arXiv:1801.06837v1 [cs.SE] 21 Jan 2018

System-of-Systems Viewpoint for System Architecture Documentation

John Klein* and Hans van Vliet†
VU University, Amsterdam, Netherlands

Revised 11 Nov 2017

Abstract

Context: The systems comprising a system of systems (SoS) are independently acquired, operated, and managed. Frequently, the architecture documentation of these existing systems addresses only a stand-alone perspective, and must be augmented to address concerns that arise in the integrated SoS.
Objective: We evaluated an architecture documentation viewpoint to address the concerns of a SoS architect about a constituent system, to support SoS design and analysis involving that constituent system.
Method: We performed an expert review of documentation produced by applying the viewpoint to a system, using the active review method.
Results: The expert panel was able to used a view constructed using the baseline version of the viewpoint to answer questions related to all SoS architect concerns about a constituent system, except for questions concerning the interaction of the constituent system with the platform and network infrastructure.
Conclusions: We found that the expert panel was unable to answer certain questions because the baseline version of the viewpoint had a gap in coverage related to relationship of software units of execution (e.g., processes or services) to computers and networks. The viewpoint was revised to add a Deployment Model to address these concerns, and is included in an appendix.

Keywords: architecture documentation; system of systems; viewpoint definition; active review; expert panel; design cycle

1 Introduction

A system of systems (SoS) is created by composing constituent systems. Each constituent system retains operational independence (it operates to achieve

*john.klein@computer.org, Corresponding author
†hans@cs.vu.nl

a useful purpose independent of its participation in the SoS) and managerial independence (it is managed and evolved, at least in part, to achieve its own goals rather than the SoS goals) [1]. In order to assess suitability of the system for use in the SoS and to reason about SoS functionality and quality attributes, the architect of a SoS relies on documentation about the constituent system. In an ideal world, constituent system documentation would be available and address all SoS concerns. Our previous research (discussed in Related Work below) reports that this is not usually the case [2]. The challenge of documenting architectures whose parts are designed by separate organizations is a fundamental challenge of SoS and ultra-large scale systems [3].

Pragmatically, the SoS architect seeking information about a constituent system has limited options. If there is documentation and/or source code available, the SoS architect can attempt to learn enough about the constituent system design to address concerns about how that system will operate in the SoS. However, constituent systems are developed independently, and often there is limited or no access to documentation or code. The architect of the SoS could seek to collaborate with the architect of each constituent system to augment the constituent system architecture documentation with the information needed to address the SoS concerns. However, the managerial independence of the development and evolution of constituent systems within a SoS [1] often creates barriers to collaboration. Consider three examples of these barriers:

1. Each constituent system owner retains independent management of funding and objectives, and the constituent system architect's responsibilities for delivering system-oriented capabilities may not provide slack time to allow collaboration with the SoS architect.

2. There is no ongoing development on a particular constituent system, and so there is no architect assigned who could collaborate with the SoS architect.

3. Firms are integrating IT systems after a merger or acquisition, and the architects of particular acquired systems have been reassigned or dismissed, and so are not available to collaborate.

In each of these scenarios, collaboration between the SoS architect and the architects of each constituent system may become a tightly planned and managed, high ceremony event. The SoS architect must articulate a precise request for information, for which the constituent system architect estimates the cost to respond. The SoS owner and the constituent system owner negotiate to fund the constituent system architect's work to respond, and eventually the constituent system architect is directed to supply the requested information to the SoS architect. There is often little or no ability to iterate the information requests or seek elaboration of the responses, and given these high stakes, the architect of the SoS needs a pedigreed basis for a request for information.

The contribution of this paper is an architecture documentation viewpoint to assist SoS architects in collecting or creating sufficient documentation about constituent systems in a SoS. The viewpoint addresses stakeholder concerns about SoS design and analysis. This reusable “library viewpoint” conforms to the ISO/IEC 42010 standard for architecture description [4], and provides guidance for SoS architects to request sufficient information about constituent system architectures to satisfy SoS-level concerns about each constituent system operating in the SoS context. The viewpoint was evaluated by an expert panel in a single case mechanism experiment using the active design review method [5]. We found that the baseline version of the viewpoint covered most SoS stakeholder concerns; however, the experiment uncovered a gap in the area of deployment of software units to computers and networks. We describe how the viewpoint was reworked by adding a new model kind to address this gap.

This paper is organized as follows: §2 discusses related work in the areas of SoS and architecture documentation. §3 describes our approach to developing and evaluating the viewpoint, which was based on Wieringa’s design cycle [6]. §4 presents the results of our evaluation experiment, and our analysis and interpretation, including how the baseline version of the viewpoint was reworked based on the results of the experiment. §5 summarizes our conclusions, and the reworked viewpoint is included as an appendix.

2 Related Work

Generally, concern-driven architecture documentation approaches organize architecture documentation into *views* to address stakeholder concerns [4, 7, 8]. These approaches are widely used for software system architecture documentation, for example in the Rational Unified Process (RUP) 4+1 Views [9].

At the SoS level, view-based frameworks such as DoDAF [10] and MODAF [11] have emerged to document SoS architecture. The EU COMPASS Project [12], which ended in 2014, addressed SoS modeling, and SoS architecture documentation continues to be an area of active research [13], producing new documentation approaches such as S3 [14] and SySML-based approaches from the EU AMADEOS project [15].

The architect of a SoS must depend on the documentation of constituent systems. Our earlier research reported that one challenge to designing a SoS architecture is gaps in the architecture documentation of the constituent systems [2]. The architecture documentation of each constituent system usually focuses on the stand-alone operation of that system, and on the stand-alone development and evolution of that system. The architecture documentation for constituent systems is created during engineering development of the constituent system, for different purposes than SoS, notably constituent system bounds, constituent system goals, different modeling goals, and different characteristics of interest [16]. While functional interaction with external systems in support of the system’s standalone operation may be addressed by the architecture documentation, the quality attribute aspects of those interactions are typically not well covered.

Participation of a constituent system in a SoS introduces new concerns about that system; however, a survey by Bianchi and colleagues found that there are no applicable quality attribute frameworks for these concerns [17]. Our earlier research found interoperability to be a primary concern of SoS designers [18], and more recent work by Batista [19] and by Guessi and colleagues [13] confirmed that interoperability is a primary focus of SoS architecture documentation. The system mission characterization of Silva and colleagues provides insight into functional interoperation concerns [20]. Architecture documentation for constituent systems that addresses standalone operation may not address SoS interoperability concerns, which go beyond interface syntax. As the context for interface semantics is expanded to the SoS, behavior that might have been considered private to the system becomes externally visible. For example, design decisions such as whether to retry a failed request to an external system may not be architectural in the context of standalone operation, but become externally visible and architectural in the context of SoS operation.

The viewpoint that we developed could be considered an extension of the System Context Viewpoint defined by Woods and Rozanski [21], or of the system context diagram in the *Beyond Views* section of a Views and Beyond architecture document [7]. However, each of these focuses on how external interfaces and interactions support the independent operation of the system, and not on how the system interoperates with other systems to achieve an SoS capability.

3 Approach

Our objective is to design an artifact that contributes to the achievement of some goal. Here, the artifact is an architecture viewpoint, and the goal is to allow SoS architects to reason about a constituent system to design a SoS. Wieringa labels this a *design problem* and we used the *Design Cycle* approach [6] as follows:

1. Problem Investigation—We built on the related work discussed above to identify stakeholders in the SoS design process and their concerns related to constituent systems operating in the SoS context.

2. Treatment Design—We defined an architecture viewpoint to address those stakeholder concerns.

3. Treatment Evaluation—We evaluated the treatment by a single case mechanism experiment [6], using an expert panel to conduct an active design review [5].

3.1 Problem Investigation—Identify Stakeholders and Concerns

There are many stakeholders in a SoS and in its architecture [22]. Our focus is on the architecture design and analysis task, and specifically, reasoning about a constituent system in the context of the SoS, which narrowed the scope to the stakeholder roles listed in Table 1.

These stakeholders were selected because they are directly involved in understanding the constituent system architectures, proposing or defining changes

Table 1: Selected SoS Architecture Stakeholders

Stakeholder Name	Stakeholder Role
SoS Architect	Creates architecture designs to allow constituent systems to interoperate to achieve SoS goals. Proposes or defines necessary or desirable changes to constituent systems.
SoS Program Manager	Has ultimate responsibility for achieving SoS goals. Negotiates with program managers of constituent systems to make necessary or desirable changes to constituent systems.
Developer	Makes necessary or desirable changes to the software of the constituent systems.
SoS Testers and Integrators	Installs, configures, and tests the constituent systems interoperating as a SoS.

to those architectures for use in the SoS, and then constructing, testing, and integrating the constituent systems in the SoS.

Our earlier systematic review found that SoS research has heavily focused on interoperability concerns [18], however, our state of the practice survey indicated that practitioners designing and analyzing SoS architectures have broader technical and non-technical concerns [2]. Since this treatment will be employed by practitioners, we decided to augment the researcher-oriented findings with a survey of practitioner-focused literature to identify additional concerns about constituent systems when designing and analyzing SoS architectures.

The survey focused on an annual practitioner conference organized by the Systems Engineering Division of the National Defense Industry Association (NDIA)[1]. We reviewed all papers in the SoS Track and Architecture Track for the conferences from 2009 through 2016, and identified 14 papers that discussed SoS architecture concerns. We also reviewed the United States Department of Defense Systems Engineering Handbook for Systems of Systems [23], which provides guidance to a broad community of practice. From these sources, we identified a set of concerns that are shown in Table 2. As described below, these concerns were used to define the architecture viewpoint artifact.

