The Process of Architecting for Software / System Engineering

AMINE CHIGANI
OSMAN BALCI

Department of Computer Science
Virginia Polytechnic Institute and State University (Virginia Tech)
Blacksburg, Virginia 24061, U.S.A.

September 9, 2010

Contact Author: Prof. Osman Balci
balci@vt.edu
http://manta.cs.vt.edu/balci
Abstract

With the advent of potent network technology, software/system engineering has evolved from a traditional platform-centric focus into a network-centric paradigm where the “system of systems” perspective has been the norm. Under this paradigm, architecting has become a critical process in the life cycle of software/system engineering. The need for a structured description of the architecting process is undeniable. This paper fulfills that need and provides a structured description of the process of architecting a software-based network-centric system of systems. The architecting process is described using a set of goals that are specific to architecting, and the associated specific practices that enable the realization of these goals. The architecting process description presented herein is intended to guide the software/system architects.

Keywords: architecting process, network-centric system architecting, process area, software architecting, software architecture, software-based system architecting, system architecting, system of systems architecting.
1 Introduction

The ubiquity of the network and the ability to deploy software over a network has changed the underpinnings of software-based solutions. The “system of systems” perspective [Maier 1998] dominates much of the engineering now as new systems are composed of multiple interconnected systems to support emerging missions. One reason behind this shift is the need to reach beyond tightly-coupled environments to access data and functionality that reside on remote systems deployed on different platforms, and which are possibly owned and managed by different entities. Another reason is the dynamic and complex structures of today’s organizations, where the organization’s computing resources can span multiple national and international locations.

These changes gave rise to the Software-as-a-Service (SaaS) paradigm, where software-based systems no longer reside on users’ own computing devices. Instead, software-based solutions are provided as services over a network, and users access these services through a plethora of network-capable devices that range from mainframe computers to smart phones.

Under the SaaS paradigm, architecting has become a distinct and essential life cycle process for software engineering as well as system engineering. Today, architecting is well-recognized in practice as evidenced by job titles such as Software Architect, Solution Architect, Enterprise Architect, Application Architect, Integration Architect, and Information Architect. In addition, several programs exist to confer professional certification and certificates to architecture practitioners. Most of such programs are organizationally-based promoting individuals along the organization ladder, and are typically focused on in-house technologies and products. Examples of such programs include:

- IBM Professional Certification Program [IBM 2010].
- Microsoft Certified Architect Program [Microsoft 2010a].
- Raytheon Certified Architect Program [Raytheon 2010].

Other programs exist to grant professional architecture certification to practitioners who are interested in going through the certification process. Examples of such initiatives include:

- The International Association of Software Architects (IASA): Certified IT Architect Program [IASA 2010].
- The Open Group: IT Architect Certification Program (ITAC) [The Open Group 2010].

On the one hand, the architecting discipline has moved from mere qualitative observations of the structure of working systems into a rich repertoire of concepts, methods, standards, frameworks, and tools to create, document, analyze, and assess architecture specifications. Architecting has become an “indispensable” process in the software/system engineering life cycle [Shaw and Clements 2006]. This indispensability makes it vital for software-based solution providers to execute best architecting practices in order to deliver quality solutions.

On the other hand, the adoption of the current architecting knowledge is dependent on the existence of a defined architecting process that integrates with system/software life cycle processes. Processes are important to streamlining the work required to develop a software-based solution from inception through operation and to retirement. Solution providers typically focus on three dimensions of their development approaches: people, methods, and tools [SEI 2006]. To link these dimensions together, organizations follow processes of the software/system engineering life cycle to ensure the production of systems with
the desired functionality and quality, on time, and within budget. However, published life cycle models leave out architecting from their processes.

The Capability Maturity Model Integration for Development (CMMI-DEV) is a widely-used process improvement and appraisal approach, which provides a reference point to organizations to assess their competencies in process areas related to software/system development [SEI 2006]. A process area is a set of specific practices that achieve a set of specific goals for making improvement in that area. A specific goal of a process area is a distinctive objective that must be met in order to satisfy that area. A specific practice is an activity that must be carried out to achieve part or all of a corresponding specific goal.

CMMI’s basic philosophy is that the quality of a product or service is positively-correlated with the processes used to develop such product or service. However, CMMI also leaves out architecting as one of its 22 defined process areas. Although CMMI helps organizations identify process areas crucial to developing quality products and provides a framework to improve such processes, architecture is briefly discussed as part of a Specific Practice (SP 2.1 Design the Product or Product Component) of a Specific Goal (Develop the Design) of the Technical Solution Process Area.

