

**Architecture Program**

**American Funds®**

**<Project>**

*Project Subtitle*

Application Architecture Document

**Version: Draft 0.1**

Issue Date: Month Year

**<company>**

*© Capital Group Companies*

Document Change Log

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Updated By** | **Change Summary** |
| 0.1 | mm/dd/yyyy | ??? | Initial draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

TABLE of CONTENTS

Section Page

[1 Introduction 4](#_Toc228068116)

[2 Architecture Change Notes 4](#_Toc228068117)

[2.1 Additions 4](#_Toc228068118)

[2.2 Changes 4](#_Toc228068119)

[2.3 Deletions 4](#_Toc228068120)

[3 Architecture Overview 4](#_Toc228068121)

[3.1 Architectural Principles and Constraints 4](#_Toc228068122)

[3.2 Architecture Variables 4](#_Toc228068123)

[3.3 Conceptual Architecture Diagram 4](#_Toc228068124)

[3.4 Key Technologies 4](#_Toc228068125)

[3.5 Architectural Decisions 4](#_Toc228068126)

[4 Use-Case View 4](#_Toc228068127)

[4.1 Use-Case Patterns (Optional) 4](#_Toc228068128)

[4.2 Actors 4](#_Toc228068129)

[4.3 Architecturally Significant Uses Cases 4](#_Toc228068130)

[4.3.1 Criteria for Selecting Architecturally Significant Use-Cases 4](#_Toc228068131)

[4.3.2 Use-Case Diagram 4](#_Toc228068132)

[4.4 Rationale 4](#_Toc228068133)

[5 Analysis and Design View 4](#_Toc228068134)

[5.1 Software Patterns 4](#_Toc228068135)

[5.2 Layers 4](#_Toc228068136)

[5.3 Architecturally Significant Packages 4](#_Toc228068137)

[5.4 Domain Classes 4](#_Toc228068138)

[5.5 Use-Case Realizations 4](#_Toc228068139)

[5.5.1 <Specific Use-Case> 4](#_Toc228068140)

[5.6 Concurrency (Optional) 4](#_Toc228068141)

[5.6.1 Processes and Threads 4](#_Toc228068142)

[5.6.2 Synchronization Mechanisms 4](#_Toc228068143)

[5.7 Rationale 4](#_Toc228068144)

[6 Deployment View 4](#_Toc228068145)

[6.1 Deployment Patterns (Optional) 4](#_Toc228068146)

[6.2 Abstract Deployment Model 4](#_Toc228068147)

[6.3 Concrete Deployment Model 4](#_Toc228068148)

[6.4 Rationale 4](#_Toc228068149)

[7 Implementation View (Optional) 4](#_Toc228068150)

[7.1 Development Directory Structure 4](#_Toc228068151)

[7.1.1 <project-root> directory 4](#_Toc228068152)

[7.1.2 bin subdirectory 4](#_Toc228068153)

[7.1.3 doc subdirectory 4](#_Toc228068154)

[7.1.4 etc subdirectory 4](#_Toc228068155)

[7.1.5 lib subdirectory 4](#_Toc228068156)

[7.1.6 java subdirectory 4](#_Toc228068157)

[7.1.7 java/build subdirectory 4](#_Toc228068158)

[7.1.8 csharp subdirectory 4](#_Toc228068159)

[7.2 Development Standards (Optional) 4](#_Toc228068160)

[7.2.1 Class Naming Conventions 4](#_Toc228068161)

[7.2.2 Code Formatting Rules 4](#_Toc228068162)

[7.2.3 Testing Guidelines 4](#_Toc228068163)

[7.3 Rationale 4](#_Toc228068164)

[8 Data View 4](#_Toc228068165)

[8.1 Data Patterns (Optional) 4](#_Toc228068166)

[8.2 Database Environment 4](#_Toc228068167)

[8.3 Rationale 4](#_Toc228068168)

[9 Architectural Consequences 4](#_Toc228068169)