3.2 Treatment Design—Define the Architecture Viewpoint

Wieringa defines a *treatment* as “the interaction between the artifact and the problem context” [6, §3.1.1]. We will define an artifact—an architecture viewpoint—that will be applied by an SoS architect to create an architecture view of a constituent system that provides the information needed to reason about that constituent system when it is operating in the context of the SoS. In this section

[1]See `http://www.ndia.org/divisions/systems-engineering`

Table 2: SoS Stakeholder Concerns from Practitioner-oriented Literature

Publication	**Concerns about constituent systems in an SoS**
Benipayo 2016 [24]	Dependencies on other constituent systems
Sitterle 2016 [25]	Interface adaptability (Interoperability), Recovery
Gump 2014 [26]	Interoperability, Dependencies on other constituent systems
Manas 2014 [27]	Dependencies on other constituent systems, Shared Resources
Carson 2014 [28]	Dependencies on other constituent systems
Guertin 2014 [29]	Portability, Scalability, Dependencies on other constituent systems, Security
Baldwin 2014 [30]	Stakeholders, Dependencies on other constituent systems
Gagliardi 2013 [31]	Shared resources
Pritchett 2013 [32]	Dependencies on other constituent systems
Dahmann 2012 [33]	Interoperability Perceived needs of constituent systems Processes, cultures, working practices between different participating organizations Dependencies at development time and run time
Dahmann 2012 [34]	Dependencies on other constituent systems
Smith 2011 [35]	Interoperability context: assumptions, constraints, drivers
Lane 2010 [36]	Monitoring and measurement
DoD 2008 [23]	Technical and organizational dependencies Interoperability Synchronization of delivery of features across constituent systems (dependencies) Constituent system stakeholders Constituent system needs and constraints Constituent system evolution strategy and built-in variabilities

we focus on the design of the architecture viewpoint, however, our evaluation will consider the entire treatment.

The viewpoint definition conforms to the ISO/IEC/IEEE 42010 standard [4], using Annex B of that standard as the template for the viewpoint specification. This approach was selected because of its status as a global standard and because it is compatible with producing documentation using other approaches such as Views and Beyond (see, for example, Appendix E in [7]).

The subset of the ISO 42010 conceptual model related to viewpoint definition is reproduced in Fig. 1.

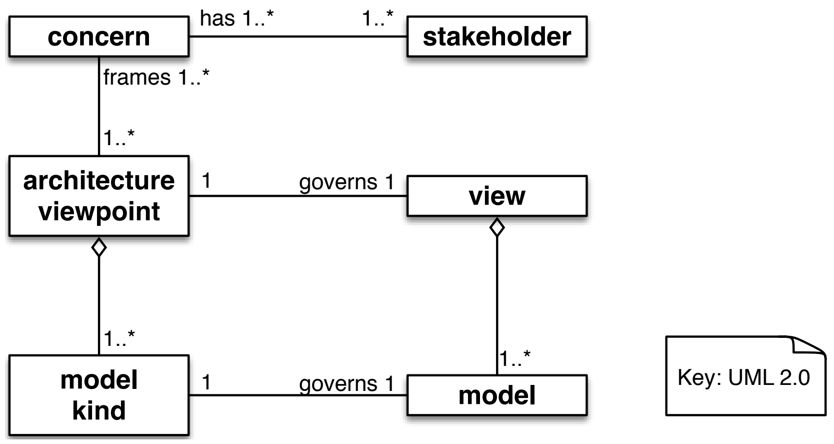

Figure 1: Extract from ISO 42010 Conceptual Model (Adapted from [4])

As shown in Fig. 1, the viewpoint definition begins by identifying stakeholders and concerns. The concerns identified in Table 2 are somewhat general. In order to define an architecture viewpoint to address the concerns, we refined these by mapping them to the set of quality attributes that Bass and colleagues [37] defined and found to be relevant to all software systems, namely performance, availability, security, testability, modifiability, and usability. In Table 3, we consider each of these quality attributes (along with a category for concerns about the system context that are shared by many stakeholders) as concerns at the SoS level, and then trace down to information needed at the constituent system level in order to address the SoS concern. This tracing was performed by considering the tactics [37] that might be applied to achieved the quality. Tactics that could be applied in the SoS context became concerns about constituent systems in Table 3. Bass and colleagues also discuss which stakeholders are typically concerned with each quality attribute, and we include this information in Table 3. Note that the SoS architect is concerned with all qualities.

Fig. 1 shows that the viewpoint comprises one or more model kinds. The model kinds were developed iteratively, using the following approach:

1. Identify the type of elements and relations needed to address each concern.
2. Group concerns that had the same types of elements and relations.
3. Define a model kind for each group of concerns.

Following this approach, we developed five model kinds, each addressing particular concerns from Table 3, and collectively addressing all concerns. The model kinds are listed below. Each model kind includes a brief discussion of the model elements and relationships, and the complete definition of the model kinds is provided in the appendix:

Table 3: Tracing SoS Concerns into Constituent Systems

SoS Concern	Constituent System Concern	Stakeholder
Performance	Shared resources: what is shared, how is use shared, behavior when insufficient resource is available (run time dependencies, monitoring and measurement, interoperability context)	SoS Architect Program Manager
Security	Authentication: identity validation repository (interoperability) Authorization: remote access to system and resources (interoperability) Encryption: algorithms and key management (interoperability)	SoS Architect Tester/Integrator Program Manager
Testability	Execution time dependencies: startup sequencing (run time dependencies) Fault detection and logging: internal (monitoring and measurement)	SoS Architect Tester/Integrator
Modifiability	Build time dependencies: COTS, FOSS, development environment, process/culture/working practices Run time dependencies on other constituent systems Variabilities: Affecting interfaces, decision model (dependencies) (evolution and built-in variabilities)	SoS Architect Developer
Availability	Fault detection and logging: interfaces (monitoring and measurement) Fault recovery (interoperability context)	SoS Architect Tester/Integrator Program Manager
Usability (for SoS operators)	Configuration dependencies among constituent systems (development time and run time dependencies)	SoS Architect
Context	Perceived needs and constraints of constituent systems Processes, cultures, working practices between different participating organizations Constituent system stakeholders	All

- Constituent System Stakeholders/Concerns–provides architecture context for the SoS architect and Program Manager by providing insight into the perceived need of the constituent system, and identifies stakeholders who may be impacted by the constituent system's operation within the SoS. The model elements are constituent system stakeholders and stakeholder concerns, with the relation *stakeholder has a concern.*

- Constituent System Execution Time Context—addresses concerns related to dependencies at execution time, shared resources, and to a lesser extent, provides overall context. The model elements are the constituent system and external software that the system interacts with during execution.

The relations are execution time interactions, e.g., sends/receives message, call/return, read/write data, etc.

- Constituent System Code Context—addresses concerns related to implementation dependencies. The model elements are the constituent system software, and external modules (e.g., libraries, development tools, packages, etc.). The relation is *uses.*

- Constituent System Interface Information Model—addresses concerns related to semantic interoperation of data elements. The elements are information elements of interest to the SoS (e.g., a SoS that deals with geo-location might use concepts like position, elevation, and direction), and information elements in the constituent system software. Relations are *logical associations* (1-to-1, 1-to-N, N-to-M), *specialization/generalization*, and *aggregation.*

- Shared Resource Model—addresses concerns about runtime resource sharing. Elements are components in the SoS representing resources used by the constituent system and by other systems, including processor compute cycles, memory, disk space, network bandwidth, files, databases, virtual infrastructure, or physical resources such as a display, antenna, or radio frequency. Relations are *acquires/releases* and *consumes.*

Table 4 shows the mapping from the concerns listed in Table 3 to the model kinds listed above.

The 42010 standard permits three approaches to accommodate multiple model kinds:

- Define multiple independent viewpoints, with each viewpoint comprising a single model kind. We rejected this approach because the models are not independent: In our case, omitting one model kind leaves a set of concerns uncovered.

- Define a framework that comprises multiple viewpoints, with each viewpoint comprising a single model kind. The standard characterizes a framework as "establishing a common practice. . . within a particular domain of application or stakeholder community" [4, §4.5]. We rejected this approach for two reasons: The resulting artifact was applicable beyond a single application domain, and a framework does not inherently imply that all viewpoints are used together, and so we have the model omission issue described above.

- Define a single viewpoint that comprises multiple model kinds. We selected this approach because it treats the set of model kinds as an atomic unit.

The single viewpoint was titled "SoS Constituent System Viewpoint". The complete viewpoint definition that conforms to the ISO 42010 standard is presented in the appendix to this paper (§6 below).

Table 4: Mapping Concerns to Model Kinds

Concern (from Table 3)	**Model kind(s) that address the concern**
Shared resources—what is shared, how is use shared	Shared Resource Execution Time Context Deployment
Behavior when insufficient resource is available (run time dependencies, monitoring and measurement, interoperability context)	Shared Resource Interface Information Execution Time Context
Authentication—identity validation repository (interoperability)	Interface Information Shared Resource Execution Time Context
Authorization—remote access to system and resources (interoperability)	Interface Information Shared Resource Execution Time Context
Encryption—algorithms and key management (interoperability)	Interface Information Shared Resource Execution Time Context
Execution time dependencies—startup sequencing (run time dependencies)	Execution Time Context Deployment
Fault detection and logging—internal (monitoring and measurement)	Interface Information Execution Time Context
Fault recovery (interoperability context)	Execution Time Context Deployment
Build time dependencies—COTS, FOSS assumptions	Code Context
Development environment dependencies (development time dependencies, process/culture/working practices)	Code Context Deployment Stakeholder/Concerns
Variabilities affecting interfaces	Interface Information
Decision model (dependencies, evolution and built-in variabilities)	Code Context Interface Information
Configuration dependencies among constituent systems (development time and run time dependencies)	Code Context Execution Time Context Interface Information
Perceived needs and constraints of constituent systems	Stakeholder/Concerns
Processes, cultures, working practices between different participating organizations	Stakeholder Concerns
Constituent system stakeholders	Stakeholder/Concerns

3.3 Treatment Evaluation—Active Design Review by Expert Panel

Treatment evaluation is "the investigation of a treatment as applied by stakeholders in the field...to investigate how implemented artifacts interact with their real-world context" [6, §3.1.5]. Our evaluation criteria were that the viewpoint provides sufficient coverage of concerns, and that a view created using the viewpoint provides sufficient detail to allow an SoS architect to reason about the constituent system operating in the context of the SoS.

