

# ***Software Architecture***

## ***View Model***

*BSc*



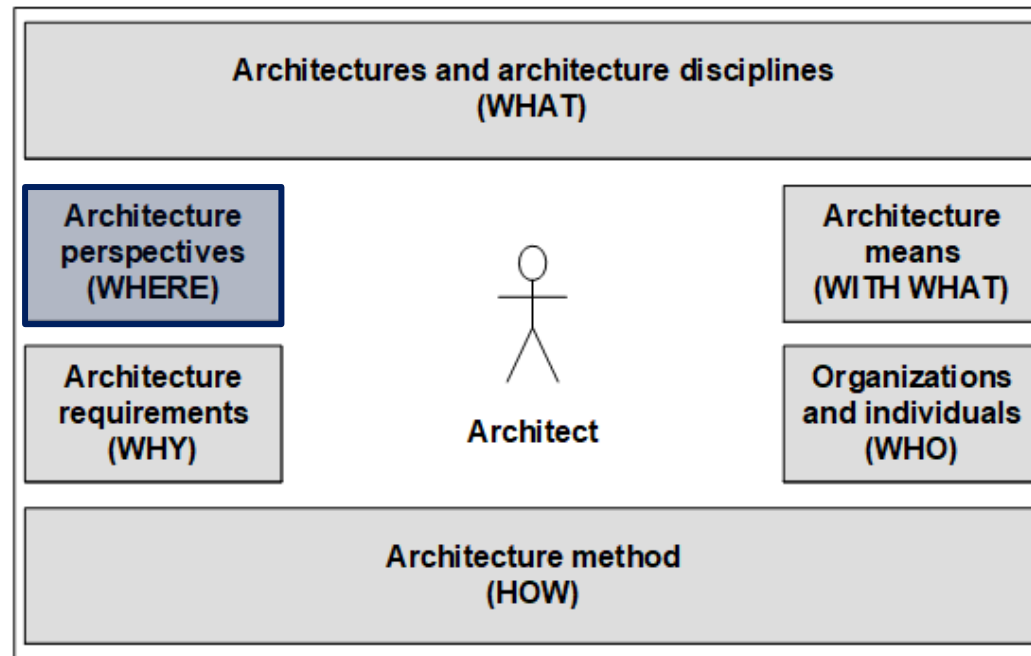
*Ingo Arnold*

# Lecture Opening

## Architecture Orientation Framework



Architecture WHERE deals with the **representation of architecture** in general as well as software architecture in particular [Vogel, Arnold et al 2011].



# Lecture Opening

## Motivation

In this unit, we will introduce **the concepts of architecture representation** including modelling, views, viewpoints, and view models. To do this, we will look at standard definitions and derive the core aspects of software architecture for us from these.

Beyond a general overview over view models per se, we will introduce and investigate the concrete view model of Philippe Kruchten [Kruchten 1995] .

# Lecture Opening

## Learning Objectives

You ...

- know essential building blocks as well as the benefits of architecture view models
- understand the relationship between architecture descriptions and view models
- know that different view models exist for different insight purposes
- know Kruchten's **4+1 view model** and understand how to apply it in the context of your own architecture descriptions