[9.1 Benefits 4](#_Toc228068170)

[9.2 Liabilities 4](#_Toc228068171)

[9.3 Trade-offs 4](#_Toc228068172)

[9.4 Addressing Each Perspective 4](#_Toc228068173)

[9.4.1 Availability 4](#_Toc228068174)

[9.4.2 Performance 4](#_Toc228068175)

[9.4.3 Usability 4](#_Toc228068176)

[9.4.4 Security 4](#_Toc228068177)

[9.4.5 Evolution 4](#_Toc228068178)

[9.4.6 Internationalization (optional) 4](#_Toc228068179)

[9.4.7 Regulation (optional) 4](#_Toc228068180)

[10 Appendix 4](#_Toc228068181)

[10.1 Definitions, Acronyms, and Abbreviations 4](#_Toc228068182)

[10.2 Architectural Representation 4](#_Toc228068183)

[10.2.1 Views 4](#_Toc228068184)

[10.2.2 Perspectives 4](#_Toc228068185)

[11 References 4](#_Toc228068186)

[11.1 Project Artifacts 4](#_Toc228068187)

[11.2 Software Patterns 4](#_Toc228068188)

[11.3 Methodology 4](#_Toc228068189)

# Introduction

[Help Text contains tips and guidance for developing each section of the Application Architecture Document

The Help Text style can be hidden by:

Selecting Format 🡪 Styles and Formatting… and choosing the Help Text style

Selecting Modify…

Selecting Format 🡪 Font…

Check the Hidden Effects checkbox and click OK]

[The introduction of the Project Development Plan provides an overview of the entire plan. It can optionally include descriptions of the purpose, audience, and overview of the Project Development Plan.]

This *Application Architecture Document* provides a comprehensive architectural overview of the <Project> system, using a number of different architectural views to depict different aspects of the system. Its purpose is to:

* Provide both business stakeholders and technical stakeholders a framework around which design and implementation decisions can be coordinated and business impact analyzed.
* Provide a common technical vision to the development team to facilitate a cohesive and consistent implementation.
* Communicate technical decisions to architecture and technology review boards in order to conform with enterprise architectural governance standards.

It is intended to capture and convey the significant architectural decisions which have been made on the system.

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

This *Application Architecture Document* is a central artifact employed in the development of the <Project> system. The views presented in this document are distilled from information extracted from the <Project>’s [*Application Model Set*](#Application_Model_Set). Other information used in this document is derived from the <Project>’s [*Use-Case Model Survey*](#Use_Case_Model_Survey) .

The audience for this *Application Architecture Document* is composed of various stakeholders that have an interest in the development of this system. These stakeholders include:

* **Customers and Sponsors –** Customers and Sponsors will use this document to verify that the functionality and quality properties required by them is being implemented in the *<Project>* project.
* **Project Managers** – Project Managers use this document to manage the *<Project>* project. This primarily includes:
  + defining the scope of the project,
  + ensuring that important requirements are addressed in the application architecture, and
  + partitioning the work of the project into iterations.
* **Reviewers** – Reviewers use this document as a representation of the *<Project>* system in order to evaluate the system for completeness, quality, consistency, and conformance to enterprise standards.
* **Architects** – Architects use this document to provide a common technical vision to the development team in order to facilitate a cohesive and consistent implementation.
* **Development Team** – Team Members use this document to understand the architectural direction of the project and as a guide for their work in executing project design, implementation, and maintenance tasks.
* **System Administrators** – System Administrators responsible for the orderly operation of the <Project> system use this document to verify that the application and environment provides the necessary controls and capabilities to effectively monitor and operate the system.