The Introduction, above, described the high stakes involved in acquiring documentation about constituent systems, arising from the managerial independence of the systems. Therefore, an initial evaluation of this untested treatment through an observational case study [6, §17] or through technical action research [6, §19] would not be a responsible approach. We chose to perform a single case mechanism experiment [6, §18] using an expert panel, to complete the initial evaluation without impacting a real-world project. Expert panel assessment has been used by a number of researchers (e.g., Dyba [38], van den Bosch and colleagues [39], and Beecham and colleagues [40]), and so we saw this approach as both prudent and appropriate for an initial evaluation of an untested artifact. According to Hakim [41], small samples are effective to test explanations, particularly in the early stages of work. Expert panel recruitment is discussed below in §3.3.4, and panel demographics are shown below in §4.1.

Our treatment artifact is architecture documentation. Nord and colleagues provide a six-step structured approach to reviewing architecture documentation [42], which we followed for our evaluation. Steps 1-5 are discussed in subsequent subsections, and the results of the review (Step 6) are discussed below in §4. This six-step process is comparable to the eight-step process used by Beecham and colleagues [40], collapsing multiple process steps in several places.

3.3.1 Step 1: Establish the Purpose of the Review

We aim to produce an artifact to guide an SoS architect to request the architecture information about a constituent system that is sufficient for the architect and other stakeholders to reason about that system in the context of a SoS.

As discussed above, our criteria for the treatment evaluation were that the viewpoint provides sufficient coverage of concerns, and that a view created using the viewpoint provides sufficient detail to allow an SoS architect to reason about the constituent system operating in the context of the SoS.

3.3.2 Step 2: Establish the Subject of the Review

We are evaluating the treatment: the artifact applied in context, specifically the viewpoint applied to a system to produce an architecture view. For this, we applied the viewpoint to the Adventure Builder system, which was chosen because it has an openly available architecture description to use as the basis for constructing the view documentation.

The Adventure Builder system is a reference application, developed for a a fictitious company that sells adventure travel packages that bundle airline transportation, hotel accommodations, and guided activities. The system has a customer-facing website that allows customers to shop and purchase adventure packages, and a service-oriented architecture back-end that integrates with external payment processing and travel provider services.

The view documentation that was produced is available as part of the review instrument, shown below in §7.

3.3.3 Step 3: Build or Adapt Appropriate Question Sets

Nord and colleagues identified several review styles:

- Questionnaire—reviewers assess the artifact using a list of questions provided by the review organizer.
- Checklist—reviewers rate the artifact using a list of yes/no questions (a special case of the questionnaire style).
- Subjective review—stakeholders also play the role of reviewer and pose questions to themselves.
- Active review—architects ask questions that require reviewers to use the subject artifact in order to answer the questions.

We chose to use an active review style, as this approach ensures that the reviewers skim the entire artifact and read some parts of the artifact in detail. It also evaluates the treatment (artifact in use), and not just the contents of the artifact.

However, we also wanted to understand how our experts would use their knowledge and experience to approach a SoS design problem, and so we incorporated a subjective review, where each expert formulated questions about a SoS design problem, and then later in the review, answered these as active review questions.

The instrument we created for the review had multiple parts (see §7, below):

1. Demographic information about the reviewer.
2. A narrative vignette that created a usage scenario for the architecture view. It asked each reviewer to play the role of a SoS architect tasked to integrate the Adventure Builder system into an SoS, and specified a design problem with three new SoS capabilities.
3. Subjective review questions—we asked each reviewer to record three questions that they had about the architecture of the Adventure Builder system, related to the design problem.
4. We next provided the architecture view artifact.

5. For each of the three new capabilities, we created several questions about the new SoS design, and the reviewer had to use the architecture view to answer the questions.

   Our questions were refined from the "Key Design Decisions" question list from Nord and colleagues [42, §4.5]. Two examples of the questions are shown here; the entire instrument is contained in the Appendix in §7.

   - Does the Adventure Builder system have existing request-response interfaces with external systems? If so, what protocols/technologies are used for these interfaces?
   - The inputs to the new payment processing interface are: Card Type, Card Number, Card Expiration, and Card Security Code/CCV. Can the current Adventure Builder system provide all of these elements?

   Reviewers also answered the three subjective review questions that they created earlier in the review. We asked reviewers to annotate their responses indicating the document sections that they consulted to answer each question. We wanted to limit the duration of the review exercise to one hour, and so we presented eight active review questions, which along with the three subjective review questions, required each reviewer to answer a total of 11 questions.

6. The instrument concluded by recording the amount of time that the reviewer spent, and with open-ended questions about the realism of the scenario, the contents of the architecture view, and any comments about the review exercise.

3.3.4 Step 4: Plan the details of the review

Nord and colleagues define three major activities in this step: Constructing the review instrument in light of constraints and intentions for reporting results; identifying actual reviewers; and planning review logistics.

Our review instrument is outlined above in §3.3.3. It includes the question set, reflects how we addressed review time constraints, and collects demographic and other information to support our reporting of research results. We decided to present all reviewers with the same set of eight active review questions (i.e. all participants were assigned the same treatment).

Experts were recruited from a population of experienced practitioners in the areas of SoS and enterprise system integration, who have worked in the field for several years and have been responsible for designing the architecture for several SoS. Experts were targeted to represent different backgrounds and system domains, as recommended by Kitchenham and colleagues [43]. Additionally, we sought geographic diversity and organizational diversity. We recruited eight experts from eight different organizations to participate in the review exercise, and seven accepted.

Finally, our review logistics were simple: We emailed the instrument to each reviewer, who worked independently to complete the review and return the completed instrument to us by email.

3.3.5 Step 5: Perform the Review

As noted above, we performed the review by sending an identical survey instrument by email to each reviewer, and receiving a completed instrument sent back to us by email.

4 Analysis and Results

4.1 Expert Panel Demographics

The expert panel demographics are presented in Table 5. These data were self-reported by each expert. These data were collected to verify that the invitee met the inclusion standards in §3.3.4, and to assess the diversity of experience of the panel.

4.2 Active Review Question Responses

Here we discuss responses to the eight active review questions that we created and which were assigned to all reviewers.

All of the participants answered the eight active review questions correctly, with some minor variations in responses due to differing interpretations of the question wording and the context. In particular, the use of the unqualified term "interface" in several questions proved confusing: This was interpreted to mean programmatic interface or user interface, or both.

Six of the seven reviewers indicated which model(s) they used to answer each question (Some reviewers used more than one model to answer a question.). These responses showed that every model was used by at least one reviewer to answer at least one question, indicating that the questions covered the breadth of the artifact.

Based on the small sample size (N=7), we hesitate to perform statistical analysis on the relationship between questions and models; however, we show the frequency of each model's use in Table 6.

4.3 Subjective Questions

As discussed above in §3.3.3, in addition to the eight active review questions that we developed, we asked each reviewer to specify three questions that they thought were important for this design problem. In addition to helping triangulate to improve the quality of evaluation (discussed in the next section), it provided direct insight into an SoS architect's concerns when presented with a design problem.

Table 5: Expert Panel Demographic Information (N=7)

Variable	**Reported Value(s)**
Current position title	Software Architect (3) Chief Enterprise Architect (1) Chief Technical Officer (1) Solution Architect (1) Information Architect (1)
Current industry	Consulting services—multiple industry domains (4) Government (2) Financial services (1)
Prior industry experience (multiple responses allowed)	Software product development (4) Financial services (2) Academia (2) Consulting services—multiple industry domains (1) Government (1) Utility (1) Telecommunications (1) Transportation (1) Defense (1)
Nationality	USA (3) Netherlands (2) Brazil (1) UK (1)
Number of years of professional experience developing or integrating systems	18-40 years (Average = 30, Median = 30)
Approximate number of system integration projects worked on	Responses ranged from 8 to “more than 100”.

Our seven reviewers posed three questions each. There was significant overlap among these 21 questions, and we clustered the questions into six categories, shown in Table 7.

The questions in the Platform category could not be readily answered by the reviewers using the architecture documentation provided. The viewpoint included a Code Context model kind, which could represent the dependencies on platform code modules, including application containers, operating systems and database libraries, and virtual machines. However, the viewpoint does not include a deployment model that would directly address this category of concerns by showing the relationship between the execution elements of the constituent system—processes, services, applications, etc.—to computer nodes and networks

Table 6: Active Review Coverage of Models

Model Name	**Number of times used to answer active review question**
Stakeholder/Concerns	13
Execution-time Context Model	12
Code Context Model	8
Interface Information Model	22
Shared Resource Model	3

Table 7: Subjective Question Categorization

Category	**Example Questions**	**Frequency (N=21)**
Platform	What is the platform/technology stack/runtime environment used by the constituent system?	7
Data Model	What is the logical data model used in the constituent system? How is customer-identifying or user-identifying data handled?	6
Implementation Quality/Risk	What are the known problems in the constituent system? What is the development history (internal, acquired, outsourced, etc.)?	3
User Interface	What is the user interface exposed by the constituent system?	2
Functional Structure	What is the functional structure of the constituent system?	2
Architecturally-Significant Requirements	What are the quality attribute requirements for the constituent system?	1

(e.g., [7, §5.2]).