# Lecture Agenda

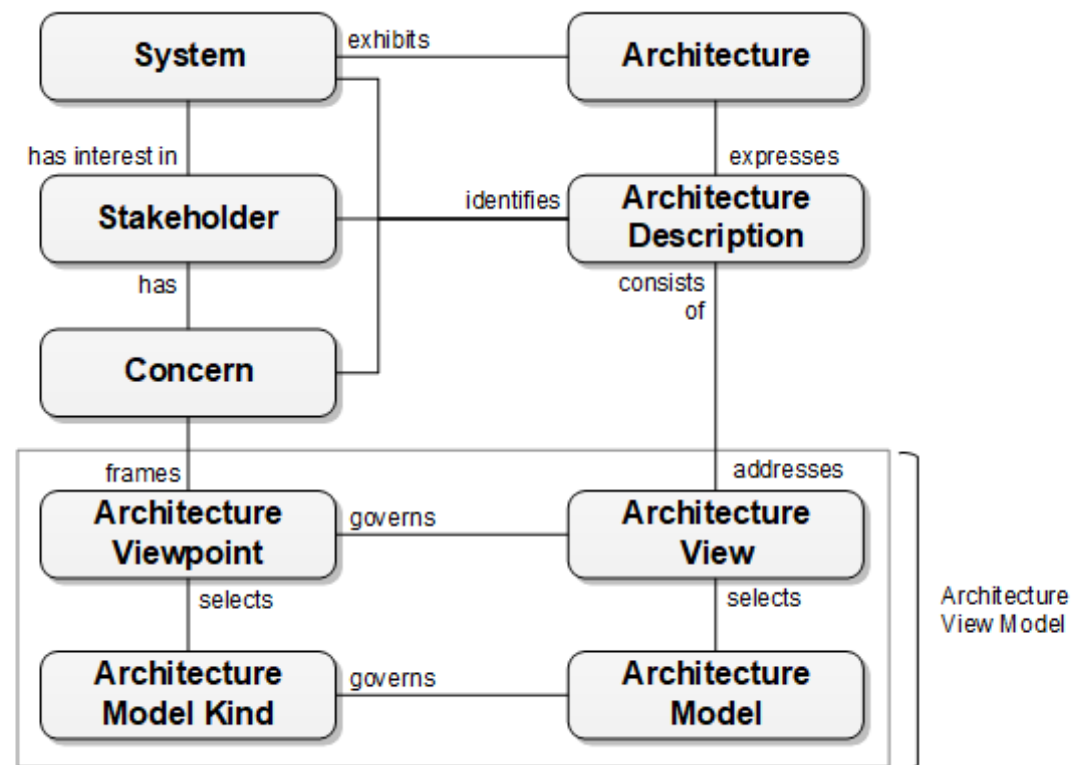


- Architecture View Model
- Architecture Description
- Kruchten 4+1 View Model

# Architecture View Model

## Architecture meta model

An architecture view model standardizes architecture manifestation. A view model explains the *why*, *what*, *who*, *when*, and *how* of architecture manifestation, each related to a specific area of an overall architecture description.



# Architecture View Model

## Viewpoint, view, and view model

A **view model** is a systematics composed of **viewpoints**, **types of models**, and relationships between them.

A **viewpoint** is a definition of the perspective from which a view is taken. It is a specification for constructing and using a view.

A **view** is what you see, while a viewpoint specifies from where you look. Viewpoints are templates for the concrete views that are instantiated from them.

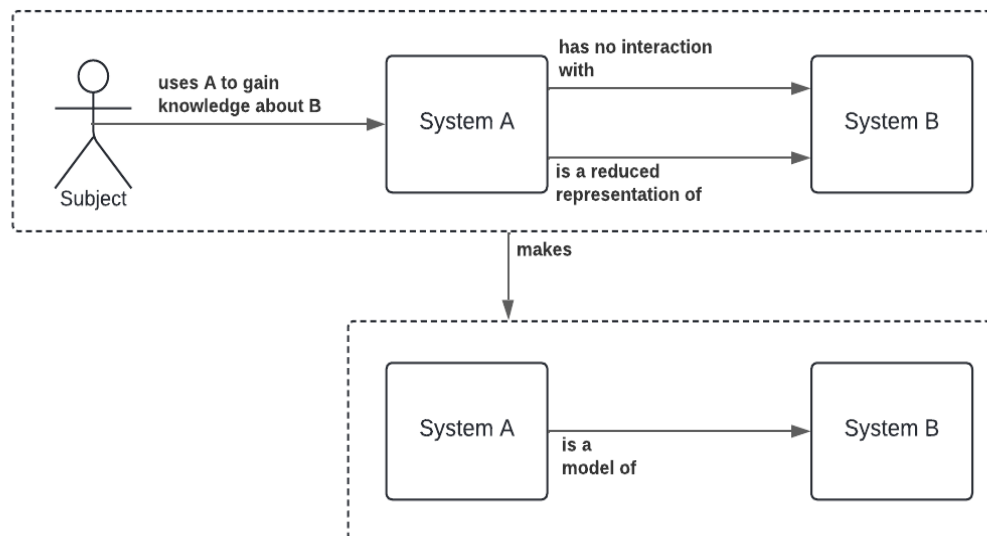
A view model clarifies for each viewpoint whose interests it serves, who is responsible for creating a view, how views are instantiated and evolved, how views are navigated to arrive at holistic insights, what model types are used to constitute views, or when views can be considered complete.