The *Application Architecture* passes through three milestones. Each milestone represents a phase in the *Application Architecture’s* development; with each successive phase introducing more details into the *Application Architecture* and greater validation of architectural decisions. The three milestones are:

1. **Candidate Architecture Milestone** – represents the completion of the initial phase of an *Application Architecture*. At this milestone the *Application Architecture* must have just enough detail (typically in the [Use-Case](#4.__________________Use-Case_View) and [Logical](#5.__________________Logical_View) Views) to enable *System Stakeholders* to understand the scope of the system and to create the initial iteration plans.
2. **Baseline Architecture Milestone** – represents the validation and stabilization of the application’s architecture through prototyping and early development. At this milestone the *Application Architecture* has more details and those details have been proven through partial implementation and testing. The *Development Team* and the *System Stakeholders* agree on the Baseline Architecture and it becomes the stable model on which the remaining system development will be based.
3. **Final Architecture Milestone** – represents completion of the construction phase of the system and any minor changes or additions that have been included in the *Application Architecture* to accommodate the pragmatics of system development. At this milestone the *System Architecture* is complete for the current release of the system. The Final Architecture becomes the starting point for new releases of the system and is used in architecture assessments and planning.

**Note:** This document is currently at the <Architecture Milestone> **Architecture Milestone.**

[This subsection describes what the rest of the Project Development Plan contains and explains how the Project Development Plan is organized.]

This document is organized into 10 major sections:

* [Introduction](#_Introduction) – The current section that describes the purpose, audience, and organization of the architecture plan for the *<Project>* project.
* [Architecture Change Notes](#_Architecture_Change_Notes) – Lists the additions, changes, and deletions in each version and phase of the *<Project>*‘s architecture.
* [Architecture Overview](#_Architecture_Overview) – Describes the principles and constraints that apply to the *<Project>* project, and describes the high level characteristics and major decisions of the *<Project>*’s architecture.
* [Use-Case View](#4.__________________Use-Case_View) – which contains use cases and scenarios that encompasses architecturally significant behavior, classes, or technical risks.
* [Analysis and Design View](#_Analysis_and_Design) – which contains the most important analysis and design classes and their organization into subsystems, and the organization of these subsystems into layers. Optionally contains the description of the system’s tasks (process and threads), their interactions and synchronization, and the allocation of design classes and objects to tasks.
* [Deployment View](#7.__________________Deployment_View) **–** which contains the description of the various physical nodes for the most typical platform configurations, and the allocation of tasks (from the Process View) to the physical nodes. This view need only be used if the system is distributed.
* [Implementation View](#8.__________________Implementation_View) - which contains an overview of the implementation model and its organization in terms of modules into packages and layers.
* [Data View](#9.__________________Data_View_(optional)) - which contains a description of the persistent data storage aspect of the system. This view is optional if there is little or no persistent data, or the translation between the design model and the data model is trivial.
* [Architectural Consequences](#_Architectural_Consequences) – Presents the benefits, liabilities, and trade-offs associated with the decisions embodied in this architecture. This section also discusses how the architecture addresses each of the required quality properties.

[Appendices](#_Appendices) and [References](#_References) are located at the end of the document.

# Architecture Change Notes

The current version and milestone is: <Architecture Version> <Architecture Milestone>.

## Additions

|  |  |  |
| --- | --- | --- |
| **Version and Phase of AAD** | **Location of Addition in AAD** | **Description of Addition** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Changes

|  |  |  |
| --- | --- | --- |
| **Version and Phase of AAD** | **Location of Change in AAD** | **Description of Change** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Deletions

|  |  |  |
| --- | --- | --- |
| **Version and Phase of AAD** | **Location of Deletion in AAD** | **Description of Deletion** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Architecture Overview

## Architectural Principles and Constraints

[This section describes the software requirements and objectives that have some significant impact on the architecture, for example, 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, and so on.]

