Enter Project Name Here

Enter Version Number Here

Software Architecture Document

<Month and 4-digit year>

NOTE: *This template contains a paragraph style called Instructional Text. Text using this paragraph style is designed to assist the reader in completing the document. Text in paragraphs added after this help text is automatically set to the appropriate body text level. For best results and to maintain formatting consistency, use the provided paragraphs styles.*

Revision History

NOTE: *The revision history cycle begins once changes or enhancements are requested after the initial version of the Software Architecture Document has been completed.*

Date	Version	Description	Author
<mm/dd/yy>	<x.x>	<details>	<name>

Table of Contents

Software Architecture Document

1. Introduction

Provide an overview of the entire Software Architecture Document. Include the scope, definitions, acronyms, abbreviations, references, and overview of this document.

Define the role or purpose of the Software Architecture Document as it relates to the overall project documentation. Briefly describe the structure of the document.

1.1. Scope

A brief description of what this document applies to, what is affected by it, or influenced by it.

1.2. Definitions, Acronyms, and Abbreviations

Provide definitions for all terms, acronyms, and abbreviations required to properly interpret how they are used in this document. Provide a reference to the project's glossary document if necessary.

1.3. References

Provides a complete list of all referenced documents referenced in this document. Identify each document by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. Use a reference to an appendix in this document or another document if necessary.

1.4. Overview

Describe what the rest of this document contains and explain its organization.

2. Architectural Representation

Describe what software architecture the system implements. To illustrate the architecture, include the commonly used set of views known as the "4 + 1 architecture view model", (first proposed by Philippe Kruchten and depicted below in Figure 1 Software Architecture: The "4+1 View" Model).

The "4+1 architecture view model" illustrates five uniquely viewed perspectives into the design of the architecture: Use-Case, Logical, Process, Deployment, and Implementation. Each viewpoint describes:

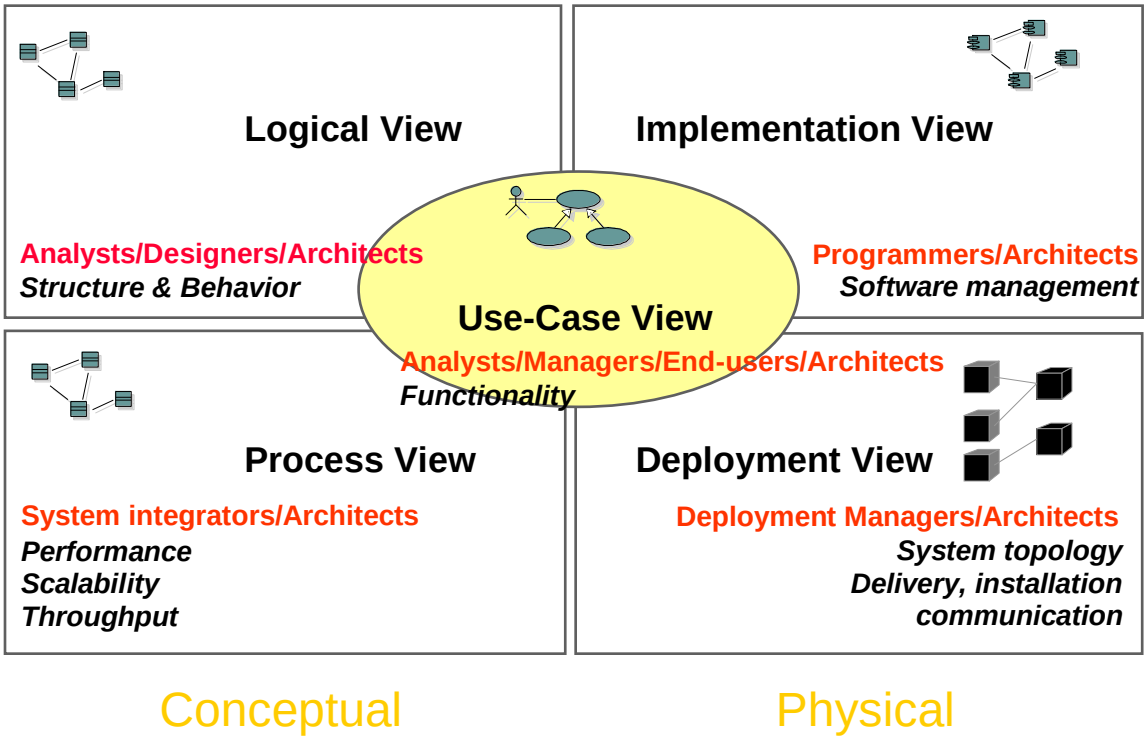
- *One or more system model(s) and view(s) of those models;*
- *Stakeholders interested in the view(s); and*
- *Stakeholder concerns in the view(s).*