# Architecture View Model Modelling

**Modelling** is probably one of the activities most people primarily associate with the profession of architecture.

Leo Apostel [Apostel 1960], in his definition of the relationship of system to model, makes it clear that nothing is a model per se. Something becomes a model when it is used to represent something else:

“Any subject using a system A that is neither directly nor indirectly interacting with system B, to obtain information about the system B, is using A as a model for B.”





# Architecture View Model Modelling

## A model ...

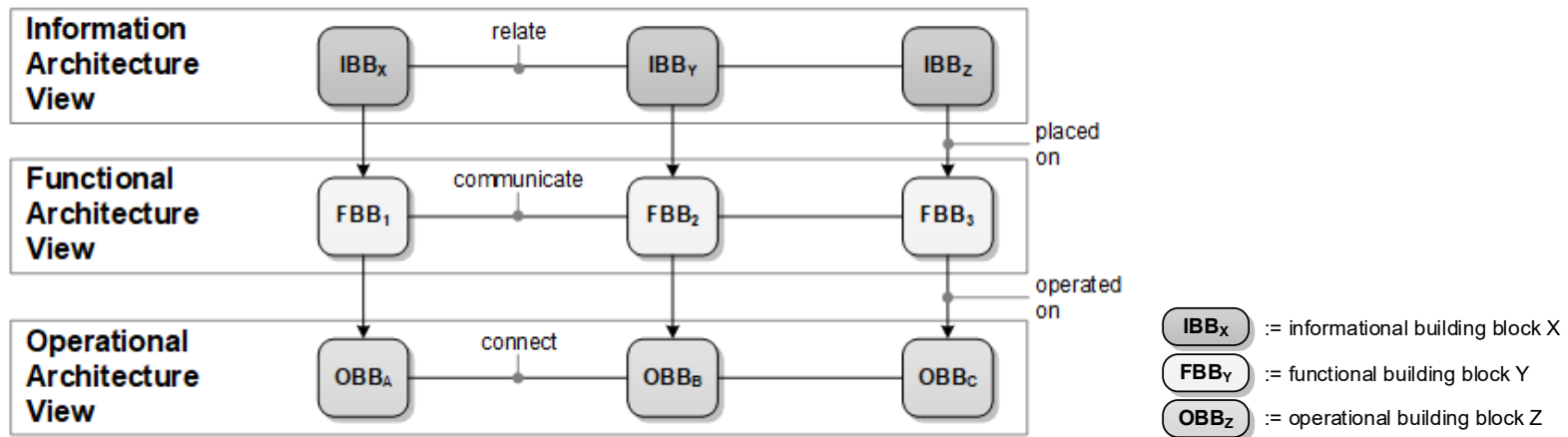
- is a **representation of a subject of interest**, where the model provides a smaller scale, and abstracted representation of the subject matter serving the interest
- **suppresses irrelevant details** in favour of those aspects relevant for the interest at hand
- needs to be **conscious of the purpose** it serves
- the effort of creating and maintaining a model must never outweigh the value of having it
- abstracts real world phenomena for the purpose of understanding aspects relevant to **answer interesting questions**
- is often based on **standardized modelling nomenclature** which can be textual, graphic, iconographic, or hybrids
- **enables communication, collaboration**, and influences how we think about and perceive the world represented by the model

# Architecture View Model

## View model

View models **enable traceability between their viewpoints**—that is, between concrete architecture views and models. By navigating, associating, or combining views, valuable insights emerge, and an architecture description gains plasticity.

**Solution architecture-related view models** typically distinguish between three elementary views and differentiate between functional, informational and operational building blocks. Furthermore, such view models clarify elementary relationships that exist between their views.