[Examples]

The major design and implementation constraints for the system are:

* Simplicity – Do the simplest thing that could possibly work. It is better to start with simple designs and evolve them to handle progressively complex functionality.
* Testability – Classes and object collaborations must be designed in such a way that they can be automatically tested in both isolation (unit testing) and in combination (functional testing).
* Reusability – The system should maximize the use of reusable artifacts where ever possible. This includes reusing the following kinds of artifacts:
  + Reference Architecture
  + Use-Cases
  + Business Rules
  + Design Patterns
  + Services
  + Object-Oriented Frameworks
  + Object-Oriented Toolkits
* Resiliency – Systems that have *evolutionary architecture* are able to rapidly evolve to meet customer requirements, adapt to changing external market conditions, respond to internal business conditions, and transcend implementation technology.

An *evolutionary architecture*:

* + Is organized by one or more architectural patterns
  + Contains a high density of design patterns [[Gamma+1995]](#Gamma_et_al_1995)
  + Encapsulates implementation-specific technology
  + Has a testable design
  + Contains a large suite of unit tests and functional tests
  + Exists in an environment of continuous refactoring [[Fowler1999]](#Fowler_1999)
  + Is both a structure and a process

Systems with evolutionary architecture remain effective far longer than other systems because they can adapt to change more easily than other systems.

* Consistency – Designs and code that follow common conventions and standards are easier to communicate because they do not require “translation” from each team member’s personal conventions and they promote consistency at all levels of the architecture.

## Architecture Variables

[This subsection describes the overall architecture type of the system in terms of architecture variables that represents various characteristics of the system architecture.]

|  |  |
| --- | --- |
| **Architecture Variable Name** | **Value** |
| Number of Logical Layers | n |
| Number of Deployment Tiers | n |
| Application Data Access | Sequential | Shared/Synchronized |
| Application User Interaction | Isolated | Interacting |
| Application Client/Presentation Types | List each of the client or presentation types used in the application from the following:   * Thick Client * Thin Web Client * Rich Web Client * Batch Client * Service Client |
|  |  |
|  |  |

## Conceptual Architecture Diagram

[This optional subsection depicts the overall system structure in an informal sketch.]

|  |  |
| --- | --- |
| **Component** | **Description** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Key Technologies

|  |  |  |
| --- | --- | --- |
| **Technology** | **Description** | **Rationale** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Architectural Decisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Decision** | **Description** | **Rationale** | **View** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Behavioral 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.]

All behavior within the system is captured by use-cases. The users of the system and other systems interacting with the system are modeled as actors. Together the use-cases and actors form a use-case model. This model describes all the ways the system can be used by its environment [Overgaard+2004].

Use-Case Patterns (Optional)

[List and briefly describe the use-case patterns used to structure the Use-Case Model.]

The <Project> system architecture uses the following use-case patterns to organize its Use-Case Model:

* <pattern>

## Actors

<significant actors diagram>

|  |  |
| --- | --- |
| **Application Actor** | **Description** |
|  |  |
|  |  |
|  |  |

## Architecturally Significant Uses Cases

### Criteria for Selecting Architecturally Significant Use-Cases

### Use-Case Diagram

<use-case diagram>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Application Use-Case** | **Description** | **Business Value** | **Technical Risk** | **Priority** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Rationale

Rationale for the structure of the use-case model. Reference any architectural decision in section 2.5.

# Analysis and Design View

## Software Patterns

[List and briefly describe the software patterns used to structure the Analysis and Design Models.]

The <Project> system architecture uses various kinds of software patterns (architectural patterns, design patterns, and concurrency patterns) to structure the classes and objects of its packages. The most prominent software patterns include:

* <pattern>

[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. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

## Layers

[This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.]

The <Project> system is organized into a typical layered architecture [[Evans2000]](#Evans_2004)[[Buschman+1996]](#Buschmann_et_al_1996) with 4 layers. These layers are:

* Interaction –
* Application –
* Domain –
* Infrastructure –

## Architecturally Significant Packages

[For each significant package, include a subsection with its name, its 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, brief description, and, optionally a description of some of its major responsibilities, operations and attributes.]

<subsystems diagram>

## Domain Classes

[For each significant package, include a subsection with its name, its 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, brief description, and, optionally a description of some of its major responsibilities, operations and attributes.]

