7. SI process improvements as a result of requirements engineering activities.

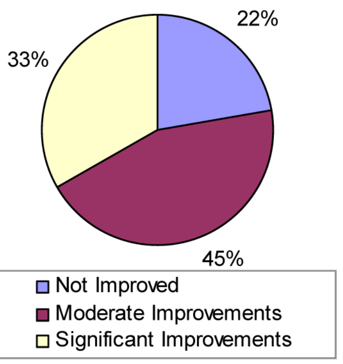


Fig. 8. SI process improvements as a result of requirements engineering.

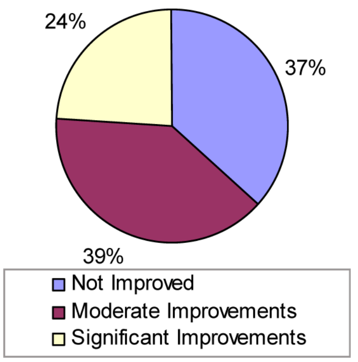


Fig. 9. SI process improvements as a result of system architecture activities.

ments (second bar from left) results in exceptional improvement in systems integration process in 38% of the projects. Figs. 8 and 9 show the percentage of projects where improved requirements engineering and system architecture activities resulted in three levels of improvements in systems integration process. Fig. 10 illustrates improvements at five different levels in the systems integration process complexity on the vertical axis as a result of the system architecture factors listed on the horizontal axis. For example, increasing physical modularity in system architecture (first bar from right) results in exceptional improvement in systems integration process in 14% of the projects.

Figs. 11 and 14 depict the survey responses on the level of improvement in the quality of verification and validation across five requirements engineering and 13 system architecture factors, respectively. These factors are labeled on the horizontal axis on the bar chart. The responses are normalized in order to show the percent of how much the quality of verification and validation improved and is shown in the vertical axis on the bar chart. For example, in Fig. 11, prioritizing system requirements (third bar from left) results in exceptional improvement in quality of verification and validation in 25% of the projects and in Fig. 14, increasing physical modularity in system architecture (first bar from right) results in exceptional improvement in quality of verification and validation in 33% of the projects. Figs. 12 and 13 show the percentage of projects where improved requirements engineering and system architecture activities resulted in three levels of improvements in the quality of verification and validation.

V. 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 verification and validation are shown in the fishbone charts in Figs. 15 and 16.

A. Impacts of Requirements Engineering on System Integration Process Complexity and Quality of Verification and Validation

The results of the survey confirmed the belief that an improved requirement engineering process results in reduced systems integration complexity and a higher quality system verification and validation. The improved requirements engineering process will result in significant improvements by reducing systems integration complexity and improving the quality of verification and validation. The significant system requirements factors that result in reducing the systems integration complexity