Provide the following views:

- *The Use-Case View to describe functional and non-functional significant architectural requirements (SARs). Use cases constitute the glue that unifies all the other views. Model elements in this view include:*
  - *Actors*
  - *Use Cases*
  - *Classes*
  - *Collaborations*

- *The Logical View to describe key design mechanisms, architecturally important design elements, their interdependencies and the organization of these elements into subsystems and layers. Model elements in this view include:*
  - *Objects*
  - *Classes*
  - *Collaborations*
  - *Interactions*
  - *Packages*
  - *Subsystems*
- *The Implementation View to describe key implementation elements such as code artifacts, executables, and modules. Model elements in this view include:*
  - *Modules*
  - *Subsystems*
  - *Interfaces*
- *The Process View to describe processes and threads, and the allocation of the logical and/or the implementation elements to these processes and threads. Model elements in this view include:*
  - *Tasks, threads, processes*
  - *Interactions*
- *The Deployment View to describe system nodes such as computers or routers, and the allocation of the logical, implementation, or process elements to those nodes. Model elements in this view include:*
  - *Nodes*
  - *Modules*
- *In addition to the above views, a "Data View" should be included in the Architectural Representation whenever persistent data objects are included in the system (they usually are in most software systems). The data view describes the logical data model of the system and includes an Entity Relationship Diagram (ERD). For a description of Entity Relationship diagramming please refer to the whitepaper <http://www-106.ibm.com/developerworks/rational/library/content/03July/2500/2785/2785_uml.pdf>*

Figure 1 Software Architecture: The "4+1 View" Model[1]



3. Architectural Goals and Constraints

Describe the software requirements and objectives that have some significant impact on the architecture. Examples of goals and constraints include safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code.

4. Use-Case View

This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.

5. Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages, and for each significant package, its decomposition into classes and class utilities. Introduce architecturally significant classes and describe their responsibilities as well as a few important relationships, operations, and attributes.

5.1. Overview

Describe the overall decomposition of the design model in terms of its package hierarchy and layers.

[1] The Unified Modeling Language User Guide Copyright 1999 by Addison-Wesley, all rights reserved; Mastering Object Oriented Analysis and Design with UML Copyright @ 2003 Rational Software, all rights reserved.

5.2. Architecturally Significant Design Packages

For each significant package, include a subsection with its name, a brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, a brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.

6. Process View

Describe the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.

7. Deployment View

Describe one or more physical network (hardware) configurations on which the software runs (this is a view of the deployment model). Include with each configuration the physical nodes (computers, CPUs) that execute the software and their interconnections (bus, LAN, point-to-point). Also include a mapping of the processes of the Process View onto the physical nodes.

8. Implementation View

Describe the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.

8.1. Overview

Provide names and definitions for the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram showing the relationships between the layers.

8.2. Layers

For each layer, include a subsection with its name, a listing of the layer's subsystems, and a component diagram.

9. Data View

Provide a description of the system's persistent data storage perspective. This section is optional if there is little or no persistent data or the translation between the Design Model and the Data Model is trivial. Describe the architecturally significant aspects of logical data model (such as design patterns used), physical data model (such as normalized vs. de-normalized design approaches), Database views, choice of transaction strategy, distribution, concurrency, fault tolerance. Describe how the project made plans to ensure data quality, control redundancy, and appropriately use authoritative sources of data. Describe the mechanisms intended for data access by external systems.

10. Size and Performance

Provide a description of the major size and performance characteristics of the software that impact the architecture, as well as the target performance constraints.

11. Quality

A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and others. If these characteristics have special significance, such as safety, security or privacy implications, delineate them clearly.