Evidently, architecting requires more prominence in the life cycle than what it currently has in the CMMI models [Valerdi et al.]. We advocate that architecting should have its own process area. The purpose of this paper is to provide a structured description of the process of architecting software-based network-centric system of systems. We propose that such a description of the architecting process be added as a CMMI process area at Level 3. We describe the architecting process area following the CMMI-DEV structure. However, for the purposes of this paper we only emphasize the specific goals and specific practices of the architecting process area.

The remainder of this paper is organized as follows. Section 2 presents an overview of the architecting process area. The specific goals and practices of the architecting process are described in Section 3. Finally, concluding remarks are provided in Section 4.

2 The Architecting Process Area

We define the architecting process area and propose it as an Engineering Process Area of the CMMI-DEV at Maturity Level 3. Engineering process areas are processes that span the entire development life cycle of a system from its inception to retirement. Maturity Level 3 process areas are established processes within the organization and are executed in a proactive manner.

The purpose, activities, and performers of the architecting process, as well as the uses of architecture specification are described below.

2.1 The Purpose of the Architecting Process

The purpose of architecting is to develop an architecture specification for a software-based system (or a system of systems). The process of architecting takes the Problem Specification and Requirements Specification as input and produces an Architecture Specification as an output work product.

The development of an architecture specification focuses on several objectives. The primary objective is to ensure that system quality attributes (i.e., non-functional requirements) such as interoperability and security are architected for early on in the development life cycle. Another objective is to ensure that the architecture specification is a usable asset for decision makers to evaluate investment alternatives of resources such as systems, system components, and middleware technologies that will be used in the
development of the system. Moreover, the development of an architecture specification focuses on ensuring that the produced architecture is represented from different perspectives to facilitate communication among various stakeholders.

2.2 Activities of the Architecting Process

The architecting process area encompasses all activities performed to produce artifacts constituting the architecture specification. An architecture specification describes the fundamental organization of components, the relationships among these components, the mapping of these components to their environment, and the principles and guidelines governing the system design and evolution [IEEE 2007].

The activities of the architecting process focus on:

- Identifying system components and establishing the relationships among them based on the system requirements.
- Creating an architecture that satisfies the requirements by either choosing an existing architecture, making up a composite architecture, or creating an architecture from scratch.
- Describing the architecture using a multi-view description approach and evaluating it to ensure that it meets the system requirements.

The architecting process typically follows a systems perspective [Kossiakoff and Sweet 2003]. It looks at the system as a whole. Externally, the architecting process takes into account interfacing systems, the operating environment, and the users of the system. Internally, it focuses on the architectural arrangements that represent the system. Therefore, the architecture specification must include software, hardware, and human (when appropriate) components.

2.3 Performers of the Architecting Process

The architecting process is carried out under different roles. The roles and job titles of architecting practitioners vary by organization and domain. However, these roles can be categorized as follows:

- A single architect who is solely responsible for the entire architecting process.
- A team of architects (also referred to as the architecture team) who collaborate throughout the entire architecting process. In this case, there is often a lead architect who manages the team.
- A single architect who leads a team of engineers (for requirements, hardware, network, database, usability, security, etc.) throughout the entire architecting process.

We use the term “architect” to refer to any of the roles listed above.

2.4 Uses of Architecture Specification

An architecture specification is used by several stakeholders for various reasons including [DoDAF 2009a, b, c]:

- Making acquisition decisions about whole systems, system components, and technologies that can be used in the development.
- Discovering the true requirements of the system and refining existing ones.
- Creating an overall strategy for aligning independent systems into a common use or purpose (i.e., system of systems).
• Evolving the system by deploying new technologies, adding services, and replacing/retiring old capabilities.

Equally important, architecture specifications are used as means of communication between the architect and the stakeholders, and among stakeholders. Therefore, architecture specification is produced from multiple perspectives (viewpoints) to enable such communication since stakeholders often have disparate stakes in the architecture.

CMMI-DEV defines a process area by identifying the specific goals of that area and the associated specific practices that enable the realization of those goals. We follow the CMMI-DEV structure and describe the specific goals and practices of the architecting process area below.

### 3 Specific Goals and Practices of the Architecting Process

We define the process of architecting for software/system engineering in accordance with the CMMI-DEV structure by using a set of specific goals (SGs) and their associated specific practices (SPs) as shown in Table 1. Each SG and SP is described below.