The questions in the Implementation Quality/Risk category address issues that are important to understand when designing a SoS; however, this information is not part of the architecture (i.e. structures comprising elements, relationships, and properties) of the constituent system, and these concerns can be addressed by reviewing or inspecting non-architectural artifacts such as an issue tracking system or source code repository.

The questions in the User Interface category could not be readily answered by the reviewers using the architecture documentation provided. Further research is needed into the underlying concern—a possible explanation is that the enterprise business system context for the vignette that we used for the exercise triggered this concern based on the expert's experiences, even though none of the desired new capabilities involved the user interface.

The question about Architecturally-Significant Requirements was not readily answered by that reviewer using the architecture documentation provided. A complete ISO/IEC/IEEE 42010-compliant architecture description would contain rationale that includes architecturally significant requirements [4, §5.8]. Our review instrument included only a subset of a complete architecture description containing the view that was the subject of the evaluation, and the view contained models with no rationale.

Questions in the Data Model and Functional Structure categories were readily answered by the reviewers using the architecture documentation provided.

4.4 Interpretation and Viewpoint Rework

Based on the experiment results discussed above, we found that the baseline version of the architecture viewpoint adequately covered the SoS stakeholder concerns that we had identified in our Problem Investigation. However, our expert review panel's subjective review questions uncovered three categories of concerns that we had not identified in our Problem Investigation, and were not addressed by the baseline version of architecture viewpoint.

Below, we discuss each of these categories of concerns, and how the baseline version of viewpoint definition was reworked to produce the fubak viewpoint definition presented in the Appendix (§6).

4.4.1 Runtime Deployment Environment Concerns

The first category of concerns that was not addressed by the baseline version of the architecture viewpoint involved the runtime deployment environment of the constituent system. The subjective review questions that raised this concern covered two areas: the software platform (operating system, application server, database manager, service bus, etc.), and the physical deployment (mapping of software to compute nodes and networks).

In developing the viewpoint definition, we expected that the software platform concerns would be addressed by the Execution Time Context Model and/or the Code Context Model; however, those models in the Adventure Builder System artifact provided to the reviewers did not contain sufficient detail to address the concern. We have reworked the viewpoint definition by extending the "Elements" section of these two models to add the software platform elements to the model, and by extending the "What's it for" section to add that the model is used to answer questions such as those posed by the expert panel.

The physical deployment concerns arise from the distributed nature of a SoS. This physical distribution affects performance, availability, and possibly security and other qualities. It is necessary for the SoS architect to understand the physical deployment of the constituent system (how software is mapped to compute and network resources) because, when the system becomes part of an SoS, those compute and network resources may be shared with other constituent systems, or may be configured differently from the constituent system's stand-alone architecture.

The Shared Resource Model definition identifies network bandwidth and compute resources as elements that may be shared, in order to address concerns about performance. In the Adventure Builder System artifact provided to the reviewers, that model was represented as a table, with deployment information provided as part of the description of each element. This presentation style did not provide sufficient detail to address the reviewer's concern. Many architecture documentation approaches define a Deployment Model, for example the Deployment Style defined by Clements and colleagues [7] or the Physical View defined by Kruchten [9]. In this model, elements are units of software execution (e.g., processes, services, etc.), and physical infrastructure (e.g., computer nodes and networks), and the relation of "executes on" maps software to physical infrastructure. This model is used to address concerns about performance, availability, and possibly security and other qualities. We have reworked the viewpoint definition to add a Deployment Model.

4.4.2 Implementation Quality and Risk Concerns

The second category of concerns that was not addressed by the baseline version of the architecture viewpoint involved the quality of the implementation of the constituent system, and assessing risk in integrating it into the SoS.

These concerns might be seen to intersect with several of the concerns drawn from the practitioner-oriented literature shown in Table 2, namely "Processes, cultures, working practices" and "Constituent System Evolution Strategy", but on the whole, they were not considered in developing the baseline version of the viewpoint.

As discussed above, it is important to understand these issues when designing a SoS. The expert panel's questions reflect common architecture approaches, such as the Risk and Cost Driven Architecture approach [44]. However, we think that this is not part of the architecture (i.e. structures comprising elements, relationships, and properties) of the constituent system, and that these concerns can be addressed by reviewing artifacts such as an issue tracking system or source code repository. We did not rework the baseline version of the viewpoint in response to this gap.

4.4.3 User Interface Concerns

The third category of concerns that was not addressed by the baseline version of the architecture viewpoint involved the user interface of the constituent system. As noted above, the constituent systems in a SoS are characterized by operational independence, which would imply one of two user interface integration patterns:

- *Mashup*, where the SoS user interface is developed using APIs of the constituent systems. In this case, the constituent system's user interface is not presented directly.

- *Portal*, where the user interface of a constituent system is presented in a sub-window (e.g., frame or pane) of the SoS user interface. In this case, the constituent system's user interface is presented in its entirety, without modification.

Therefore, concerns about the user interface, per se, do not appear to be generalizable SoS concerns, but there may be system-specific concerns. For example, a mashup approach would introduce concerns that would be addressed by the Execution Time Context Model and the Interface Information Model. A portal could introduce concerns about the user interface display as a shared resource, to be addressed by the Shared Resource Model.

We did not rework the baseline version of the viewpoint in response to this gap; however, this may be an area for further research.

4.5 Threats to Validity

Construct validity is the degree to which the case is relevant with respect to the evaluation objectives [45]. Here, our objective was to evaluate the ability of an architecture documentation viewpoint to address the concerns of a SoS architect about a constituent system within the SoS, in order to support SoS design and analysis involving that constituent system. The selection of the active review evaluation method ensured that the reviewers at least skimmed the entire document, and our recording of the sections of the document used to answer each question ensured that certain sections were read in detail. Also, the reviewer's positive comments about the realism of the review vignette support the construct validity of the experiment. Our objective *was not* to compare this treatment (use of the viewpoint to reason about the constituent system) to other treatments (e.g., using documentation and code from the constituent system to reason about the system).

Internal validity concerns hidden factors, which is a concern when examining causal relations [45]. Our use of the active review method introduced the potential threat to internal validity that the questions created for the review may have been unconsciously influenced by our knowledge of the Adventure Builder system architecture and its documentation. We mitigated this risk by also incorporating subjective review questions: prior to reading the architecture documentation, each reviewer created three questions, and then later used the documentation to answer those questions. This use of triangulation increases the reliability of our results [45].

External validity is related to the generalizability of the results in the context of a specific population. As discussed above in §3.3, we chose to use an expert panel with a small sample size. According to Hakim [41], small samples are effective to test explanations, particularly in the early stages of work. By definition, our single case mechanism experiment does not support statistical generalization, and so suffers the same external validity challenges of all case study research [46]. Our total response rate (recruitment to completion) was

87.5%, from an expert panel with a diversity of experience and system domain coverage, so we believe that our findings are valid at least for SoS and constituent systems that are similar in size and scope to the SoS described in the vignette that formed the basis of our review.

5 Conclusions and Future Work

In this paper, we have introduced an architecture viewpoint to address the concerns of a SoS stakeholders about a constituent system within the SoS, in order to support SoS design and analysis involving that constituent system. We evaluated this viewpoint using a single case mechanism experiment: An expert panel performed an active design review using a question set that we provided. The expert panel also created subjective questions, which provided additional insight into the concerns of a SoS architect when solving a design problem and improved the quality of our data by mitigating internal validity concerns inherent in the active review process.

The evaluation results were generally positive, with the viewpoint showing promise in providing guidance for SoS architects seeking architecture knowledge about a constituent system. However, the evaluation identified a gap in the baseline version of the viewpoint definition: It was missing a deployment model for the constituent system that shows the relationship of the software to computer nodes and networks. The viewpoint definition presented in the appendix has been reworked to reflect this change.

The viewpoint conforms to the ISO/IEC/IEEE 42010:2011 (E) standard for architecture description, and the revised viewpoint comprises five model kinds: Constituent System Stakeholders/Concerns, Constituent System Execution Time Context, Constituent System Code Context, Constituent System Interface Information Model, Shared Resource Model, and Deployment Model.

The managerial independence of constituent systems poses challenges for SoS architecture designers, and frequently the architecture knowledge acquisition process involves high stakes activities that risk damage to the architect's reputation and other consequences. Further empirical research in this area must be designed within the constraints of this context. Our results provide the confidence to evaluate this viewpoint using methods such as case study or technical action research.

6 Appendix: Viewpoint Definition

The viewpoint defined here is a revised version of the baseline viewpoint used to create the artifact that was the subject of the experiment discussed in the body of this paper. The following revisions were made to the baseline version of the viewpoint, as described in §4.4:

- The "Elements" sections of the Execution Time Context Metamodel (Table 10) and the Code Context Metamodel (Table 11) were revised to specify that platform elements such as operating system, application server, and database manager should be included.

- A new metamodel was added. The Deployment Metamodel (Table 14) relates software units of execution (e.g., processes or services) to the execution environment of computers and networks. Table 15 and Table 16 were revised to add a reference to the new Deployment Metamodel.

This viewpoint definition follows the template in Annex B of ISO 42010 [4].