<analysis class diagram>

|  |  |
| --- | --- |
| **Domain Class** | **Description** |
|  |  |
|  |  |
|  |  |

## Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality.

### <Specific Use-Case>

Analysis Classes

<analysis class diagram>

Basic Flow (Analysis)

<analysis sequence diagram>

Design Classes

<design class diagram>

Basic Flow (Design)

<design sequence diagram>

## Concurrency (Optional)

### Processes and Threads

<object diagrams>

|  |  |
| --- | --- |
| **Process Type** | **Description/Purpose** |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Thread Type** | **Description/Purpose** |
|  |  |
|  |  |
|  |  |

### Synchronization Mechanisms

<object diagrams>

|  |  |
| --- | --- |
| **Synchronization Mechanism** | **Description** |
|  |  |
|  |  |
|  |  |

## Rationale

Rationale for the structure of the analysis and design models. Reference any architectural decision in section 2.5.

# Deployment View

[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. It is a view of the Deployment Model. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software, and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the Process View onto the physical nodes.]

## Deployment Patterns (Optional)

[List and briefly describe the deployment patterns used to structure the Deployment Model.]

The <Project> system architecture uses the following deployment patterns to organize its Deployment Model:

* <pattern>

## Abstract Deployment Model

[Classify the nodes and components in the deployment environment]

<deployment diagrams>

## Concrete Deployment Model

[Describe the physical instantiation of the Abstract Deployment Model, including actual server machines, network connections, components (application servers, database servers, etc).]

<deployment diagrams>

## Rationale

Rationale for the structure of the deployment models. Reference any architectural decision in section 2.5.

# Development View (Optional)

[This section describes the overall structure of the implementation model, the decomposition of the software into directories and files that represent subsystem and artifacts in the implementation model, and any architecturally significant implementation elements.]

## Development Directory Structure

[Describes the directory structure of the development environment showing the various kinds of files and artifacts present in the system’s development environment. A mapping from the directory structure to the logical package structure is also indicated.]

<actual project directory structure>

### <project-root> directory

The root directory of the <Project> system development environment.

### bin subdirectory

The directory that contains executable programs and scripts. All programs, utilities, and scripts should be installed into this directory.

### doc subdirectory

The directory that contains documentation artifacts that are closely related to code artifacts. These artifacts include:

* requirements
* business rules
* use-cases
* UML models
* Javadoc
* User manuals and tutorials

### etc subdirectory

The directory that contains miscellaneous files required for the operation and testing of the system. These files include:

* configurations
* properties
* test data

### lib subdirectory

The directory that contains binary deployment artifacts such as:

* Java archives
* Web archives
* Enterprise archives
* Dynamic link libraries

### java subdirectory

The directory that contains the system’s java source code. Source code is categorized as subsystem code and test code.

#### java/src/subsystems subdirectory

The root directory of the system’s java source code. Subdirectories implement java packages that contain subsystem classes. Subsystem class implement the functionality of the application.

#### java/src/tests subdirectory

The root directory of the system’s JUnit test code. Subdirectories mirror the structure of the java packages found in the src/subsystems directory. Test classes implement unit testing and functional testing of the application’s subsystems.

### java/build subdirectory

This directory is automatically generated by the project’s build scripts. The subdirectories correspond to specific build targets defined in the project’s build scripts. **Do not put any files under this directory or any subdirectories because they will be deleted during build environment cleaning.**

### csharp subdirectory

The directory that contains the system’s C# source code. Source code is categorized as subsystem code and test code. This directory will contain subdirectories similar to <project-root>/java.

## Development Standards (Optional)

### Class Naming Conventions

<describe class naming conventions>

### Code Formatting Rules

<describe code formatting rules>

### Testing Guidelines

<describe testing guidelines>

## Rationale

Rationale for the structure of the implementation model. Reference any architectural decision in section 2.5.

# Data View

[A description of the persistent data storage perspective of the system.]