**Table 1** Specific goals and practices of the architecting process

<table>
<thead>
<tr>
<th>SG 1: Identify System Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 1.1: Identify Architecturally-Relevant Requirements</td>
</tr>
<tr>
<td>SP 1.2: Decompose the System into Components</td>
</tr>
<tr>
<td>SP 1.3: Assess the Component Decomposition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SG 2: Establish Relationships among the Identified Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 2.1: Establish Structural Relationships</td>
</tr>
<tr>
<td>SP 2.2: Establish Behavioral Relationships</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SG 3: Create the Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 3.1: Select an Existing Architecture</td>
</tr>
<tr>
<td>SP 3.2: Make Up a Composite Architecture</td>
</tr>
<tr>
<td>SP 3.3: Create an Architecture from Scratch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SG 4: Describe the Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 4.1: Describe the Architecture from an All Perspective</td>
</tr>
<tr>
<td>SP 4.2: Describe the Architecture from a Capability Perspective</td>
</tr>
<tr>
<td>SP 4.3: Describe the Architecture from a Data and Information Perspective</td>
</tr>
<tr>
<td>SP 4.4: Describe the Architecture from an Operational Perspective</td>
</tr>
<tr>
<td>SP 4.5: Describe the Architecture from a Project Perspective</td>
</tr>
<tr>
<td>SP 4.6: Describe the Architecture from a Services Perspective</td>
</tr>
<tr>
<td>SP 4.7: Describe the Architecture from a Standards Perspective</td>
</tr>
<tr>
<td>SP 4.8: Describe the Architecture from a Systems Perspective</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SG 5: Evaluate the Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP 5.1: Evaluate the Architecture Based on the Four Ps</td>
</tr>
<tr>
<td>SP 5.2: Evaluate the Architecture Following a Risk-Driven Approach</td>
</tr>
<tr>
<td>SP 5.3: Evaluate the Architecture Based on a Set of Scenarios</td>
</tr>
</tbody>
</table>
3.1 **SG 1: Identify System Components**

**Goal Statement:** System components are identified to satisfy the requirements.

The architecting process starts with identifying the software, hardware, and human (e.g., users, operators) components that will constitute the system under development. Human components may or may not be relevant parts of the architecture depending on the context of the system. These components should be identified so as to satisfy the requirements and provide the mandated capabilities. To achieve this goal, the following specific practices are employed.

3.1.1 **SP 1.1: Identify Architecturally-Relevant Requirements**

**Practice Statement:** Requirements that are germane to the architecture are identified to guide the architecting process.

Before identifying the components that will make up the system architecture, the requirements that are germane to the architecture should be identified. Architecturally-relevant requirements are those that have far-reaching implications on the architecture and span more than one component of the system.

Architecturally-relevant requirements include the functional requirements that describe major capabilities of the system – capabilities without which the system cannot satisfy the purposes intended for its use. Such requirements are important to identify system components and the relationships among them.

Architecturally-relevant requirements also include non-functional requirements (also referred to as quality characteristics or attributes) such as dependability, interoperability, and performance. Such quality characteristics are germane to the architecting process because they affect the quality of the entire system.

Quality attributes often tradeoff. For instance, enhancing security by adding extra layers of authentication and data encryption introduces a communication overhead and therefore affects the performance negatively. Thus, it is essential to identify such tradeoffs among quality characteristics to ensure that proper architecture decisions are made when creating the architecture.

The outcome of this specific practice should consist of a list of architecturally-relevant requirements – functional and non-functional. In addition, this specific practice should produce a set of potential tradeoffs that may exist among the identified quality characteristics.

3.1.2 **SP 1.2: Decompose the System into Components**

**Practice Statement:** The system is decomposed into components that satisfy the identified architecturally-relevant requirements.

The system should be decomposed into software, hardware, and human components based on the requirements that the architecture must satisfy. A component is a conceptual representation of an identifiable part of a system [Kossiakoff and Sweet 2003]. A component can be a software or hardware module that encapsulates a set of related functions, data, or responsibilities associated with a user or operator.

A component is identified by mapping a function (or a set of related functions) that the system must perform into a conceptual representation that will carry out that function (or those related functions). Each
component should be given a name that describes its functions. Techniques such as modularization, separation of concerns, and information hiding should be applied in component identification.