6.1 Viewpoint Name

This defines the "SoS Constituent System Viewpoint", for use in documenting the relevant parts of the architecture of one constituent system in a SoS.

6.2 Viewpoint Overview

The need for this viewpoint is discussed in §1 of this paper.

6.3 Concerns Addressed by this Viewpoint

Table 2 and Table 3 in the body of this paper show the concerns addressed by this viewpoint, and map the concerns to the stakeholder roles identified in the next section, below. §3.3.1 also discusses the method used to identify the concerns.

6.4 Typical Stakeholders

The stakeholder roles addressed by this viewpoint are shown in Table 1 in the body of this paper.

These stakeholders were selected because they are directly involved in understanding the constituent system architectures, proposing or defining changes to those architectures for use in the SoS, and then constructing, testing, and integrating the changed constituent systems in the SoS.

6.5 Model Kinds/Metamodels

This viewpoint specifies of a number of model kinds[2].

We apply the principle of separation of concerns, and so each model kind is defined using a single architecture style [7]: module styles address development time concerns, component and connector styles address execution time concerns, and allocation styles map between software elements and their environment.

Each model kind is specified as a metamodel. The metamodel template is shown in Table 8, and is based on the Style Guide Template defined by Clements and colleagues [7].

[2]In the terminology of ISO 42010, a *viewpoint* applied to a system yields a *view*. Analogously, the standard defines a *model kind*, which, when applied to a system, yields a *model*.

Table 8: Template used to specify metamodels for model kinds in this viewpoint

Name:	**Name of the model kind**
Type:	Module, component and connector, or allocation, as defined by Clements and colleagues [7].
Elements:	The types of elements allowed in this model kind, and the properties that should be attached to each element instance.
Relations:	The types of relations among elements allowed in this model kind, and the properties that should be attached to each relation instance.
Constraints:	Any model construction constraints, such as cardinality of element or relation types or topology constraints.
What's it for:	Brief description of how the model kind is used to support SoS architecture tasks such as design, analysis, evolution, or evaluation.
Notations:	Recommended notations for documenting the model kind, such as table, diagram, or list.

The first model kind is defined in Table 9, and represents the stakeholders and their concerns for the constituent system. This provides architecture context for the SoS architect and Program Manager by providing insight into the perceived need of the constituent system, and identifies stakeholders who may be impacted by the constituent system's operation within the SoS.

The second metamodel in this viewpoint, shown in Table 10, addresses concerns related to dependencies at execution time, shared resources, and to a lesser extent, overall context.

Concerns related to development time are addressed in the metamodel defined in Table 11.

The metamodel defined in Table 12 addresses general information interoperation concerns.

Concerns about resource sharing are addressed in the metamodel defined in Table 13.

Concerns about deployment of software onto computers and networks are addressed in the metamodel defined in Table 14.

6.6 Correspondence rules

There are no specific correspondence rules for the models constructed using this viewpoint.

Table 9: Constituent System Stakeholders/Concerns Metamodel

Name:	**Constituent System Stakeholders/Concerns**
Type:	Allocation
Elements:	Constituent system stakeholders Stakeholder concerns about system architecture
Relations:	A stakeholder has a concern
Constraints:	Stakeholders can have multiple concerns. Multiple stakeholders can have the same concern.
What's it for:	Aids in understanding the scope of the constituent system, and who will be impacted by changes made to the constituent system to allow it to join the SoS.
Adding Assumptions:	List any stakeholders that were considered but intentionally excluded. Note concerns that were identified but not addressed by the architecture.
Notations:	List—one item per stakeholder, with list of concerns. Matrix—one row per stakeholder, one column per unique concern, "x" at row-column intersection means that the stakeholder in that row has the concern in that column

6.7 Operations on views

6.7.1 Creating a view of a constituent system using this viewpoint

In some cases, the information needed to create a view using this viewpoint already exists in the architecture documentation for the constituent system. Table 15 and Table 16 map the information required for this viewpoint to sources in two commonly used documentation frameworks: Views and Beyond [7], and DoDAF [10].

6.7.2 Interpretive, Analysis, and Design Methods

These operations on a view created from this viewpoint are discussed in the "What's it for" section of the metamodels specified above.

6.8 Examples and Notes

The evaluation instrument in §7 provides an example of applying this viewpoint to create a view on a constituent system.

Table 10: Constituent System Execution Time Context Metamodel

Name:	**Constituent System Execution Time Context**
Type:	Component and Connector
Elements:	Running system External software that the system interacts with
Relations:	Any interaction at execution time (e.g., sends/receives message, call/return, reads/writes data, interrupts, synchronizes with) Property: Interfaces used for the interaction on self and external software Property: Direction of interaction (initiated by constituent system or external system)
Constraints:	An interface on the constituent system may be used to interact with multiple external systems Multiple external systems may interact with the constituent system through the same interface on the constituent system
What's it for:	Aids in understanding the scope of the constituent system to analyze the impacts of necessary or desired changes Identifying viable SoS subsets and activity sequencing during SoS integration
Adding Assumptions:	Startup behavior should be documented, using a notation such as a message sequence diagram Monitoring and performance measurement behavior should be documented, using notations such as message sequence diagrams and state transition diagrams.
Notations:	Diagram—e.g., Context Diagram from Clements [7] List—one item per constituent system interface, with list of external systems and interfaces that it interacts with Matrix—rows are interfaces on the constituent system, columns are interfaces on external systems, "S" at a row-column intersection means that the constituent system interface sends an interaction to the external system, "R" means that the constituent system interface receives an interaction from the external system

Table 11: Constituent System Code Context Metamodel

Name:	**Constituent System Code Context**
Type:	Module
Elements:	Constituent system software External modules (libraries, packages, development tools, etc.) that the constituent software depends on
Relations:	Uses Properties: type of dependency (e.g., code generation, build, unit test, integration test), version identification or key features used for external modules, source of external modules (e.g., FOSS, COTS, GOTS)
Constraints:	Many-to-many
What's it for:	Aids in understanding the scope of the constituent system to analyze the impacts of necessary or desired changes. Identifying mismatches among external dependencies that will constrain deployment decisions or interactions among constituent systems in the SoS.
Adding Assumptions:	What evolution is assumed for the external modules? Are there new features or capabilities that are expected to be available that the constituent system will use?
Notations:	Diagram—e.g., Uses Context Diagram from Clements [7] List Matrix—This structure may be documented in a Dependency Structure Matrix generated for static analysis of the constituent system code.

Table 12: Constituent System Interface Information Metamodel

Name:	**Constituent System Interface Information Model**
Type:	Module
Elements:	Information elements of interest to the SoS (e.g., a SoS that deals with geo-location might have concepts like position, elevation, and direction) Information elements in the constituent system software architecture Properties: Should include units, timeliness, precision, security level, etc., as applicable
Relations:	Between SoS and constituent system information elements, and from constituent system elements to sub-elements (to refine details). Logical associations (1-1, 1-n, n-m) Specialization/generalization (is-a) Aggregation
Constraints:	None
What's it for:	Understanding how common concepts in the SoS are represented in a constituent system, and identifying mismatch between representations among constituent systems in the SoS
Adding Assumptions:	Explicitly identify SoS information elements that have no relationship to the constituent system
Notations:	Logical data modeling notations (ERD, UML)

Table 13: Shared Resource Metamodel

Name:	**Shared Resource**
Type:	Component and Connector
Elements:	Component(s) representing a resource that is used by the constituent system and by other external systems. These include processor computing cycles, memory, disk space, network interfaces, network bandwidth, files, databases or repository, virtual infrastructure, and system physical resources such as a display, radio frequency, or antenna. Component(s) in the constituent system that use the shared resource
Relations:	Any interaction during execution that acquires, consumes, or releases the shared resource.
Constraints:	None
What's it for:	Analyzing capacity and performance/availability of the SoS. Identifying cases of undesirable SoS behavior due to mismatch between resource sharing approaches of constituent systems.
Adding Assumptions:	Is the resource explicitly or implicitly acquired and released? What is the behavior if insufficient (or no) resources are available?
Notations:	Static diagrams—e.g., from Views and Beyond component and connector style guide [7] Behavior diagrams—message sequence charts, state transition diagrams, etc.

Table 14: Deployment Metamodel

Name:	**Deployment**
Type:	Allocation
Elements:	Software units of execution, with properties that specify the execution needs and constraints Computers and networks that execute software, with properties that specify the execution resources (e.g., compute cycles, memory, storage, and bandwidth) provided
Relations:	Software units execute on computers and networks.
Constraints:	None
What's it for:	Analyzing performance and availability, and possibly security and other qualities. Assessing operating cost (e.g., required hardware and software, operations staff skills).
Adding Assumptions:	Can any part of the execution environment be virtualized? What are the assumptions about the network's ability to reach the internet or particular resources on a private network?
Notations:	Table—Rows are instances or types of software elements, columns are instances or types of computers and networks Diagram—e.g., Deployment Diagram from Clements [7]

Table 15: Mapping from SoS Viewpoint Metamodels to Views and Beyond Approach

Viewpoint Metamodel Name	**Source of information in a Views and Beyond architecture document**
Constituent System Stakeholders/Concerns	Information Beyond Views—Documentation Roadmap. Stakeholder/View Matrix (typically generated by the architect but not explicitly included in the architecture documentation).
Constituent System Execution Time Context	Context diagram from one of the component and connector views, e.g., client-server, SOA, pipe and filter, or publish-subscribe.
Constituent System Code Context	Context diagram from a module uses view.
Constituent System Interface Information Model	Interface documentation for externally-visible interfaces (from component and connector views), or a data model view packet focused on externally-visible information elements.
Shared Resource	Component and connector view.
Deployment	Deployment view primary presentation or context diagram.