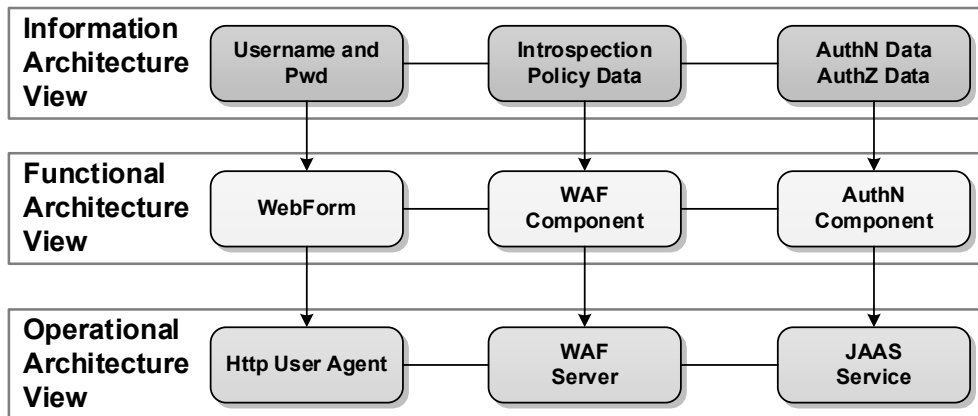


# Architecture View Model

## View model example

In this **example**, *WebForm*, *WAF Component* and *AuthN Component* are functional components. In a WebForm, *username* and *pwd* are entered. A WAF (i.e., Web Application Firewall) uses *introspection policy data* (e.g., blacklisted characters) to perform input validation of form fields sent via http.

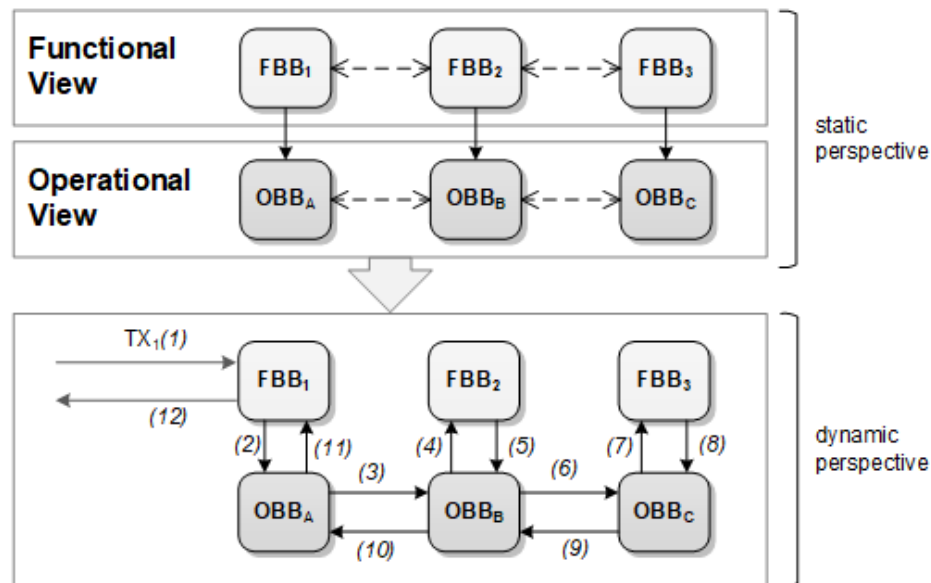
Finally, the AuthN component checks the received AuthN information (username, pwd) against *AuthN data* to verify the authenticity of the user and then against *AuthZ data* to check whether the requested access (i.e. to the website landing page) can be granted.



# Architecture View Model

## View model

For example, understanding how functional building blocks are placed on operational building blocks allows you to simulate both, the horizontal and vertical execution threads of a transaction.