The quality of a component is determined by examining two characteristics: cohesion and coupling. In principle, a component should be put together to have the highest possible cohesion and the lowest possible coupling. **Cohesion** is the degree to which the component’s elements are related to each other. It depicts the “togetherness” among the elements comprising the component. A component with high cohesion directs all its elements towards achieving a single objective. **Coupling**, on the other hand, is the degree to which a component is built based on the logic of another component. Low coupling is desired, implying that there is minimal or no logical dependence among the components. When components are created with the highest possible cohesion and the lowest possible coupling, several architecture quality characteristics are enhanced, namely maintainability and modifiability.

Decomposition or modularization of a system into components continues until the component at the leaf node (i.e., the one that is no further decomposed) possesses the highest possible cohesion and the lowest possible coupling. Two approaches are commonly used to carry out this decomposition: vertical and horizontal.

**Vertical decomposition** slices the system vertically by decomposing it in a top-down fashion. It starts by decomposing the system into components at level 1. A level 1 component is selected for decomposition into components at level 2. Then, a level 2 component is selected for decomposition into components at level 3. This decomposition continues until reaching the leaf component that has the highest possible cohesion and the lowest possible coupling.

**Horizontal decomposition** modularizes the system level by level. It starts by decomposing the system into all of the components at level 1. Each level 1 component is further decomposed into components at level 2. Then, each level 2 component is further decomposed into components at level 3. All of the components at a level must be identified before proceeding to the next level. When a component is identified as possessing the highest possible cohesion and the lowest possible coupling, it is designated as a leaf component and is not decomposed further.

Both vertical and horizontal decomposition approaches can be intertwined for the same architecting project.

### 3.1.3 SP 1.3: Assess the Component Decomposition

**Practice Statement:** Decomposition of the system into components is assessed against all architecturally-relevant requirements.

Component decomposition should be assessed based on coverage and depth. The identified components should cover all of the requirements identified in SP 1.1. A matrix should be created to map each architecturally-relevant requirement to a component (or a set of components) that addresses that requirement. Gaps identified in this mapping should be addressed by either mapping a requirement to an already-identified component or by identifying a new component. Identifying a new component to address a particular requirement should follow the guidelines outlined in SP 1.2.

Component decomposition should also be assessed based on depth. Decomposition depth is evaluated by examining abstraction level and cohesion and coupling of each component. If a component is found to have an undesirable amount of cohesion and/or coupling, the principle of separation of concerns [Dijkstra
1982] should be applied to determine whether it should be decomposed further or combined with another component.

This specific practice ensures that each architecturally-relevant requirement can be addressed by the identified components and that each component is properly created.

### 3.2 SG 2: Establish Relationships among the Identified Components

**Goal Statement:** Relationships among identified components are established.

Relationships among the identified components should be established. A relationship defines how two components relate to each other in the context of the system being architected. Many perspectives exist from which a relationship can be established; for example, based on structure or behavior.

The Specific Practices that enable the accomplishment of the goal stated above are described below.

#### 3.2.1 SP 2.1: Establish Structural Relationships

**Practice Statement:** Structural relationships among identified components are established.

Structural relationships represent how the components can be structurally organized to form a unified blueprint of the system. This perspective focuses on the static layout of these components, which can be either a logical or a physical layout.

The logical layout of components is concerned with the conceptual grouping of these components. The following are example types of logical relationships:

- Dependency: Component A depends-on Component B.
- Use: Component A uses Component B.
- Hierarchy: Component A is-part-of Component B or Component A is-a-child-of Component B.

The physical layout of components is concerned with relationships among software, hardware, infrastructure, and deployment components. The following are example types of physical relationships:

- Interfacing: Relationships that highlight the interfaces between two hardware components, between a software component and a hardware component, or between a software/hardware component and a human operator/user.
- Deployment: Relationships that highlight, among other aspects, the distribution of components over the network, resource-sharing, and communication channels.

#### 3.2.2 SP 2.2: Establish Behavioral Relationships

**Practice Statement:** Behavioral relationships among identified components are established.

Behavioral relationships among components represent interactions. This perspective focuses on the dynamic behavior of the system during execution. Behavior can be represented in many ways including processes, events, and services.
- Process: Component A starts/pauses/stops a process X.
- Service: Component A sends a request to Component B and Component B processes the request and may send a reply back.
- Event: Component A publishes a notification and one or more components subscribe to receive that notification.

Structural (static) and behavioral (dynamic) relationships help in the identification of a suitable architecture. For instance, the decomposition of the system into components may reveal that some components are responsible for producing data or performing tasks for other components. In such a case, the client-server architecture may seem plausible as a candidate architecture for the system under development. Therefore, the set of relationships identified in this specific practice provides the foundation to begin the creation of the system architecture.