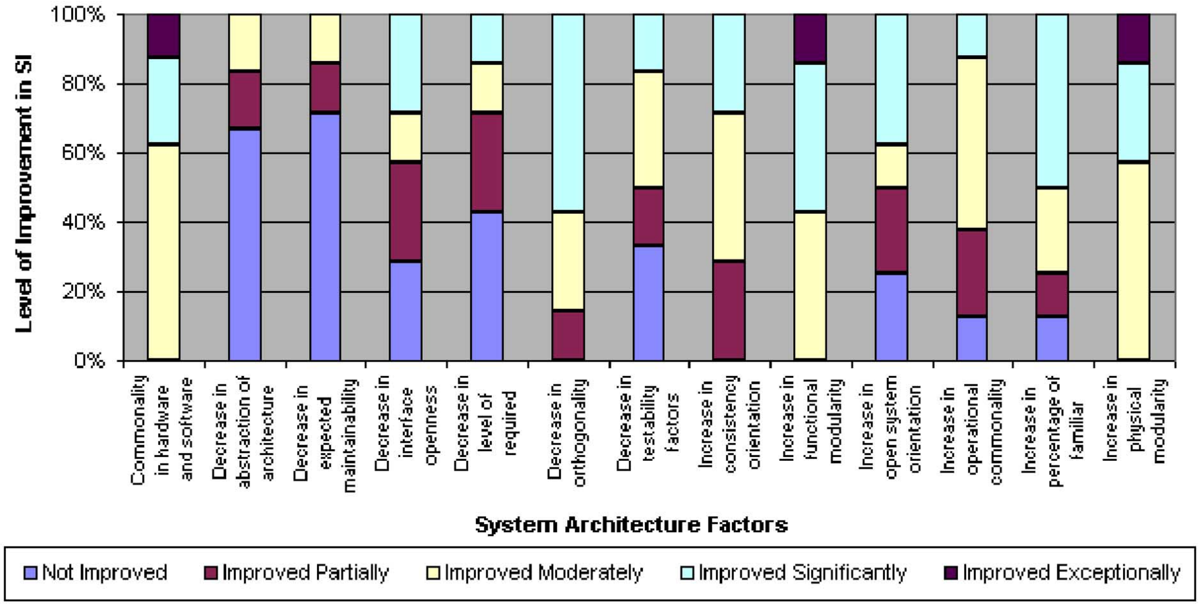


Fig. 10. SI process improvements as a result of system architecture activities.

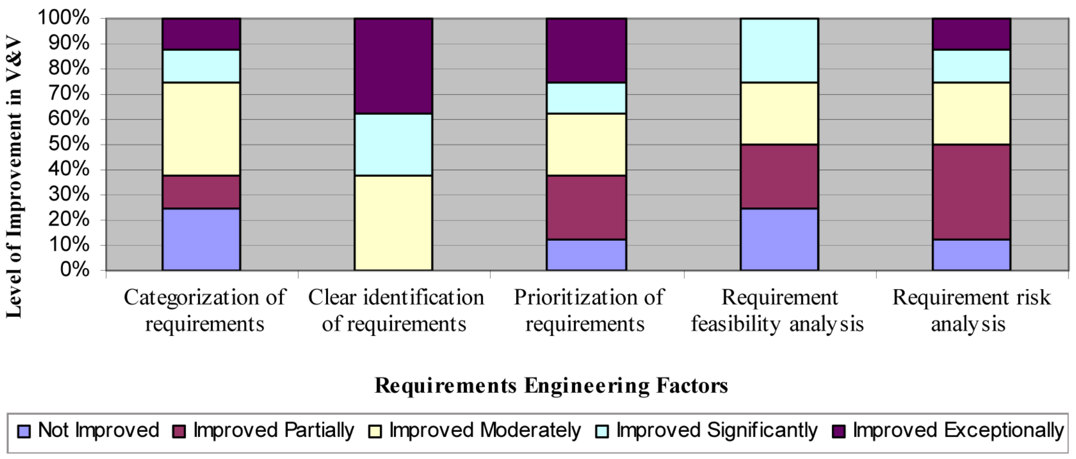


Fig. 11. V&V quality improvements as a result of requirements engineering activities.

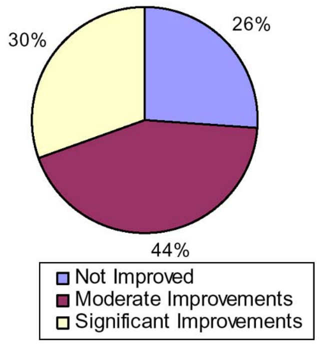


Fig. 12. V&V quality improvements as a result of requirements engineering.

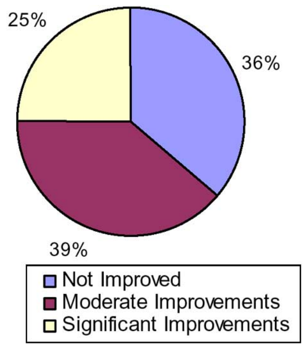


Fig. 13. V&V quality improvements as a result of system architecture activities.

and in improving the quality of verification and validation are discussed in the following.