Table 16: Mapping from SoS Viewpoint Metamodels to DoDAF

Viewpoint Model Name	**Source of information in a DoDAF architecture document**	**Comments**
Constituent System Stakehold-ers/Concerns	AV-1 Overview and Summary Information PV-1 Project Portfolio Relationships	DoDAF does not generally treat stakeholders as a first-order concern. These DoDAF views provide insight into the operational, maintenance, and development stakeholders.
Constituent System Execution Time Context	SvcV-1 Services Context Description SvcV-3b Services-Services Matrix	
Constituent System Code Context	SvcV-1: Services Context Description	If the information is included, it is most likely to appear in the SvcV-1.
Constituent System Interface Information Model	SvcV-2: Services Resource Flow Description SvcV-6: Services Resource Flow Matrix StdV-1 Standards Profile	DoDAF use the concept of "resource flows" to identify interfaces and protocols.
Shared Resource	SvcV-3b Services-Services Matrix SvcV-10c Services Event-Trace Description	Shared resources may not be explicitly identified, but can be discovered using the SvcV views.
Deployment	SvcV-1 Services Context Description SvcV-3a Systems-Services Matrix	DoDAF "services" usually include both software and hardware elements, without explicit refinement. The DoDAF views noted here may provide insight, but are unlikely to provide all the information needed to create this model.

7 Appendix: Evaluation Instrument (including example use of the viewpoint)

This appendix contains the instrument used to evaluate the viewpoint, as discussed above in §3.3.3.

In order to construct the evaluation instrument, we had to use the viewpoint to construct system architecture documentation for the Adventure Builder System. The models were created in accordance with the viewpoint definition specified in §6, and so this also provides an example of the use of the viewpoint.

The evaluation instrument begins on the next page.

This is a role-playing exercise. You will play the role of an architect at a fictitious company called "Social Travel". Your business sells travel packages, and provides a number of social networking capabilities to allow your users to connect and share information with each other. You are responsible for a collection of integrated enterprise systems:

- *TravelPhotos*:provides photo storage, tagging, and sharing with other Social Travel users.
- *TravelStuff*: an online retailer for travel-related gear
- *TravelIns*: marketing travel insurance, such as trip cancellation coverage, international medical coverage, and emergency rescue/evacuation coverage.
- Enterprise-wide Social Features: a back-end system for cross-cutting features such as Profiles, Friends, Tags, Recommendations, Sharing, etc.

Your company has just acquired a smaller company called "Adventure Builder" that specialized in selling adventure travel packages. You need to integrate some of their systems with your enterprise systems, to realize three new capabilities:

New Capability #1—Social features
This capability adds social features to Adventure Builder catalog browsing. A user can see which trips her friends have taken, see comments about trips from other users, see trip photos from other users, share plans and itineraries with friends, etc.
The architecture approach to achieve this capability will store the social data repository (i.e. user profile, tags, sharing links, comments, etc.) outside of the Adventure Builder system. The Adventure Builder system must provide keys (or IDs) for data inside Adventure Builder that the social repository can link to. We also need to insert new widgets and elements into the customer UI to allow access to the social features.
New Capability #2—Common payment processing interface
This capability changes the Adventure Builder payment processing interface to use the same interface as is used by the existing TravelStuff systems.
The architecture approach to achieve this capability is to completely replace Adventure Builder's "Bank" interface.
New Capability #3—Add cross-sell features This capability changes the Adventure Builder user interface to cross-sell products from TravelStuff and TravelIns when user books trip. For example, based on the destination, offer relevant clothing products from TravelStuff and insurance from TravelIns against a hurricane forcing the trip to be cancelled.
The architecture approach to achieve this capability is to make a request to existing SocialTravel cross-sell business logic trip with itinerary information, and then insert new widgets and elements into the customer UI to display the cross-sell offers.

In order to scope the work to develop these new capabilities, you need information about the Adventure Builder system. Below, please list three questions that you have about the Adventure Builder system architecture. These questions do not have to be the highest priority, or listed in any particular order. We are trying to understand how you approach this problem.

Your Questions:
Question 1:
Question 2:
Question 3:

As often happens in a corporate acquisition, you do not have direct access to your counterpart architect at Adventure Builder. In this case, as soon as the deal closed, the Adventure Builder architect cashed in her stock and left on a trip around the world.

However, before she left, she prepared a documentation package for you that addresses concerns related to integrating an existing system into an enterprise or into a system of systems (SoS). This documentation refers to the as-is existing system as a "constituent system" of the SoS.

This documentation has five models, summarized here:

Constituent System Stakeholders/Concerns Model: Maps stakeholders in the constituent system to their concerns about that system. This helps you understand the scope of the constituent system, architecture drivers, and who will be impacted by changes that you make to the constituent system to allow it to join the SoS.

Constituent System Execution Time Context Model: Describes any runtime interactions between the constituent system and external systems, including request/response, data exchange, message passing, and error/exception handling.

Constituent System Code Context Model: Identifies external modules (libraries, packages, development tools, etc.) that the constituent software depends on, along with the type of dependency.

Constituent System Interface Information Model: Information elements within the constituent system that are of interest to the SoS (e.g., a SoS that deals with geo-location might have concepts like position, elevation, and direction). Note that this data may not be accessible through existing external interfaces of the constituent system.

Shared Resource Model: Component(s) that represent a resource used by the constituent system and by other external systems. These resources include processor computing cycles, memory, disk space, network interfaces, network bandwidth, files, databases or repository, virtual infrastructure, and system physical resources such as a display, radio link, or sensor. This model also includes component(s) in the constituent system that use each shared resource.

The models in the documentation package are shown on the following pages. There is a lot of detail here—skim it for now, and some later questions will ask you to look at it more closely. Again, you can print this document if you want to.

Some of the models have traceability references back to the Adventure Build architecture description, which is available at `https://wiki.sei.cmu.edu/sad/index.php/The_Adventure_Builder_SAD`, but you should not need to refer to that directly.

Model: Constituent System Stakeholders/Concerns

Table 17: Constituent System Stakeholders/Concerns Model for the Adventure Builder System

Stakeholder	**Concerns**	**Source**
Product Manager	Modifiability—add new business partners quickly	QAS1
Product Manager	Usability, Performance—purchase action latency	QAS2
Product Manager Operations	Latency under load	QAS3
Operation Finance	Reliability—Purchase requests to OPC are idempotent. Usability—Successful purchases are always acknowledged to customer.	QAS4
Legal Info. Sec. Product Manager	Security—Payment processing transactions are secure and meet all internal policies and regulatory compliance requirements.	QAS5
Operations Info. Sec. Product Manager	Denial of Service attack is detected and handled.	QAS6
Operation	24/7 availability. Failures detected and notification issued within 30 seconds.	QAS7

Model: Constituent System Execution Time Context

Table 18: Constituent System Execution-Time Context Model for the Adventure Builder System

External System	**Interfaces to the external system and interface properties**	**Source**
Bank	Interface: CreditCard Service/SOAP/Adventure Builder invokes request	Top Level SOA View Primary Presentation
Airline Provider	Interface: AirlinePO Service/SOAP/Adventure Builder invokes request Interface: Web Service Broker/SOAP/Adventure Builder receives request	Top Level SOA View Primary Presentation
Lodging Provider	Interface: LodgingPO Service/SOAP/Adventure Builder invokes request Interface: Web Service Broker/SOAP/Adventure Builder receives request	Top Level SOA View Primary Presentation
Activity Provider	Interface: ActivityPO Service/SOAP/Adventure Builder invokes requestInterface: Web Service Broker /SOAP/ Adventure Builder receives request	Top Level SOA View Primary Presentation
User's Email Client	Interface: SMTP/SMTP/external configuration file	Top Level SOA View Primary Presentation

Also, the top-level workflow diagram is applicable here to show how these interfaces are used in practice. This is shown in Fig. 2.

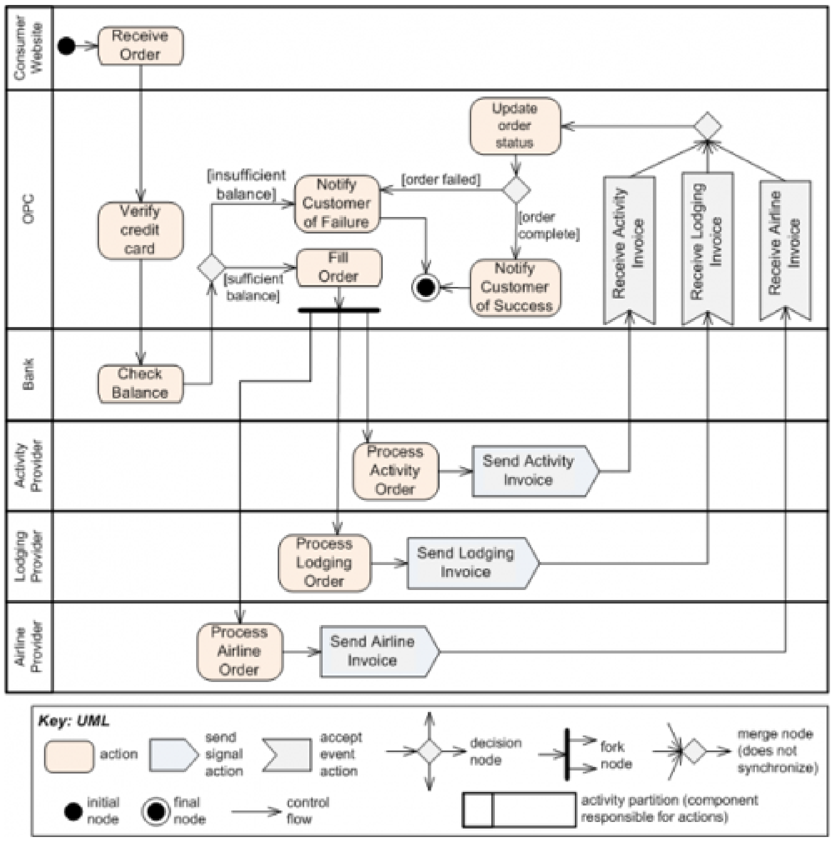

Figure 2: Workflow behavior diagram from Adventure Builder Architecture

Model: Constituent System Code Context

Table 19: Constituent System Code Context Model for the Adventure Builder System

External Module used by AB System	**Properties**	**Source**
gwt (Google Web Toolkit)	Dependency Type: Build (generates Javascript for execution); Version unspecified; Open Source	Top Level Module Uses Diagram
waf (Web Application Framework)	Dependency Type: Build?; Version “Java Blueprints”; Open Source	Top Level Module Uses Diagram
wsdls	Dependency: Build; Version unspecified; license unspecified	Top Level Module Uses Diagram