### 3.3 SG 3: Create the Architecture

**Goal Statement:** System architecture is created based on the identified components and the relationships among them.

The identified components and the relationships that exist among them form the basis of the overall architecture. Architecture creation, therefore, consists of making decisions about which architecture(s) best satisfies the requirements of the system.

Three approaches to making architecture decisions exist: (1) select an existing architecture, (2) make up a composite architecture from existing ones, or (3) create an architecture from scratch.

#### 3.3.1 SP 3.1: Select an Existing Architecture

**Practice Statement:** An existing architecture is selected as the overarching architecture of the system.

The major architectures that have been in use for architecting a software-based network-centric system of systems include: Client-Server Architecture, Distributed Objects Architecture, Peer-to-Peer Architecture, and Service-Oriented Architecture. For a given architecting task at hand, one of these architectures can be selected if it satisfies the requirements of the system under development.
3.3.1.1 Client-Server Architecture

The client-server architecture (CSA) consists of the following five conceptual layers designated with circled numbers in Figure 1:

![Conceptual layers of the client-server architecture](image)

**Figure 1** Conceptual layers of the client-server architecture

*Layer 1.* Data Source Layer (e.g., a relational database management system)

*Layer 2.* Data Mapping Layer (e.g., Entity Enterprise Java Beans, ActiveX Data Objects)

*Layer 3.* Business / Application Logic Layer

*Layer 4.* Web Container Layer (consisting of controller/mediator and server-side presentation preparation components)

*Layer 5.* Client Presentation Layer (e.g., Asynchronous JavaScript and XML (AJAX), Cascading Style Sheets (CSS), Document Object Model (DOM), Extensible HTML (XHTML), Extensible Markup Language (XML), Extensible Stylesheet Language Transformation (XSLT), HyperText Markup Language (HTML))

Two major industry standards can be employed to build a system under CSA: Java Platform, Enterprise Edition (Java EE) [Oracle 2010] and Microsoft .NET Framework [Microsoft 2010b].
3.3.1.2  Distributed Objects Architecture

The distributed objects architecture (DOA) consists of objects, which are software applications that run on distributed computers with different hardware and operating systems. An object $O_j$ can act as a server and provide services $S(O_j)$ or act as a client and consume services as depicted in Figure 2. The communication among objects is mediated by the object request broker (ORB). Objects can be implemented in different programming languages and can be integrated with ORB using an interface definition language (IDL). A system created under DOA can be connected to another DOA-based system using the Inter-ORB communications over a network to form a system of systems.

Two major industry standards can be employed to build a system under DOA: Common Object Request Broker Architecture (CORBA) [OMG 2010] and Distributed Component Object Model (DCOM) [Microsoft 1996].

![Figure 2 Conceptual representation of the distributed objects architecture](image-url)
3.3.1.3 Service-Oriented Architecture

The service-oriented architecture (SOA) consists of the conceptual layers as depicted in Figure 3. Layers 3 to 6 in Figure 3 make up the major components of SOA built on top of the network structure consisting of layers 0 to 2.

![Figure 3 Conceptual layers of SOA](Glass2008;BalciandOrmsby2008)

3.3.1.4 Peer-to-Peer Architecture

The peer-to-peer (P2P) architecture consists of a set of networked computers, where each computer makes some of its resources (e.g., computational power, storage, music files, and technical documents) directly available to other computers on the network as depicted in Figure 4. A P2P computer (node) acts as a supplier or consumer of resources.
3.3.2 **SP 3.2: Make Up a Composite Architecture**

**Practice Statement:** Two or more known architectures are selected to make up a composite architecture of the system.

The second approach to architecture creation is to combine a number of known architectures to create a composite architecture for the system under development. Known architectures include the ones described in SP 3.1. In this context, known architectures are used to describe various parts of the system. The result is an overall architecture that is made up of several known architectures.

Figure 5 shows an example composite architecture of a system consisting of CSA and DOA. This is the architecture of a simulation development environment intended for use by geographically-dispersed users over the network. The overall architecture is CSA. However, the CSA’s business/application logic layer is developed using the CORBA Component Model (CM) to satisfy the requirement for a heterogeneous execution environment. Thus, a composite architecture is created by combining CSA with DOA.

**Figure 5** A composite architecture consisting of CSA and DOA
3.3.3  **SP 3.3: Create an Architecture from Scratch**

**Practice Statement:** *A new architecture is created from scratch based on the identified components and relationships.*

The third approach to architecture development is to create an architecture from scratch. In such a case, the architect uses the identified components and their relationships as the building blocks of the new architecture. Therefore, the main purpose in this case is to come up with a novel organization of these components.