Clear Identification of System Requirements: There has been significant 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

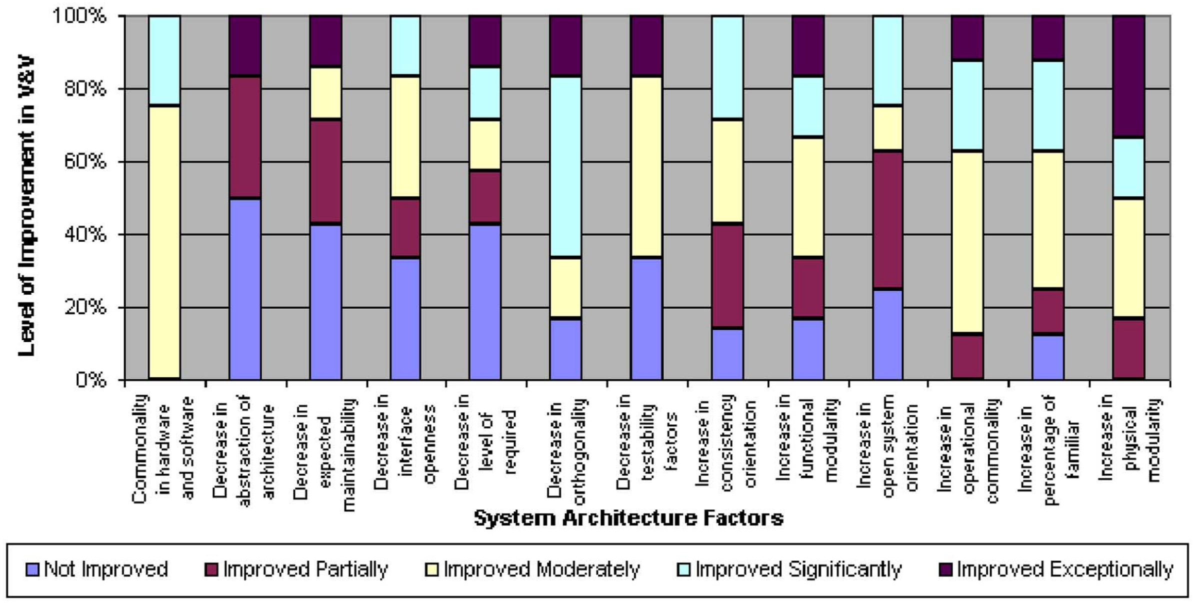


Fig. 14. V&V quality improvements as a result of system architecture activities.

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 life cycle.

By performing this activity, the system capabilities and functions can be clearly identified. This activity impacts every phase of systems 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 with feasibility and effort required can be analyzed up in the life cycle and can be mitigated. The result is lower integration operational complexity and organizational complexity.

Prioritization of System Requirements: 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 life cycle. 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 life cycle with stakeholder participation. Prioritizing and concentrating on core functionalities/capabilities first and then building upon them can also help in configuration management. This activity helps in reducing both the technical and configuration complexity of integration.

B. *Impacts of Systems Architecture on Systems Integration Complexity and Quality of Verification and Validation*

The results of the survey suggest that there will be a significant improvement in systems integration complexity and quality of verification and validation 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. This is due to the fact that there is significant increase in the use of COTS, strategic outsourcing, and value based capabilities development, and in the need for supply chain excellence. Another architecture trend is the migration towards Service Oriented Architectures (SOA). SOAs are focused on the supportability of operational scenarios and event driven architectures which are integration friendly [32], [33]. The factors of systems integration process complexity are addressed earlier in the life cycle as a part of the architecture by adopting these integration friendly architecture methodologies. Some of the system architecture factors that impact systems 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.

