★Example: Architecture

URL : https://www.visual-paradigm.com/guide/archimate/full-archimate-viewpoints-guide/

- ☑**#example# 4+1 Views of Software Architecture**
- ☑**#example# Architectural Mechanism Attributes**

.

.

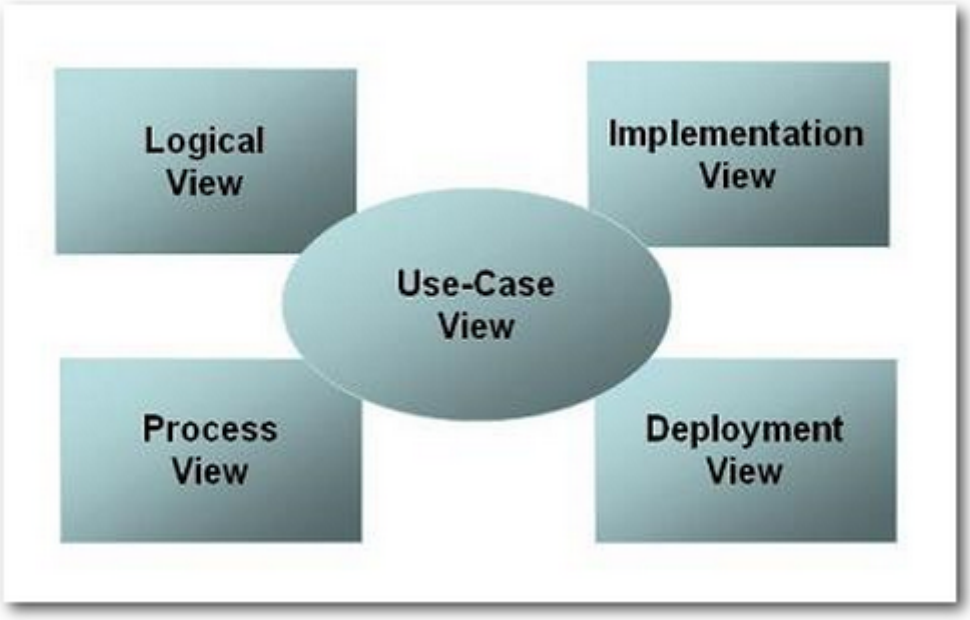
#example# 4+1 Views of Software Architecture

Summary

This example describes a possible set of views for describing a software architecture.

Main Description

You may want to consider the following views (not all views are relevant to all systems or all the stakeholders). This set of views is known as the 4+1 Views of Software Architecture [KRU95].



Use-case view (@@Use Case Diagram, Product Viewpoint):

Describes functionality of the system, its external interfaces, and its principal users. This view is mandatory when using the 4+1 Views, because all elements of the architecture should be derived from requirements.

Logical view (@@Class diagram, Application Cooperation Viewpoint): Describes how the system is structured in terms of (@@logical) units of implementation. The elements are **packages**, **classes**, and **interfaces**. The relationship between elements shows dependencies, interface realizations, part-whole relationships, and so forth. Note: This view is mandatory when using the 4+1 Views of Software Architecture.

Implementation view (@@Deployment diagram, Implementation and Deployment Viewpoint): Describes how development artifacts are organized in the file system. The elements are **files** and **directories** (any configuration items). This includes development artifacts and deployment artifacts. This view is optional when using the 4+1 Views.

Process view (@@Sequence diagram): Describes how the run-time system is structured as a set of elements that have run-time behavior and interactions. Run-time structure often bears little resemblance to the code structure. It consists of rapidly changing networks of communication objects. The elements are components that have run-time presence (**processes**, **threads**, **Enterprise JavaBeans™ (EJB™)**, **servlets**, **DLLs**, and so on), **data stores**, and **complex connectors**, such as queues. Interaction between elements varies, based on technology. This view is useful for thinking about run-time system quality attributes, such as performance and reliability. This view is optional when using the 4+1 Views.

Deployment view (@@Deployment diagram, Technology Usage Viewpoint): Describe how the system is mapped to the **hardware**. This view is optional when using the 4+1 Views.

In addition, you may wish to represent the following,

Data view: A specialization of the logical view. Use this view if persistence is a significant aspect of the system, and the translation from the design model to the data model is not done automatically by the persistence mechanism.

.

.

#example# Architectural Mechanism Attributes

Summary

This example illustrates how to represent attributes for Architecture Mechanisms.

Main Description

The following shows an example of how to capture information for Architectural Mechanism. The attributes of two possible mechanisms are shown: Persistence and Communication.

- **Persistence**

  For all classes with instances that may become persistent, you need to identify:

  - **Granularity:** What is the range of size of the objects to keep persistent?
  - **Volume:** How many objects (number) do you need to keep persistent?
  - **Duration:** How long does the object typically need to be kept?
  - **Retrieval mechanism:** How is a given object uniquely identified and retrieved?
  - **Update frequency:** Are the objects more or less constant? Are they permanently updated?
  - **Reliability:** Do the objects need to survive a crash of the process, the processor, or the whole system?

- **Communication**

  For all model elements that need to communicate with components or services that are running in other processes or threads, you need to identify:

  - **Latency:** How fast must processes communicate with another?
  - **Synchronicity:** Asynchronous communication
  - **Size of message:** A spectrum might be more appropriate than a single number
  - **Protocol:** Flow control, buffering, and so on

Notice that there is no design-level information or specification here. Instead, this is more about collating and refining architecturally significant requirements.

.

.