This approach is suitable in cases where known architectures do not provide the appropriate structure needed to satisfy the requirements. Such architectures are sometimes called custom-made or domain-specific architectures [Agrawala *et al.* 1992].

Custom-made architectures should follow basic guidelines such as the following.

- The new architecture should propose a new structure that is not represented by a known architecture.
- The new structure should specify the types of components, the types of relationships among them, and should provide guidelines specific to how the architecture can be instantiated into a particular design.
- The new architecture should be applicable to create architectures for similar systems. This is because architectures provide reusable abstractions for a particular class of problems. A new architecture should provide a new architecture solution to a class of systems.

3.4  **SG 4: Describe the Architecture**

**Goal Statement:** *Architecture description consists of a set of models representing multiple viewpoints of the system.*

A network-centric system of systems architecture is a complex entity that cannot be described using only one representation. We slice this complex entity in many different ways and come up with a representation for each slice. DoDAF [2009a, b, c] provides 52 representations (models) under the following eight viewpoints (perspectives) to describe an architecture.

1. All Viewpoint (AV)
2. Capability Viewpoint (CV)
3. Data and Information Viewpoint (DIV)
4. Operational Viewpoint (OV)
5. Project Viewpoint (PV)
6. Services Viewpoint (SvcV)
7. Standards Viewpoint (StdV)
8. Systems Viewpoint (SV)

No one viewpoint can properly represent the entire architecture of a system. The best practice is to describe the architecture under multiple viewpoints using a number of representations (e.g., DoDAF-described models) under each viewpoint. Which viewpoints and which representations in each viewpoint to use depend on the requirements specification given for a particular system [IEEE 2007; DoDAF 2009a, b, c; Clements *et al.* 2003; Kruchten 1995].
The Specific Practices that enable the accomplishment of the goal stated above are described below.

### 3.4.1 SP 4.1: Describe the Architecture from All Viewpoint

**Practice Statement:** Create models that describe aspects of the architecture that are pertinent to all viewpoints.

The all viewpoint (AV) is concerned with information pertinent to the entire architecture specification.

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- AV-1 Overview and Summary Information
- AV-2 Integrated Dictionary

DoDAF AV models describe the architecture effort with its scope, context, and findings, and defines a common terminology (definitions of terms) used throughout the entire architecture specification.

### 3.4.2 SP 4.2: Describe the Architecture from Capability Viewpoint

**Practice Statement:** Create models that describe the architecture from capability viewpoint.

The capability viewpoint (CV) is concerned with the description of the capabilities provided by the system. A *capability* is “the ability to achieve a desired effect under specified standards and conditions through combinations of means and ways to perform a set of tasks” [DoDAF 2009b, p. 80].

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- CV-1 Vision
- CV-2 Capability Taxonomy
- CV-3 Capability Phasing
- CV-4 Capability Dependencies
- CV-5 Capability to Organizational Development Mapping
- CV-6 Capability to Operational Activities Mapping
- CV-7 Capability to Services Mapping

DoDAF CV models are used to represent the architecture by describing:

- a strategic context for the capabilities,
- a hierarchy (taxonomy) of capabilities,
- planned achievement of capabilities at different points in time,
- dependencies between planned capabilities,
- logical groupings of capabilities,
- fulfillment of capability requirements,
- a mapping between the capabilities required and the operational activities that those capabilities support, and
- a mapping between the capabilities and the services that these capabilities enable.
3.4.3 **SP 4.3 Describe the Architecture from Data and Information Viewpoint**

**Practice Statement:** Create models that describe the architecture from data and information viewpoint.

The data and information viewpoint (DIV) is concerned with conceptual, logical, and physical levels of abstraction in representing the operational and business data and information.

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- DIV-1 Conceptual Data Model
- DIV-2 Logical Data Model
- DIV-3 Physical Data Model

DoDAF DIV models describe high-level data concepts and their relationships, data requirements and structural business process rules, and physical implementation format.

3.4.4 **SP 4.4: Describe the Architecture from Operational Viewpoint**

**Practice Statement:** Create models that describe the architecture from operational viewpoint.

The operational viewpoint (OV) is concerned with the overall context within which the system operates and the external systems with which the system interacts. Describing the architecture from this viewpoint requires capturing all system elements in one or more models at a high level of abstraction.