## Data Patterns (Optional)

[List and briefly describe the data patterns used to structure the Data Model.]

The <Project> system architecture uses the following data patterns to organize its Data Model:

* <pattern>

## Database Environment

<database schema diagrams>

## Rationale

Rationale for the structure of the data model. Reference any architectural decision in section 2.5.

# Architectural Consequences

[A description of how the Application Architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, for example safety, security or privacy implications, they should be clearly delineated.]

## Benefits

## Liabilities

## Trade-offs

## Addressing Each Perspective

### Availability

The ability of the system to be fully or partly operational when required and to effectively handle and recover from failures that affect system operation. This includes disaster recovery requirements.

[Describe how the Application Architecture addresses the Availability perspective.]

<describe>

### Performance

The ability of the system to predictably execute within its mandated performance profile (SLAs) and to handle increased processing volumes.

[Describe how the Application Architecture addresses the Performance perspective.]

<describe>

### Usability

The ease with which people who interact with the system can work effectively.

[Describe how the Application Architecture addresses the Usability perspective.]

<describe>

### Security

The ability of the system to reliably control, monitor, and audit who can perform what actions on what resources and to detect and recover from failures in security mechanisms.

[Describe how the Application Architecture addresses the Security perspective.]

<describe>

### Evolution

The ability of the system architecture and design to be flexible in the face of inevitable change balanced against the costs of providing such flexibility.

[Describe how the Application Architecture addresses the Evolution perspective.]

<describe>

### Internationalization (optional)

The ability of the system to be independent from any particular language, country, or cultural group.

[Describe how the Application Architecture addresses the Internationalization perspective.]

<describe>

### Regulation (optional)

The ability of the system to conform to local and international laws, regulations, company policies, and other rules and standards.

[Describe how the Application Architecture addresses the Regulation perspective.]

<describe>

# Appendix

## Definitions, Acronyms, and Abbreviations

[This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the Application Architecture Document.  This information may be provided by reference to the project Glossary.]

All definitions, acronyms, and abbreviations can be found in the <project> [Glossary](#Project_Glossary).

## Architectural Representation

[This section describes what Application Architecture is for the current system, and how it is represented. Of the Use-Case, Logical, Process, Deployment, Implementation View, and Data View, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]

### Views

[This section is mostly boilerplate and can be left as is.]

The architecture for the <Project> system is represented in multiple **architectural views**. Each architectural view addresses some specific set of concerns, specific to stakeholders in the development process: end users, designers, managers, system engineers, maintainers, and so on. The views capture the major structural design decisions which must be tied to the requirements (functional and supplementary) and other constraints. These choices put further constraints on the requirements and on future design decisions at a lower level. These views are organized according to a 5+1 Architecture View Model (which is an extension of the 4+1 Architecture View Model [[Krutchen2004]](#Kruchten_2004)) and use UML [[Booch+2005]](#Booch_et_al_2005)[[Rumbaugh+2005]](#Rumbaugh_et_al_2005) to represent the models contained in each view.



The views presented in this document are:

* The[Use-Case View](#4.__________________Use-Case_View) - which contains use cases and scenarios that encompasses architecturally significant behavior, classes, or technical risks. It is a subset of the **use-case model**.
* The[Logical View](#5.__________________Logical_View) - which contains the most important design classes and their organization into **subsystems**, and the organization of these subsystems into **layers**. It also contains some use-case realizations. It is a subset of the **design model**.
* The [Deployment View](#7.__________________Deployment_View) **-** , which contains the description of the various physical nodes for the most typical platform configurations, and the allocation of tasks (from the Process View) to the physical nodes. This view need only be used if the system is distributed. It is a subset of the **deployment model**.
* The [Implementation View](#8.__________________Implementation_View) - which contains an overview of the **implementation model** and its organization in terms of modules into packages and layers. The allocation of packages and classes (from the Logical View) to the packages and modules of the Implementation View is also described. It is a subset of the implementation model.
* The [Process View](#6.__________________Process_View) **-** , which contains the description of the tasks (process and threads) involved, their interactions and configurations, and the allocation of design objects and classes to tasks. This view need only be used if the system has a significant degree of concurrency. It is a subset of the **design model**.
* The [Data View](#9.__________________Data_View_(optional)) - , which contains a description of the persistent data storage perspective of the system. This view is optional if there is little or no persistent data, or the translation between the design model and the data model is trivial.