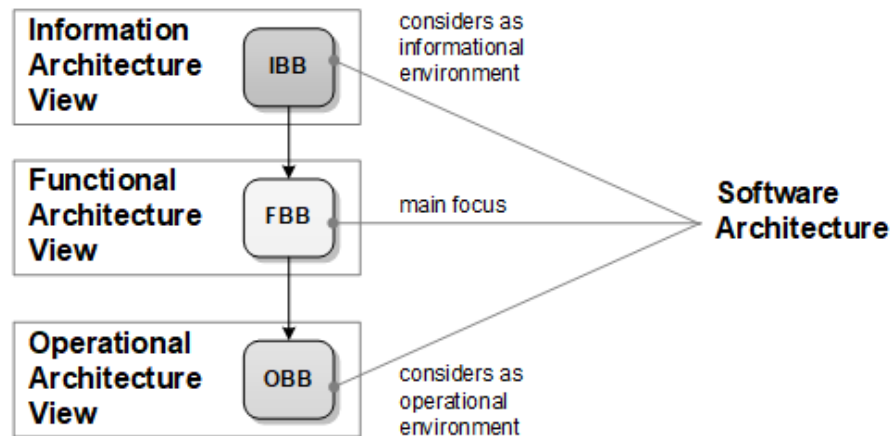
# Architecture View Model

## View model

The essential **focus of software architecture** is usually on the **functional building blocks**.

However, the software architect inevitably **considers informational** as well as **operational building blocks** when designing the software architecture.

Note on top of that that **operational building blocks are themselves functional building blocks** from the point of view of their manufacturers.

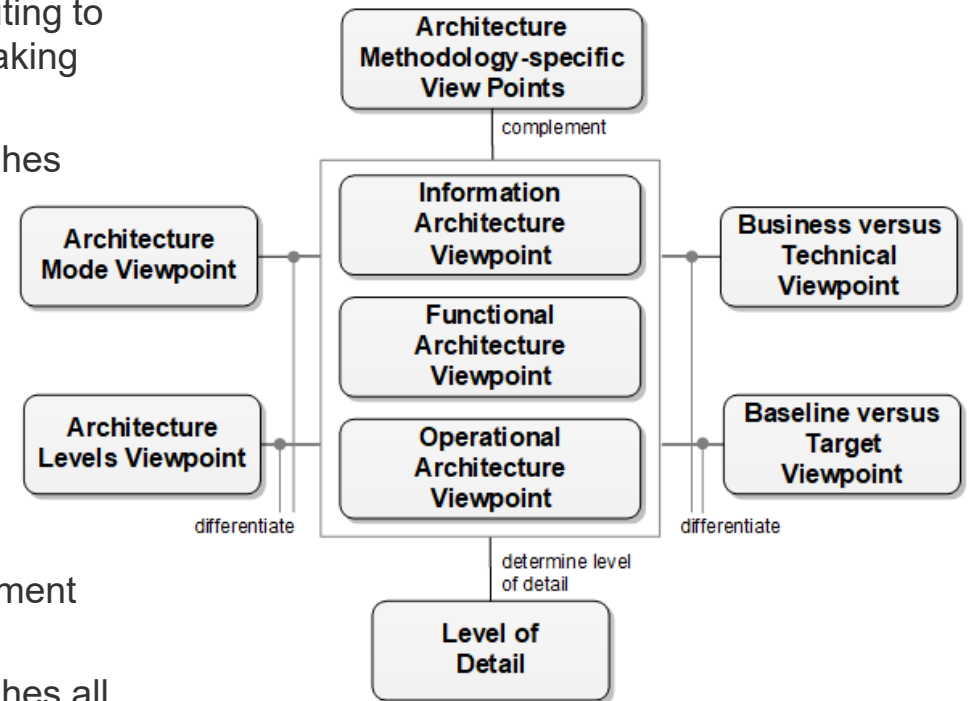


# Architecture View Model

## Complementary viewpoints

**Complementary viewpoints** exist, like ...

- **business versus technical viewpoint** distinguishes building blocks directly contributing to business outcomes (business) from those making indirect contributions (technical)
- **baseline versus target viewpoint** distinguishes architecture states (today, 1 year, 3 years, 5 years)
- **architecture mode viewpoint** distinguishes between a design-time, a run-time, and a manage-time architecture perspective
- **architecture methodology specific viewpoints** correspond to, for example, architecture elaboration methods, reference architecture methods, or architecture assessment methods
- **baseline versus target viewpoint** distinguishes all other viewpoints when chronologically different architecture states need to be delineated