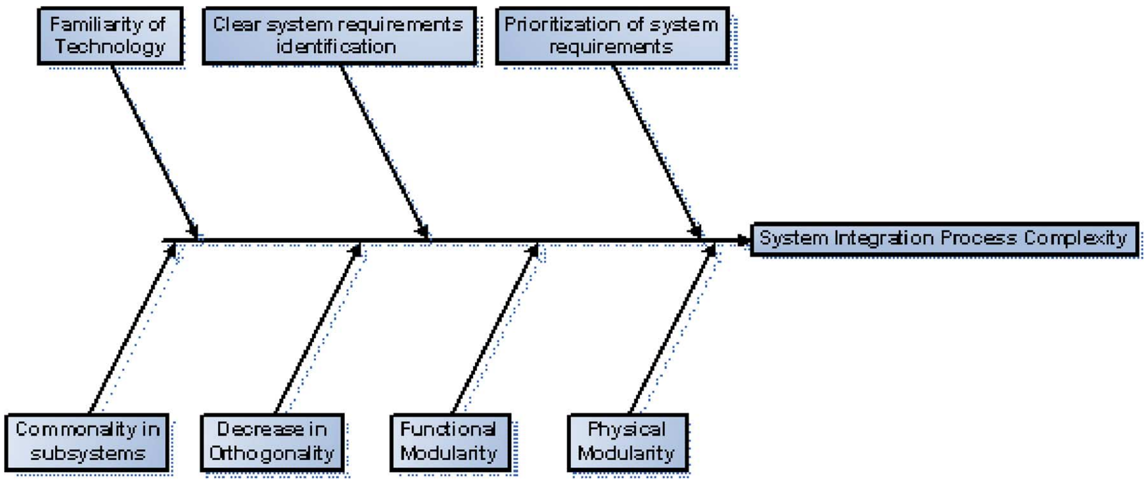

Familiarity of Technology
Clear system requirements identification
Prioritization of system requirements
System Integration Process Complexity
Commonality in subsystems
Decrease in Orthogonality
Functional Modularity
Physical Modularity

Fig. 15. Major factors impacting SI process complexity.

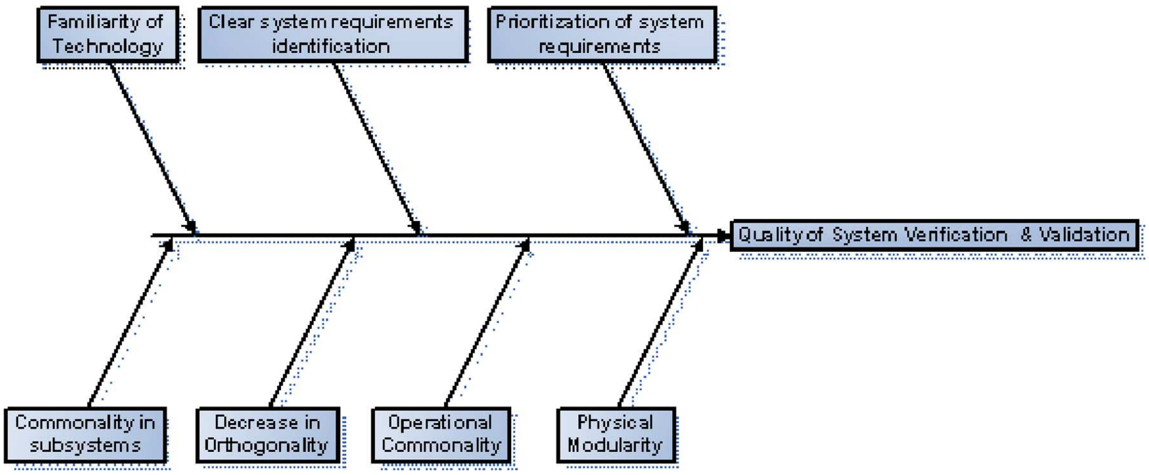

Familiarity of Technology
Clear system requirements identification
Prioritization of system requirements
Quality of System Verification & Validation
Commonality in subsystems
Decrease in Orthogonality
Operational Commonality
Physical Modularity

Fig. 16. Major factors impacting quality of V&V.

Based on the survey results, the factors of system architecture that impacted systems integration process complexity and quality of system verification and validation the most (resulting in at least partial improvement) are discussed in the following.

Commonality in Hardware and Software Subsystems: This system architecture factor emerged to be an important factor impacting both systems 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 in familiarity of subsystems. This reduces the technical complexity associated with systems integration process.

Increase in Physical Modularity: Quality of verification and validation can have at least moderate improvements and Systems 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 systems integration. Modularity also results in higher levels of maintainability and support. This reduces the technical complexity associated with systems integration process.

Increase in Functional Modularity: Systems 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 because it adds to the ease of building upon or upgrading the existing functionalities. This is a key factor in evolutionary and iterative system developments. The risks associated with system architecture during rapid system development would also be reduced by addressing functional modularity. This reduces the technical complexity associated with systems 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 verification and validation.

Decrease in Orthogonality: This factor of system architecture results in improvements in both quality of verification and validation and systems 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 systems integration process.