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- OV-1 High-Level Operational Concept Graphic
- OV-2 Operational Resource Flow Description
- OV-3 Operational Resource Flow Matrix
- OV-4 Organizational Relationships Chart
- OV-5a Operational Activity Decomposition Tree
- OV-5b Operational Activity Model
- OV-6a Operational Rules Model
- OV-6b State Transition Description
- OV-6c Event-Trace Description

DoDAF OV models are used to represent the architecture by describing:

- a high-level graphical operational concept,
- resource flows exchanged between operational activities,
- resources exchanged and the relevant attributes of the exchanges,
- organizational context, role or other relationships among organizations,
- capabilities and operational activities organized in a hierarchical structure,
- context of capabilities and operational activities and their relationships among activities, inputs, and outputs,
- business rules that constrain operations,
• business process (activity) responses to events, and
• traces of actions in a scenario or sequence of events.

OV models serve as a means of providing the big picture of what the architecture is and what the system is supposed to do. In addition, OV models form the architecture basis for communication among non-technical stakeholders such as managers, customers, and decision makers.

3.4.5 **SP 4.5 Describe the Architecture from Project Viewpoint**

**Practice Statement:** Create models that describe the architecture from project viewpoint.

The project viewpoint (PV) is concerned primarily with architecture models that describe project management activities. This viewpoint addresses assignments of development efforts and project planning.

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- PV-1 Project Portfolio Relationships
- PV-2 Project Timelines
- PV-3 Project to Capability Mapping

DoDAF PV models focus on the following aspects of the architecture:

- **Development:** Models in this category describe how architecture components should be assigned to development teams. They also highlight the interdependencies among the components to ensure that proper planning of milestones is achieved and that development bottlenecks are avoided.
- **Planning:** Models in this category describe aspects of the system that relate to project management such as cost estimation, commercial-off-the-shelf (COTS) components, and release planning.

3.4.6 **SP 4.6: Describe the Architecture from Services Viewpoint**

**Practice Statement:** Create models that describe the architecture from services viewpoint.

The services viewpoint (SvcV) is concerned with describing the services that provide access to system capabilities. In general, a service is a means to access a system capability [DoDAF 2009b], and has a predefined description of how it can be invoked [Erl 2008].

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- SvcV-1 Services Context Description
- SvcV-2 Services Resource Flow Description
- SvcV-3a Systems-Services Matrix
- SvcV-3b Services-Services Matrix
- SvcV-4 Services Functionality Description
- SvcV-5 Operational Activity to Services Traceability Matrix
DoDAF SvcV models are used to represent the architecture by describing:

- services, service items, and their interconnections,
- resource flows exchanged between services,
- relationships among or between systems and services,
- relationships among services,
- functions performed by services and the service data flows among service functions (activities),
- mapping of services (activities) back to operational activities (activities),
- service resource flow elements being exchanged between services and the attributes of that exchange,
- measures (metrics) of services,
- evolution of services over time, and
- rules, transition, and trace of services.

### 3.4.7 SP 4.7: Describe the Architecture from Standards Viewpoint

**Practice Statement:** Create models that describe the architecture from standards viewpoint.

The standards viewpoint (StdV) is concerned with the documentation of the required standards used and that must be adhered to during the development and operation of the system.

DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- StdV-1 Standards Profile
- StdV-2 Standards Forecast

DoDAF StdV models describe operational, business, technical, and industry policies, standards, guidance, constraints, and forecasts that are applicable to capability and operational requirements, system engineering processes, systems, and services.

### 3.4.8 SP 4.8 Describe the Architecture from Systems Viewpoint

**Practice Statement:** Create models that describe the architecture from systems viewpoint.

The systems viewpoint (SV) is concerned with describing component systems that make up the overall system. This viewpoint is relevant for the architecture specification of systems of systems, network-centric systems, and families of systems.
DoDAF [2009a, b, c] provides the following models to represent the architecture from this viewpoint:

- SV-1  Systems Interface Description
- SV-2  Systems Resource Flow Description
- SV-3  Systems-Systems Matrix
- SV-4  Systems Functionality Description
- SV-5a Operational Activity to Systems Function Traceability Matrix
- SV-5b Operational Activity to Systems Traceability Matrix
- SV-6  Systems Resource Flow Matrix
- SV-7  Systems Measures Matrix
- SV-8  Systems Evolution Description
- SV-9  Systems Technology and Skills Forecast
- SV-10a Systems Rules Model
- SV-10b Systems State Transition Description
- SV-10c Systems Event-Trace Description

DoDAF SV models are used to represent the architecture by describing:

- systems, system items, and their interconnections,
- resource flows exchanged between systems,
- relationships among systems,
- the functions (activities) performed by systems and the system data flows among system functions (activities),
- a mapping of system functions (activities) back to operational activities (activities),
- a mapping of systems back to capabilities or operational activities (activities),
- system resource flow elements being exchanged between systems and the attributes of that exchange,
- measures (metrics) of systems, and
- evolution, rules, transition, and trace of systems.