Model: Constituent System Interface Information Model

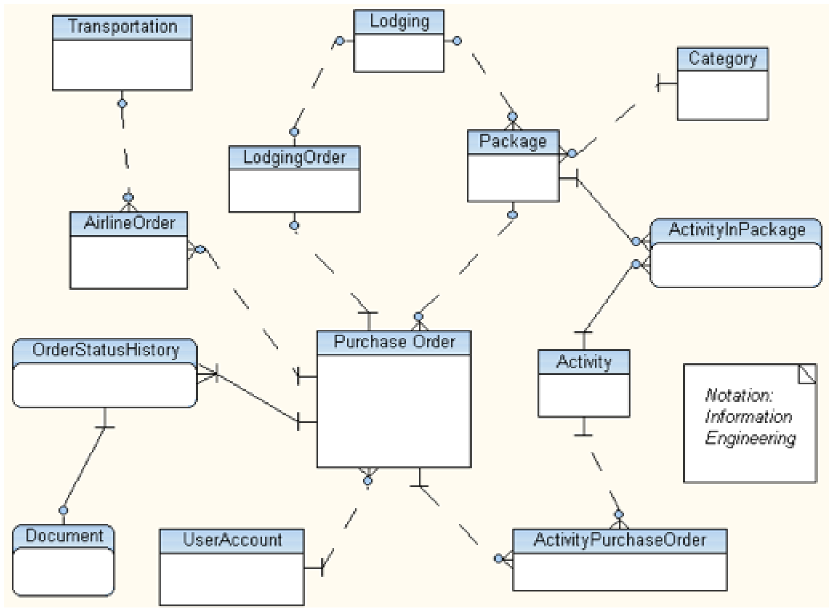

Figure 3: Part 1 of Constituent System Interface Information Model for the Adventure Builder System (From Adventure Builder Data Model View)

Our approach for new capability #2 requires us to replace the Bank interface. Part of the information model for the Bank interface is shown in Fig. 4.

Table 20: Element Catalog for Part 1 of Interface Information Model

Information Element	**Description**
PurchaseOrder	Aggregate of transportation, lodging, package, and activity orders.
UserAccount	An end user of the AdventureBuilder application. We store email id, password, and contact info.
AirlineOrder	Aggregate of purchased transportation entries.
Transportation	Each transportation entry is a flight available for booking in our travel packages. For each one, we record: name, departure and arrival airports, days and times, airline name, flight number, rate, cabin class.
LodgingOrder	Aggregate of purchased lodging entries.
Lodging	A hotel, guesthouse or B&B that can be used for lodging in our travel packages. For each type of lodging, we store name, description, location info, room description, rates.
Package	A travel package available in our catalog. A package specifies lodging and a list of activities. Attributes of a package include name, description, rate per person, category and a representative image to show to the user.
Category	A category of adventure travel packages. Examples: island packages, mountain adventures. This categorization helps the user to browse through our catalog of packages. Category data consists of a name, description and a representative image to show on the user screen.
Activity	An adventure activity available. Examples: snorkeling, fishing, bird watching, rafting, surfing. Activities are available in selected packages. Information stored for each activity include: name, description, rate, and a representative image to show to the user.
ActivityInPackage	This entity represents the many-to-many relationship between activity and package. It simply lists the activities in each package (“join table”)
ActivityPurchase-Order	Aggregate of purchased activity entries (“join table”)

```
   Type CreditCard
// This type is used to store the credit card information of the user.
String cardExpiryDate
String cardNumber
String cardType
```

Figure 4: Part 2 of Constituent System Interface Information Model for the Adventure Builder System

Model: Shared Resource Model

In the stand-alone Adventure Builder system, there are no resources shared with any external systems.

In the new SoS, we have several resources that now will be shared. These are shown in Table 21. For each resource identified in Table 21, the source of the information in the Adventure Builder Architecture Documentation is shown.

Table 21: Shared Resource Model for the Adventure Builder System

External Resource	**Adventure Builder Resource Usage**	**Resource shared with**	**Source**
Bank Interface (accessed through Firewall)	Validate credit card for every customer purchase (Call/Return)	Other SocialTravel.com applications (Call/Return)	OPC C&C View and Deployment View
Adventure Order Processing DB (executes on srv-dbopc)	The Order Processing Component uses this for consumer account data, consumer purchases and external invoices (R/W)	Consumer Website (mostly read for authentication, write only at account creation and update) Social features (read—need to characterize workload) Cross-sell (read—once per purchase)	OPC C&C View and Deployment View
Consumer UI (executes on srv-web1 and srv-web2)	Primary shop and purchase workflows.	Social Features—insert new content for tagging, sharing, etc. Cross-sell— insert cross-sell features.	Top Level Uses View and Install View and Deployment View

Now you are going to use the models provided in the view of the AB system to answer some questions. We have a few questions related to each of the new capabilities. Obviously, in a real integration project, there would be a multitude of questions: Here, we attempt to cover a small sample of these questions to assess the utility of the models.

In each answer, please note which models you consulted to make your decision. If you cannot find sufficient information in the models to make your decision, please state this (these questions were formulated without consulting the models).

First, let's consider our approach to achieving the first new capability: adding social features to the AB system. These social features will create runtime dependencies between the AB system and the Social Travel system.

Question	**Your Answer**
1. The AB system has 7x24 availability. The Social Travel systems have had recent outages, and there is ongoing work to improve availability. Which AB stakeholders do you need to engage with to understand the AB availability requirements?	
2. You need to expose the social features in the AB web user interface, which uses gwt (Google Web Toolkit). The Social Travel system uses V2.7.0 of the gwt. Which version does AB use?	
3. Is there an external programmatic interface to access the User Account and Purchase Order data elements?	

Next, let's consider the second new capability, which requires us to replace the payment processing interface on the AB system. AB calls this interface the "Bank" interface.

Question	**Your Answer**
4. The inputs to the new payment processing interface are: Card Type, Card Number, Card Expiration, and Card Security Code/CCV. Can the current AB system provide all of these elements?	
5. Using the new payment processing interface may impact the purchase completion latency of the AB system. Which stakeholders do you need to consult with about the performance requirements?	

Now, let's consider the third new capability, which adds cross-selling to the AB purchase user interface. Our approach is to have the AB system make a request to a Social Travel system when the purchase order has been placed, the necessary details about the travel order (e.g., traveler's gender, destination, date, activity type, etc.) and the Social Travel cross-sell business logic will return a list of five items to offer to the traveler.

Question	Your Answer
6. Does the AB system have existing request-response interfaces with external systems? If so, what protocols/technologies are used for these interfaces?	
7. Is the traveler's gender available in the AB system?	
8. Does an AB purchase order contain the activities that are included in the adventure?	

Finally, let's go back to the three questions that you listed at the very start of the exercise. Try to use the models to answer these questions—if one of your questions duplicates one of the previous questions, you can just note that here. In each answer, please note which models you consulted to make your decision. If you cannot find sufficient information in the models to make your decision, please state this.

Question	Your Answer
Your first question (There is no need to copy the question again here, unless you find that helpful)	
Your second question	
Your third question	

Two wrap-up questions:

Question	Your Answer
How much time did you spend on this exercise?	

Do you have any comments on the exercise? Did the role-playing questions represent the types of questions that an architect responsible for this integration might ask? Did you notice any gaps in the coverage of the models that was not exposed by the questions? Do you have any comments about the utility of the models in this viewpoint?

Your Answer:

This is the end of the exercise. Thank you for your contribution. We will share the results with you as soon as we complete the analysis.

References

[1] M. W. Maier, Architecting principles for systems-of-systems, Systems Engineering 1 (4) (1998) 267–284. doi:10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

[2] J. Klein, S. Cohen, R. Kazman, Common platforms in system-of-systems architectures: The state of the practice, in: Proc. IEEE/SMC Int. Conf. on System of Systems Eng., SoSE, 2013 [cited 1 Jan 2017].
URL http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=68315

[3] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, K. Wallnau, Ultra-large-scale systems: The software challenge of the future, Tech. rep., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2006) [cited 1 Apr 2017].
URL http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=30519

[4] ISO/IEC/IEEE, ISO/IEC/IEEE 42010: Systems and software engineering - architecture description, Standard, ISO/IEC/IEEE (2011).

[5] D. L. Parnas, D. M. Weiss, Active design reviews: principles and practices, in: Proc. 8th Int. Conf. on Software Engineering, ICSE '85, 1985, pp. 132–136.