Increase in Percentage of Familiar Technology: By having familiar technologies in system architecture, quality of verification and validation and systems integration process can be improved moderately. This can be achieved by adopting good product line architectures and nested layer architectures. This factor reduces the programmatic and configuration complexities of systems integration process.

Functional modularity as a characteristic of system architecture plays an important role in reducing systems 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 systems integration complexity. These differences could be attributed to the nature of the tasks involved in systems integration and system verification and validation. Modularity is the extent to which the system is made up of well defined, functionally non-overlapping, and 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, systems integration is simplified.

The results also indicate that the most of the factors that impact systems 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 life cycle. 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.

VI. Conclusion

The specific nature of application-domain plays a critical role in the impact of the system architecture and system requirements related factors on systems 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. Finally, the relevance of these findings to system of systems and extended enterprises is unknown, and bears further research.

References

[1] D. Carney and P. Oberndoff, "Integration and interoperability models for system of systems," presented at the Syst. Softw. Technol. Conf., Pittsburgh, PA, 2004.

[2] A. Dogru, S. Delcambr, C. Bayrak, M. Christiansen, and M. Tanik, "The development of an integrated system design environment," in *Proc. 2nd Int. Conf. Syst. Integr. (ICSI)*, 1992, pp. 691–698.

[3] Stevens Institute of Technology, Hoboken, NJ, "Fundamentals of system engineering," SYS 625 Course Notes, SEEM, 2005.

[4] W. Rossak and S. Prasad, "Integration architectures—A framework for systems integration decisions," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1991, pp. 545–550.

[5] D. Smith, L. O'Brien, K. Kontogiannis, and M. Barbacci, "The architect: Enterprise integration," *SEI News*, vol. 5, no. 4, pp. 1–14, 2002.

[6] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE 610-12, IEEE Computer Society, 1990.

[7] M. W. Evans and J. Marciniak, *Software Quality Assurance and Management*. New York: Wiley, 1987.

[8] N. Howard, C. Rolland, and A. Qusaibaty, "Process complexity: Towards a theory of intent-oriented process design," presented at the Int. Conf. Inf. Syst. (INFOS), Egypt, 2004.

[9] R. Jain, A. Chandrasekaran, and O. Erol, "A systems integration process model ($\mathrm{SI}^{\mathrm{PM}}$)," Stevens Inst. Technol., Hoboken, NJ.

[10] R. Jain, A. Chandrasekaran, and O. Erol, "Proposing a systems integration readiness framework," Stevens Inst. Technol., Hoboken, NJ.

[11] International Council on System Engineering (INCOSE), "Systems engineering handbook, Ver. 3" INCOSE Technical Board, Seattle, WA, 2006.

[12] G. Elias and R. Jain, "Exploring attributes for systems architecture evaluation," presented at the Conf. Syst. Eng. Res. (CSER), Hoboken, NJ, 2007.

[13] R. McCabe and M. Pollen, "Evaluating architectures with system attributes," presented at the Softw. Productivity Consortium, Herndon, VA, 2004.

[14] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston, MA: Addison-Wesley, 2003.

[15] Software Engineering Institute (SEI), Carnegie Mellon Univ., Pittsburgh, PA, "Quality measures taxonomy," 2006 [Online]. Available: http://www.sei.cmu.edu/str/taxonomies/qm_tax_body.html

[16] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2001.

[17] *Software Engineering—Software Measurement Process*, ISO/IEC 15939, 2002.

[18] Open Systems Joint Task Force, "Open systems joint task force," 2005 [Online]. Available: http://www.acq.osd.mil/osjtf/

[19] R. Jorgensen and I. Philpott, "Architectural abstractions," in *Proc. INCOSE Symp.*, 2002, pp. 1–9.

[20] *Logistics Support Analysis*, MIL-STD-1388-1A, DoD, Pars. 20, Apr. 1983.

[21] D. Buede, *The Engineering Design of Systems*. New York: Wiley Series in Systems Engineering, 2000.

[22] D. Verma and L. H. Johannesen, "Supportability assessment and evaluation during system architecture development," in *Proc. INCOSE Symp.*, Minneapolis, MN, Jul. 2000, pp. 699–706.