3.5 **SG 5: Evaluate the Architecture**

| Goal Statement: The architecture is evaluated to determine how well it enables the system to exhibit the required quality characteristics. |

System architecture influences many system quality indicators (attributes or characteristics) such as adaptability, dependability, deployability, extensibility, interoperability, maintainability, modifiability, openness, performance, scalability, survivability, and testability [Balci and Ormsby 2008]. The goal of architecture evaluation is to find out how well an architecture enables a system to possess a desired set of quality characteristics under a given set of intended uses of the system.

Architecture evaluation should be conducted by using a methodology such as the ones listed below:

- MSAAM: Military System Architecture Assessment Methodology [Balci and Ormsby 2008]
- ATAM: Architecture Tradeoff Analysis Method [Bass *et al.* 2003]
- SAAM: Software Architecture Analysis Method [Kazman *et al.* 1994]
- CBAM: Cost-Benefit Analysis Method [Nord *et al.* 2003]
- ALMA: Architecture Level Modifiability Analysis [Lassing 2002]
The Specific Practices that enable the accomplishment of the goal stated above are described below.

### 3.5.1 SP 5.1 Evaluate the Architecture Based on the Four Ps

**Practice Statement:** The architecture is evaluated from four perspectives: product, process, project, and people.

Architecture evaluation is considered to be a confidence building activity. The more comprehensive and detailed the evaluation is, the more confidence we can gain in the evaluation. Four major perspectives or four Ps influence the architecture quality: product, process, people, and project. Architecture evaluation can be approached from any one of the four Ps, but a combination of all four provides the best balance and results in a much higher level of confidence in the evaluation. Therefore, an architecture should be assessed from the four perspectives by way of assessing: [Balci and Ormsby 2008]

1. the architecture itself as a **product**,
2. the **process** used in creating the architecture,
3. the quality of the **people** employed in creating the architecture, and
4. architecture development **project** characteristics (e.g., capability maturity, documentation, planning, risk management).

### 3.5.2 SP 5.2 Evaluate the Architecture Following a Risk-Driven Approach

**Practice Statement:** A risk-driven evaluation approach is employed to ensure early identification and mitigation of risks.

A risk-driven approach is advocated for architecture evaluation. The evaluation should consider all potential risks including the following: [Balci and Ormsby 2008]

- **Acceptance Risk** is the probability that the system architecture will not pass the acceptance test with respect to the prescribed acceptance criteria.
- **Integration Risk** is the probability that the system architecture components will not be successfully integrated.
- **Interface Risk** is the probability that the system architecture will not successfully interface with the required external systems.
- **Performance Risk** is the probability that the performance of the system to be built based on the proposed architecture will not be acceptable.
- **Reliability Risk** is the probability that the reliability of the system to be built based on the proposed architecture will not be acceptable.
- **Reproducibility Risk** is the probability that a component of the proposed system architecture cannot be reproduced. For example, if the component relies on a piece of specialized hardware and the specialized hardware is no longer in production, then the system may not be reproduced at a reasonable cost.
- **Supportability Risk** is the probability that the system to be built based on the proposed architecture cannot be properly maintained after its delivery. For example, there may be no
technical support, no support for correcting errors, and no support for making improvements and upgrades.

- **Technological Risk** is the probability that one or more components of the proposed system architecture depend on undeveloped technologies that cannot be developed in an acceptable time frame.
- **Utility Risk** is the probability that the system to be built based on the proposed architecture will be less useful than required by the system stakeholders.

4 Concluding Remarks

Architecting should be designated as a process between the Requirements Engineering and Design processes in the life cycle models for software engineering as well as for systems engineering. The architecting process plays a critically important role in engineering a software-based network-centric system of systems.

Architecting a network-centric enterprise system, a system of systems, or a family of systems poses significant complexity. The complexity can be overcome by way of modularization. The DoD architecture framework decomposes the architecture representation into eight viewpoints (perspectives) and defines a total of 52 models to represent the architecture under these viewpoints. Which models to build for a particular project depends on the requirements specification document. Certainly, developing all 52 models is an onerous task requiring significant resources and time.

References