These views, in essence, are extracts illustrating the "architecturally significant" elements of the models.

[Optionally list the views that will be used in this **Application Architecture Document**.]

### Perspectives

Perspectives are a set of related quality properties that require consideration across a number of architectural views. The perspectives that are addressed in this document are:

* **Availability** – The ability of the system to be fully or partly operational when required and to effectively handle and recover from failures that affect system operation. This includes disaster recovery requirements.
* **Performance** – The ability of the system to predictably execute within its mandated performance profile (SLAs) and to handle increased processing volumes.
* **Usability** – The ease with which people who interact with the system can work effectively.
* **Security** – The ability of the system to reliably control, monitor, and audit who can perform what actions on what resources and to detect and recover from failures in security mechanisms.
* **Evolution** – The ability of the system architecture and design to be flexible in the face of inevitable change balanced against the costs of providing such flexibility.
* **Internationalization** – (optional) The ability of the system to be independent from any particular language, country, or cultural group.
* **Regulation** – (optional) The ability of the system to conform to local and international laws, regulations, company policies, and other rules and standards.

[Optionally list the perspectives that will be used in this **Application Architecture Document**.]

# References

[This subsection should provide a complete list of all documents referenced elsewhere in the Application Architecture Document. Each document should be identified by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

## Project Artifacts

[Example Project Artifact References]

*<Project> Proposal/Business Case*. Version ?. Links:

<URL>

*<Project> Application Model Set*. Version ?. Links:

<URL>

*<Project> Use-Case Model Survey*. Version ?. Links:

<URL>

*<Project> Glossary*. Version ?. Links:

<URL>

## Software Patterns

[Example Software Pattern References]

[Alur+2003] D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns: Best Practices and Design Strategies, Second Edition.* Paolo Alto, CA: Sun Microsystems Press 1998. Links:

<http://www.corej2eepatterns.com>

<http://java.sun.com/blueprints/corej2eepatterns/Patterns>

<http://www.amazon.com/exec/obidos/tg/detail/-/0131422464>

[Gamma+1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns-Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995. Links:

<http://en.wikipedia.org/wiki/Design_Patterns>

<http://www.amazon.com/exec/obidos/tg/detail/-/0201633612>

[Overgaard+2004] Gunnar Overgaard, Karin Palmkvist. *Use Cases: Patterns and Blueprints*. Indianapolois, IN: Addison-Wesley, 2004. Links:

<http://www.amazon.com/exec/obidos/tg/detail/-/0131451340>

## Methodology

[Example Methodology References]

[Booch+2005] G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide, Second Edition*. Boston, MA: Addision-Wesley, 2005. Links:

<http://www.uml.org>

<http://www.amazon.com/exec/obidos/tg/detail/-/0201571684>

[Fowler1999] Martin Fowler. *Refactoring: Improving the Design of Existing Code*, pp. 295-299. Reading, MA: Addison-Wesley, 1999. Links:

<http://www.refactoring.com>

<http://www.amazon.com/exec/obidos/tg/detail/-/0201485672>

[Kruchten2004] Phillipe Kruchten. *The Rational Unified Process: An Introduction, Third Edition*, pp. 85-90. Boston, MA: Addision-Wesley, 2004. Links:

<http://www.amazon.com/exec/obidos/tg/detail/-/0321197704>

[Rumbaugh+2005] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Boston, MA: Addision-Wesley, 2005. Links:

<http://www.uml.org>

<http://www.amazon.com/exec/obidos/tg/detail/-/0321245628>