[6] R. J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, Heidelberg, Germany, 2014.

[7] P. Clements, F. Bachman, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, 2nd Edition, Addison-Wesley, 2011.

[8] N. Rozanski, E. Woods, Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, Addison-Wesley, 2005.

[9] P. Kruchten, The Rational Unified Process: An Introduction, 3rd Edition, Addison-Wesley Professional, Boston, MA, USA, 2003.

[10] DOD Deputy Chief Information Officer, The DOD architecture framework version 2.02, Standard, United States Department of Defense (2010) [cited 18 Oct 2016].
URL http://dodcio.defense.gov/Library/DoD-Architecture-Framework/

[11] Ministry of Defence, MOD architecture framework (2012) [cited 1 Jan 2017].
URL https://www.gov.uk/guidance/mod-architecture-framework

[12] COMPASS. Comprehensive modelling for advanced systems of systems [online] (2014) [cited 1 June 2017].

[13] M. Guessi, V. V. G. Neto, T. Bianchi, K. R. Felizardo, F. Oquendo, E. Y. Nakagawa, A systematic literature review on the description of software architectures for systems of systems, in: Proc. 30th Ann. ACM Symp. on Applied Computing, SAC '15, 2015, pp. 1433–1440. doi:10.1145/2695664.2695795.

[14] J. Brøndum, L. Zhu, Towards an architectural viewpoint for systems of software intensive systems, in: Proc. 2010 ICSE Workshop on Sharing and Reusing Arch. Knowledge, SHARK '10, 2010, pp. 60–63. doi:10.1145/1833335.1833344.

[15] M. Mori, A. Ceccarelli, P. Lollini, B. Frömel, F. Brancati, A. Bondavalli, Systems-of-systems modeling using a comprehensive viewpoint-based sysml profile, Journal of Software: Evolution and Process Early View (2017) e1878–n/a. doi:10.1002/smr.1878.

[16] E. Honour, DANSE–an effective, tool-supported methodology for systems of systems engineering in Europe, in: Proc. 16th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2013 [cited 1 June 2017]. URL http://www.dtic.mil/ndia/2013/system/TH16282_Honour.pdf

[17] T. Bianchi, D. S. Santos, K. R. Felizardo, Quality attributes of systems-of-systems: A systematic literature review, in: Proc. 3rd Int. Workshop on Software Engineering for Systems-of-Systems, SESoS '15, 2015, pp. 23–30. doi:DOI10.1109/SESoS.2015.12.

[18] J. Klein, H. van Vliet, A systematic review of system-of-systems architecture research, in: Proc. 9th Int. ACM SIGSOFT Conf. on the Quality of Software Architectures, QoSA'13, ACM, Vancouver, BC, Canada, 2013, pp. 13–22. doi:10.1145/2465478.2465490.

[19] T. Batista, Challenges for SoS architecture description, in: Proc. 1st Int. Workshop on Software Engineering for Systems-of-Systems, SESoS '13, 2013, pp. 35–37. doi:10.1145/2489850.2489857.

[20] E. Silva, E. Cavalcante, T. Batista, F. Oquendo, F. C. Delicato, P. F. Pires, On the characterization of missions of systems-of-systems, in: Proc. of the 2014 European Conf. on Software Architecture Workshops, ECSAW '14, 2014, pp. 26:1–26:8. doi:10.1145/2642803.2642829.

[21] E. Woods, N. Rozanski, The system context architectural viewpoint, in: Proc. Joint European Conf. on Software Arch. and Working IEEE/IFIP Conf. on Software Arch., WICSA/ECSA'09, 2009, pp. 333–336. doi:10.1109/WICSA.2009.5290673.

[22] J. Bergey, J. Stephen Blanchette, P. Clements, M. Gagliardi, R. Wojcik, W. Wood, J. Klein, U.S. Army workshop on exploring enterprise, system of systems, system, and software architectures, Tech. Rep. CMU/SEI-2009-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2009) [cited 1 Jan 2017].
URL `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9099`

[23] ODUSD(A&T), Systems engineering guide for systems of systems, version 1.0, Guide, US Department of Defense (2008).
URL `http://www.acq.osd.mil/se/docs/SE-Guide-for-SoS.pdf`

[24] C. Benipayo, Understanding system interdependence to improve resilience of shipboard cyber physical system, in: Proc. 19th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2016 [cited 1 June 2017].
URL `http://www.dtic.mil/ndia/2016/systems/18881_CaesarBenipayo.pdf`

[25] V. Sitterle, Cross-scale resilience: Bridging system of systems and consitituent systems engineering and analysis, in: Proc. 19th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2016 [cited 1 June 2017].
URL `http://www.dtic.mil/ndia/2016/systems/18864_ValerieSitterle.pdf`

[26] J. Gump, An architecture for agile systems engineering of secure commercial-off-the-shelf (COTS) mobile communications, in: Proc. 17th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2014 [cited 1 June 2017].
URL `http://www.dtic.mil/ndia/2014/system/16845ThursTrack6Gump.pdf`

[27] J. Manas, Test perspectives for architecture, in: Proc. 17th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2014 [cited 1 June 2017].
URL `http://www.dtic.mil/ndia/2014/system/17006ThursTrack6Manas.pdf`

[28] R. Carson, Differentiating system architectures, in: Proc. 17th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2014 [cited 1 June 2017].
URL `http://www.dtic.mil/ndia/2014/system/16801ThursTrack6Carson.pdf`

[29] N. Guertin, Capability based technical reference frameworks for open system architecture implementations, in: Proc. 17th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2014 [cited 1 June

2017].
URL http://www.dtic.mil/ndia/2014/system/16909ThursTrack6Guertin.pdf

[30] K. Baldwin, J. Dahmann, Sos considerations in the engineering of systems, in: Proc. 17th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2014 [cited 1 June 2017].
URL http://www.dtic.mil/ndia/2014/system/16865ThursTrack2Baldwin.pdf

[31] M. Gagliardi, Identifying architectural challenges in system of systems architectures, in: Proc. 16th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2013 [cited 1 June 2017].
URL http://www.dtic.mil/ndia/2013/system/TH16018_Gagliardi.pdf

[32] W. Pritchett, Application of a ground system architecture framework using SysML, in: Proc. 16th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, 2013 [cited 1 June 2017].
URL http://www.dtic.mil/ndia/2013/system/W16096_Pritchett.pdf

[33] J. Dahmann, SoS pain points - INCOSE SoS working group survey, in: Proc. 15th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, San Diego, CA, 2012 [cited 27 Sep 2013].
URL http://www.dtic.mil/ndia/2012system/ttrack214770.pdf

[34] J. Dahmann, R. Heilmann, SoS Systems Engineering (SE) and Test & Evaluation (T&E): Final Report of the NDIA SE Division SoS SE and T&E Committees, in: Proc. 15th Ann. NDIA Systems Eng. Conf., NDIA, San Diego, CA, 2012 [cited 27 Sep 2013].
URL http://www.dtic.mil/ndia/2012system/ttrack214771.pdf

[35] J. Smith, P. Place, M. Novakouski, D. Carney, Examining the role of context in data interoperability, in: Proc. 14th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, San Diego, CA, 2011 [cited 27 Sep 2013].
URL http://www.dtic.mil/ndia/2011system/13101_SmithWednesday.pdf

[36] J. A. Lane, J. Dahmann, G. Rebovich, R. Lowry, Key system of systems engineering artifacts to guide engineering activities, in: Proc. 13th Ann. NDIA Systems Eng. Conf., National Defense Industry Association, San Diego, CA, 2010 [cited 27 Sep 2013].
URL http://www.dtic.mil/ndia/2010systemengr/WednesdayTrack3_10806Lane.pdf

[37] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd Edition, Addison-Wesley, 2013.

[38] T. Dyba, T. Dingsoyr, G. K. Hanssen, Applying systematic reviews to diverse study types: An experience report, in: Proc. 1st Int. Symp. on Empirical Software Eng. and Measurement, ESEM 2007, 2007, pp. 225–234. doi:10.1109/ESEM.2007.59.

[39] M. A. P. M. van den Bosch, M. E. van Steenbergen, M. Lamaitre, R. Bos, A selection-method for enterprise application integration solutions, in: Proc. 9th Int. Conf. on Business Informatics Research, BIR 2010, 2010, pp. 176–187. doi:10.1007/978-3-642-16101-8_14.

[40] S. Beecham, T. Hall, C. Britton, M. Cottee, A. Rainer, Using an expert panel to validate a requirements process improvement model, Journal of Systems and Software 76 (3) (2005) 251–275. doi:10.1016/j.jss.2004.06.004.

[41] C. Hakim, Contemporary Social Research: 13, Routledge, London, UK, 1987, Ch. Research Design: Strategies and Choices in the Design of Social Research.

[42] R. Nord, P. C. Clements, D. Emery, R. Hilliard, A structured approach for reviewing architecture documentation, Technical Note CMU/SEI-2009-TN-030, Software Engineering Institute, Pittsburgh, PA (December 2009) [cited 22 Dec 2013].
URL http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9045#

[43] B. A. Kitchenham, L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734. doi:10.1109/TSE.2002.1027796.

[44] E. R. Poort, H. van Vliet, RCDA: Architecting as a risk- and cost management discipline, Journal of Systems and Software 85 (9) (2012) 1995–2013. doi:10.1016/j.jss.2012.03.071.

[45] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2) (2009) 131–164. doi:10.1007/s10664-008-9102-8.

[46] R. Yin, Case Study Research: Design and Methods, 3rd Edition, Applied Social Research Methods Series, Sage Publications, 2003.