# Architecture View Model

## Complementary viewpoints

**Complementary viewpoints** exist, like ...

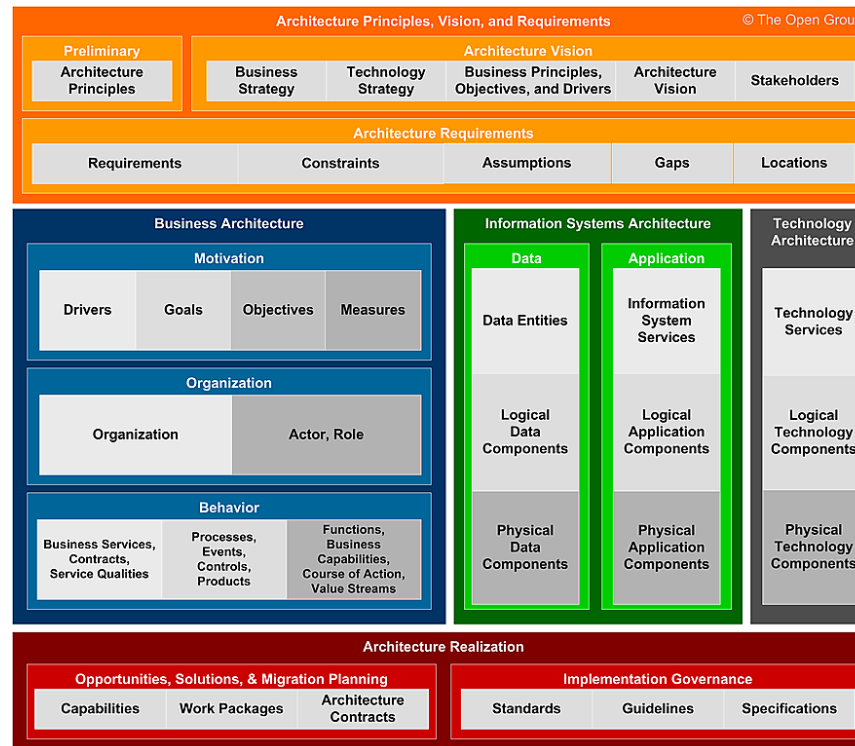
- **level of detail** needs to be calibrated across all viewpoints – it covers degree of abstractness and granularity
  - **solution level** positions landscape assets, like applications, platforms, or basic information entities. Planning horizon at this level is less than one year
  - **domain level** distinguishes structural assets (e.g. domains). Each domain manages its key landscape assets, domain-specific methodological, and reference assets. Planning horizon at the domain level is one to three years
  - **enterprise level** is the level at which you consolidate domains into an enterprise scale perspective. You also manage enterprise-wide methodological and reference assets (e.g. enterprise-wide principles). Planning horizon at enterprise level is strategic, which means three to five years



# Architecture View Model

## View model examples

The **Open Group Architecture Framework (TOGAF®)** [TOGAF 2021] proposes an approach to design, plan, implement and govern enterprise architecture by suggesting a common vocabulary, a generic information model, architecture artifacts, and tooling.





# Architecture View Model

## View model examples















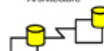















**ArchiMate®** [ArchiMate 2021] is an open and independent modelling standard for domain architectures. It supports the description, analysis, and presentation of architecture within and between designs.



# Architecture View Model

## View model examples

The **Zachman Framework** [Zachman 2021] (the old timer among the models) proposes a structured approach to holistically viewing and defining an enterprise's architecture. The basic scheme distinguishes two dimensions that introduce the intersection of two classifications.