[23] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, and F. Peruzzi, "The architecture based design method," Softw. Eng. Inst., Carnegie Mellon Univ. (SEI-CMU), Pittsburgh, PA, 2000.

[24] F. Bachmann, L. Bass, and M. Klein, "Deriving architectural tactics: A step toward methodical architectural design," SEI-CMU, Pittsburgh, PA, 2003.

[25] F. Bachmann, L. Bass, and M. Klein, "Illuminating the fundamental contributors to software architecture quality," SEI-CMU, Pittsburgh, PA, 2002.

[26] R. Kazman, R. Nord, and M. Klein, "A life-cycle view of architecture analysis and design methods," SEI-CMU, Pittsburgh, PA, 2003.

[27] *IEEE Guide For Developing System Requirements Specifications*, IEEE 1233, 1998.

[28] *IEEE Recommended Practice For Software Requirements Specifications*, IEEE 830, 1998.

[29] R. Cloutier, "Applicability of patterns to architecting complex systems," Ph.D. dissertation, Stevens Inst. Technol., Hoboken, NJ, 2006.

[30] R. Cloutier and D. Verma, "Applying the concept of patterns to systems architecture," *Syst. Eng.*, vol. 10, no. 2, pp. 138–154, 2007.

[31] R. Cloutier and D. Verma, "Applying pattern concepts to systems (enterprise) architecture," *J. Enterprise Arch.*, vol. 2, no. 2, pp. 34–50, May 2006.

[32] V. Krishnamoorthy, N. K. Unni, and V. Niranjan, "Event-driven service-oriented architecture for an agile and scalable network management system," presented at the IEEE Conf. Next Generation Web Services Practices, Bangalore, India, Aug. 2005.

[33] A. K. Harikumar, R. Lee, C. Chia-Chu, and Y. Hae-Sool, "An event driven architecture for application integration using Web services," in *Proc. IEEE Conf. Inf. Reuse Integr.*, Aug. 2005, pp. 542–547.

Rashmi Jain received the M.S. and Ph.D. degrees in technology management from Stevens Institute of Technology, Hoboken, NJ.

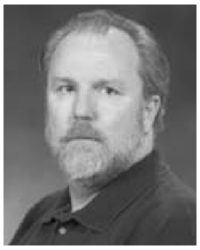
She is an Associate Professor with the Systems Engineering Department, Stevens Institute of Technology. She has over 15 years of experience of working on socio-economic and information technology (IT) systems. Over the course of her career, she has been involved in leading the implementation of large and complex systems engineering and integration projects. She is Head of Education and Research of International Council on Systems Engineering (INCOSE). In this role, she is leading the research on graduate and doctoral level Systems Engineering curriculum. Her teaching and research interests include systems integration, systems architecture and design, and rapid systems engineering.

Anithashree Chandrasekaran received the B.E. degree in electrical and electronics engineering from the P.S.G. College of Technology, Coimbatore, India, and the M.S. degree in systems engineering from the Stevens Institute of Technology, Hoboken, NJ, where she is currently pursuing the Ph.D. degree from the School of Systems and Enterprises.

Her research interests include rapid systems development and its processes, development process reengineering, risk management and modeling, systems integration, system design, and architecture. She is the president of Stevens INCOSE student chapter.

George M. Elias received the B.A. degree in information systems from Rutgers, the State University of New Jersey, Newark, NJ, and from the New Jersey Institute of Technology, Newark, NJ, and the M.S. degree in computer science from Stevens Institute of Technology, Hoboken, NJ, where he is currently pursuing the Ph.D. degree from the School of Systems and Enterprises.

He is a Systems Engineer with ITT Electronic Systems, Electronic Warfare Systems, Clifton, NJ. His research interests include systems architecture and systems engineering attributes.



Robert Cloutier (M'04) received the Ph.D. degree in systems engineering from the Stevens Institute of Technology, Hoboken, NJ, the M.B.A. from Eastern University, St. Davids, PA, and the B.S. degree from the United States Naval Academy, Annapolis, MD.

He is an Adjunct Professor with Eastern University and chairs the Rowan University Electrical and Computer Engineering Department Industry Advisory Board. He 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 belongs to the International Council on Systems Engineering (INCOSE), is a member of the Technical Leadership Team.