11.1. Applicability to Technical Reference Model/Standards Profile (TRM/SP)

Show adherence to the VA TRM/SP. New system development and selection must adhere to approved standards and rules, unless it proves to be more cost-effective over the life of the application to deviate from the standards. The standards, strategies, and guidelines establish the fundamental technologies enabling the VA to meet many of its business and information system goals. By using these standards, the VA can promote interoperability, portability and adaptability within systems, promote quality assurance, place the VA in a position to utilize current technology, and provide a framework for IT application and infrastructure development. The current TRM/SP is located VA Enterprise Architecture (EA) v2.1 at http://vaww.va.gov/oit/eam/easervice/eav2-1collection/default.asp.

11.2. Applicability to Common Services Architecture

Describe how the application aligns with the defined Common Services Architecture. As specified by the VA EA and the HealtheVet Logical Model that provides a practical interpretation thereof, the application modernization effort utilizes n-tier architecture and is based on a common, shared services approach. This design maximizes standardization and re-use while minimizing maintenance impacts, technology, and database dependence.

Does the design ensure a separation between application logic, user interface, authorization support, and data storage? Is the system divided into areas of responsibility that are implemented as "service-based" components? Does the application delegate processing to external common services where such capabilities are available?

11.3. Applicability to HSD&D Core Specifications for Rehosting Initiatives

Show adherence to the HSD&D Core Specifications for Rehosting Initiatives. New systems development and selections must adhere to HSD&D approved architectural and integrations standards and policies, unless it proves to be more cost-effective over the life of the application to deviate from the standards. The standards, strategies, and guidelines established through this specification, those unique to HSD&D or which illustrate and/or serve to implement those specified elsewhere in the VA, provide a fundamental framework of expectations of all systems enabled by HSD&D. By establishing this expectation, HSD&D promotes established VA standards through preferred implementation specifics that allow HSD&D to provide a common approach or solution to comply with all architectural goals.

12. Architectural Mechanism

Describe how the system's design supports anticipated system load, growth, availability, concurrency, and distribution requirements (including server, workstation, storage, bandwidth, and middleware).

12.1. Analysis Mechanisms

Describe the following analysis mechanisms as they pertain to the project. Analysis mechanisms represent a pattern that constitutes a common solution to a common problem across classes. They can show patterns of structure, patterns of behavior, or both. They are used during analysis to reduce the complexity of analysis, and to improve its consistency by providing designers with a shorthand representation for complex behavior.

- *Persistency: A means to make an element persistent (i.e. exist after the application that created it ceases to exist).*
- *Distribution: A means to distribute an element across existing nodes of the system.*
- *Security: A means to control access to an element.*
- *Legacy Interface: A means to access a legacy system with an existing interface.*

12.2. Analysis-to-Design-to-Implementation Mechanisms Map

For each of the analysis mechanisms in Section 12.1, identify the corresponding design and implementation mechanisms. Design mechanisms should refine the analysis mechanisms based on the constraints imposed by the implementation environment. Design mechanisms provide an abstraction of the implementation mechanisms and bridge the gap between the analysis mechanisms and implementation mechanisms. The use of abstract architectural mechanisms during design allows consideration of the mechanisms without the details. Implementation mechanisms are used during the implementation process and should refine the design mechanisms given the implementation environment. Implementation mechanisms specify the exact implementation of the mechanism. They are bound to a specific technology, implementation language, vendor, etc. Some examples of implementation mechanisms include the actual programming language, COTS products, database, and the inter-process communication/distribution technology in use. The following table is an example of an Analysis-to-Design-to-Implementation Mechanisms Mapping table as applied to a sample project:

Analysis Mechanisms	Design Mechanisms	Implementation Mechanisms
Persistency	*OODMS (new data)*	*ObjectStore*
Persistency	*RDBMS (data from legacy database)*	*JDBC to Ingres*
Distribution	*Remote Method Invocation (RMI)*	*Java 1.1 from Sun*
Security		*Reverse Engineered Secure.java and UserContextRemoteObject components*
Legacy Interface		

Figure 2. Analysis-to-Design-to-Implementation Mechanisms Mapping

Analysis Mechanisms	Design Mechanisms	Implementation Mechanisms

12.3. Implementation Mechanisms

For each analysis mechanism listed in the table in Section 12.1, create a subsection that contains a Static View and Dynamic View of the design and implementation mechanisms.

12.3.1. <Mechanism subsection>

12.3.1.1. Static View

Class diagram appears here followed by class descriptions.

Class Descriptions

12.3.1.2. Dynamic View

Sequence diagram