	DATA <i>What</i>	FUNCTION <i>How</i>	NETWORK <i>Where</i>	PEOPLE <i>Who</i>	TIME <i>When</i>	MOTIVATION <i>Why</i>	
SCOPE (CONTEXTUAL)	List of Things Important to the Business 	List of Processes the Business Performs 	List of Locations in which the Business Operates 	List of Organizations Important to the Business 	List of Events/Cycles Significant to the Business 	List of Business Goals/Strategies 	SCOPE (CONTEXTUAL)
<i>Planner</i>	ENTITY = Class of Business Thing	Process = Class of Business Process	Node = Major Business Location	People = Major Organization Unit	Time = Major Business Event/Cycle	Ends/Mean = Major Business Goal/Strategy	<i>Planner</i>
BUSINESS MODEL (CONCEPTUAL)	e.g. Semantic Model  Ent = Business Entity Rein = Business Relationship	e.g. Business Process Model  Proc = Business Process IO = Business Resources	e.g. Business Logistics System  Node = Business Location Link = Business Linkage	e.g. Work Flow Model  People = Organization Unit Work = Work Product	e.g. Master Schedule  Time = Business Event Cycle = Business Cycle	e.g. Business Plan  End = Business Objective Means = Business Strategy	BUSINESS MODEL (CONCEPTUAL)
<i>Owner</i>							<i>Owner</i>
SYSTEM MODEL (LOGICAL)	e.g. Logical Data Model  Ent = Data Entity Rein = Data Relationship	e.g. Application Architecture  Proc = Application Function IO = User Views	e.g. Distributed System Architecture  Node = IIS Function (Processor, Storage, etc.) Link = Line Characteristics	e.g. Human Interface Architecture  People = Role Work = Deliverable	e.g. Processing Structure  Time = System Event Cycle = Processing Cycle	e.g. Business Rule Model  End = Structural Assertion Means = Action Assertion	SYSTEM MODEL (LOGICAL)
<i>Designer</i>							<i>Designer</i>
TECHNOLOGY MODEL (PHYSICAL)	e.g. Physical Data Model  Ent = Segment/Table/etc. Rein = Pointer/Key/etc.	e.g. System Design  Proc = Computer Function IO = Data Elements/Sets	e.g. Technology Architecture  Node = Hardware/Systems Software Link = Line Specifications	e.g. Presentation Architecture  People = User Work = Screen Format	e.g. Control Structure  Time = Execute Cycle = Component Cycle	e.g. Rule Design  End = Condition Means = Action	TECHNOLOGY MODEL (PHYSICAL)
<i>Builder</i>							<i>Builder</i>
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)	e.g. Data Definition  Ent = Field Rein = Address	e.g. Program  Proc = Language Statement IO = Control Block	e.g. Network Architecture  Node = Address Link = Protocol	e.g. Security Architecture  People = Identity Work = Job	e.g. Timing Definition  Time = Interrupt Cycle = Machine Cycle	e.g. Rule Specification  End = Sub-condition Means = Stop	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)
<i>Sub-Contractor</i>							<i>Sub-Contractor</i>
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

# Lecture Agenda



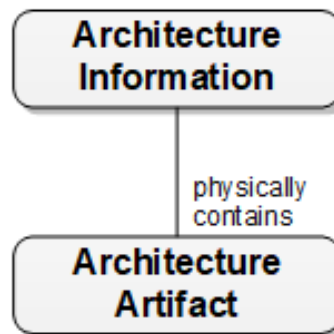
- Architecture View Model
- Architecture Description
- Kruchten 4+1 View Model

# Architecture Description

## Architecture artifacts and architecture information

**Architecture manifests in architecture descriptions** (aka: architecture artifacts, architecture work products).

Artifacts can be more or less structured and formalized. They are **means by which architects process architecture information in a tangible manner**.



# Architecture Description

## Architecture artifacts and architecture information

Physically **tangible architecture information** can be accessed, sent, and exchanged between cooperating parties in a controlled and secure manner.

**Artifacts protect their content from unauthorized access.** They can be signed, versioned, archived, and physically referenced from other artifacts.

