System Architecture Document

version x.x

for

<Project Name>

prepared by

|  |  |
| --- | --- |
| name | email |
|  |  |

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction](#_a9fawcg1q6ya) 3

[1.1. Purpose](#_n1m5hqftv9nv) 3

[1.2. Scope](#_utsgrsemvlc6) 3

[1.3. Definitions, acronyms, and abbreviations](#_rsouffcqodif) 3

[2. Architectural representation](#_wky2od3py614) 4

[2.1. Logical view](#_u8qjvik9d6xs) 4

[2.1.1. Layers and tiers](#_fx9x81vxohxk) 4

[2.1.2. Subsystems](#_bt8inmwizkrk) 5

[2.1.3. Architecturally significant design packages](#_blhf7fesstky) 5

[2.1.4. Use case realizations](#_32zbql7rkz11) 5

[2.2. Development view](#_oqhz7tt4s4sp) 6

[2.2.1. Reuse of components and frameworks](#_30s9jynilrqj) 6

[2.3. Process view](#_aktj082719lg) 7

[2.4. Physical view](#_3gn2exl32d0v) 8

[2.5. Use case view](#_wvpzolgi6sox) 9

[2.6. Data view](#_swmvrojhxbvg) 10

[3. Architectural requirements: goals and constraints](#_16wprx93mt0j) 11

[3.1. Functional requirements](#_pt1vnv79o8e2) 11

[3.2. Non-functional requirements](#_ogop1ua902nk) 11

[4. Quality](#_gmydwbnai3mm) 13

# 1. Introduction

The introduction of the Software Architecture Document provides an overview of the entire document.

## 1.1. Purpose

This section defines the role or purpose of the Software Architecture Document and briefly describes the structure of the document. Identify the intended audience for the document is identified, with an indication of how they are expected to use the document.

## 1.2. Scope

A brief description of what the Software Architecture Document applies to; what is affected or influenced by this document.

## 1.3. Definitions, acronyms, and abbreviations

Provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software Architecture Document. This information may be provided by reference to the project’s Glossary. For example:

RUP: Rational Unified Process

UML: Unified Modeling Language

SAD: Software Architecture Document

# 2. Architectural representation

Describe the top-level architectural style of the system and the view model you will adopt. Additionally describe what each individual view will provide. Many enterprise software systems are modeled using the 4+1 view illustrated in Figure 1.

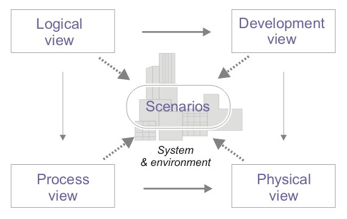


Figure 1: The 4+1 view model.

## 2.1. Logical view

Audience: Designers. The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include **Class diagram**, and **interaction diagrams** (**communication diagrams**, or **sequence diagrams**).

The logical view captures the functionality provided by the system; it illustrates the collaborations between system components in order to realize the system's use cases. Describe the architecturally significant logical structure of the system. Think of decomposition in tiers and subsystem. Also describe the way in which, in view of the decomposition, Use Cases are technically translated into Use Case Realizations.

### 2.1.1. Layers and tiers

Describe the top-level architecture style. Deploy a *UML class diagram*.

### 2.1.2. Subsystems

Describe the decomposition of the system in subsystems and show their relation.

### 2.1.3. Architecturally significant design packages

Desribe packages of individual subsystems that are architecturally significant. For each package nclude a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

### 2.1.4. Use case realizations

In this section you have to illustrate how use cases are translated into *UML interaction diagrams*. Give examples of the way in which the Use Case Specifications are technically translated into Use Case Realizations, for example, by providing a sequence-diagram. Explain how the tiers communicate and clarify how the components or objects used realize the functionality.

## 2.2. Development view

(also known as Implementaion view): Audience: Programmers. The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the **Package diagram**.

The development (or implementation) view describes the components used to assemble the system. Use a *UML component diagram* to capture this view.

### 2.2.1. Reuse of components and frameworks

Describe any third-party or home-made components and frameworks that will be reused.

## 2.3. Process view

Audience: Integrators. The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the Activity diagram.

The process view illustrates the system's processes, focusing on the runtime behavior of the system. The view illustrates parallelism and concurrency. Deploy a UML activity diagram to capture this view.

## 2.4. Physical view

(also known as deployment view) : Audience: Deployment managers. The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. UML Diagrams used to represent physical view include the **Deployment diagram**.

The deployment (or physical) view illustrates the physical components of the architecture, their connectors and their topology. Describe the physical network and hardware configurations on which the software will be deployed. This includes at least the various physical nodes (computers, CPUs), the interaction between (sub)systems and the connections between these nodes (bus, LAN, point-to-point, messaging, SOAP, http, http). Use a *UML deployment diagram* to capture this view.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Name of the node. | Node type. | Technical specifications. |

## 2.5. Use case view

(also known as Scenarios) : Audience: all the stakeholders of the system, including the end-users. The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. Related Artifacts : **Use-Case Model**.

## 2.6. Data view

(optional): Audience: Data specialists, Database administrators. Describes the architecturally significant persistent elements in the data model . Related Artifacts: Data model.

An enterprise software system would additonally require a data view. The data view describes the data entities and their relationships. Deploy an Entity-Relationship (ER) Model to represent this view. Note that the ER model is not part of the UML specification. Additionally you can deploy a UML class diagram to represent the data view where classes would correspond to data entities

# 3. Architectural requirements: goals and constraints

Requirements are already described in SRS. In this section describe *key* requirements and constraints that have a significant impact on the architecture.

## 3.1. Functional requirements

Refer to Use Cases or Use Case scenarios which are relevant with respect to the software architecture. The Use Cases referred to should contain central functionality, many architectural elements or specific delicate parts of the architecture.

The overview below refers to architecturally relevant Use Cases from the Use Case Model (see references).

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| Use case(s) or scenario(s). | Name of case(s) or scenario(s). | Description on why this use case or scenario is relevant to the architecture. | Section number where this use case or scenario is addressed in this document. |

## 3.2. Non-functional requirements

Describe the architecturally relevant non-functional requirements, i.e. those which are important for developing the software architecture. Think of security, privacy, third-party products, system dependencies, distribution and reuse. Also environmental factors such as context, design, implementation strategy, team composition, development tools, time to market, use of legacy code may be addressed.

Usually, the non-functional requirements are already in place and can be referenced here. This document is not meant to be the source of non-functional requirements, but to address them. Provide a reference per requirement, and where the requirement is addressed.

|  |  |  |  |
| --- | --- | --- | --- |
| Source | Name | Architectural relevance | Addressed in |
| e.g. Vision, SRS. | Name of requirement. | Description on why this requirement is relevant to the software architecture. | Section number where this requirement is addressed in this document. |

# 4. Quality

A description of how the software architecture contributes to the quality attributes of the system as described in the ISO-9126 (I) standard. **For example**: The following quality goals have been identified:

Scalability:

· Description : System’s reaction when user demands increase

· Solution : J2EE application servers support several workload management techniques

Reliability, Availability:

· Description : Transparent failover mechanism, mean-time-between-failure

· Solution : : J2EE application server supports load balancing through clusters

Portability:

· Description : Ability to be reused in another environment

· Solution : The system me be fully J2EE compliant and thus can be deploy onto any J2EE application server

Security:

· Description : Authentication and authorization mechanisms

· Solution : J2EE native security mechanisms will be reused