**Artifacts are agreed upon as deliverables**, whose delivery can be efficiently quantified and verified.

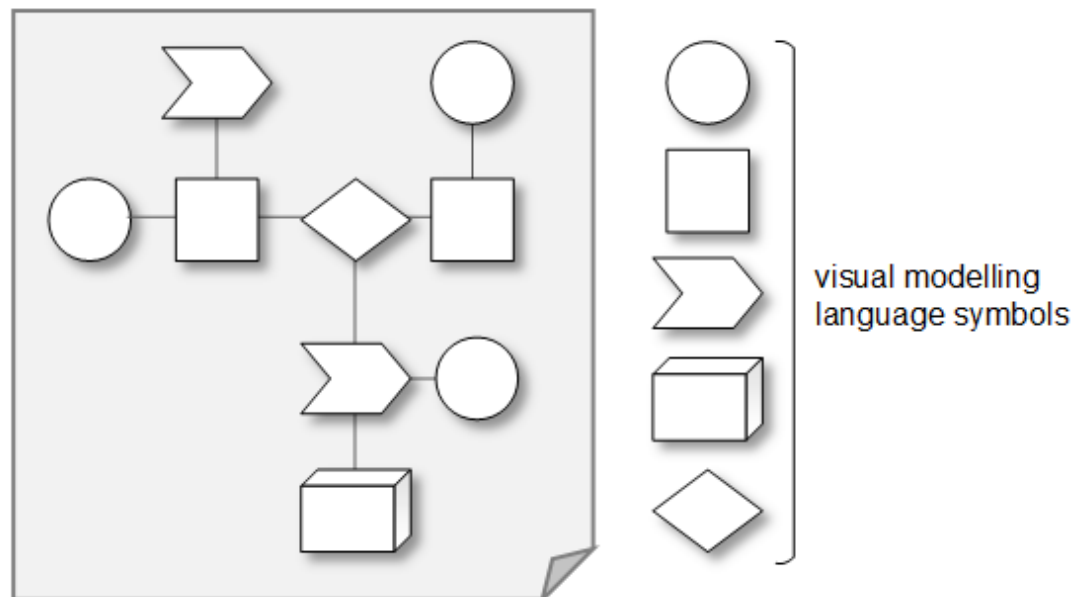
**Artifacts can represent final information, or they can be dynamically generated** based on an architecture repository.

Architecture information – structured, unstructured, or semi-structured – is compiled into artifacts and can take **various forms, including text, diagrams, lists, or matrices**. For example, statements formulated in plain English take the text form. However, text can also take the form of a formal language, such as a pseudocode fragment, which expresses the algorithmic structure of an asset's capability.

# Architecture Description

## Architecture artifacts and architecture information

**Illustrations** and diagrams exist in a variety of visual nomenclatures and modelling languages, each with its own standardized visual symbols, such as the Unified Modelling Language (UML).



# Architecture Description

## Architecture artifacts and architecture information

**Lists** are another common form of representing architecture information. A pattern catalog, a report on all architecture assets in the purchasing domain, or a portfolio viewpoint in a software architecture method are examples of artifacts that take list form

Building Blocks	Attribute <sub>1</sub>	Attribute <sub>2</sub>
Building Block <sub>A</sub>	• _____ • ____	• _____ • ____
Building Block <sub>B</sub>	• _____ • ____	• _____ • ____
Building Block <sub>C</sub>	• _____ • ____	• _____ • ____
Building Block <sub>D</sub>	• _____ • ____	• _____ • ____
Building Block <sub>E</sub>	• _____ • ____	• _____ • ____
Building Block <sub>F</sub>	• _____ • ____	• _____ • ____

attribute-based building block description

# Architecture Description

## Architecture artifacts and architecture information

**Matrices** is a universally used form that maps two architecturally relevant information dimensions, unidirectionally or bidirectionally, to each other. For example, which services are supported by a considered application.

BB <sub>x</sub> \ BB <sub>y</sub>	X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>
Y <sub>1</sub>	R <sub>11</sub>	R <sub>12</sub>	
Y <sub>2</sub>	R <sub>21</sub>		
Y <sub>3</sub>			
Y <sub>4</sub>			
Y <sub>5</sub>			
Y <sub>6</sub>			
Y <sub>7</sub>			
Y <sub>8</sub>			
Y <sub>9</sub>			
Y <sub>10</sub>			
Y <sub>11</sub>			
Y <sub>12</sub>			

building block dimensions (X<sub>m</sub>, Y<sub>n</sub>)

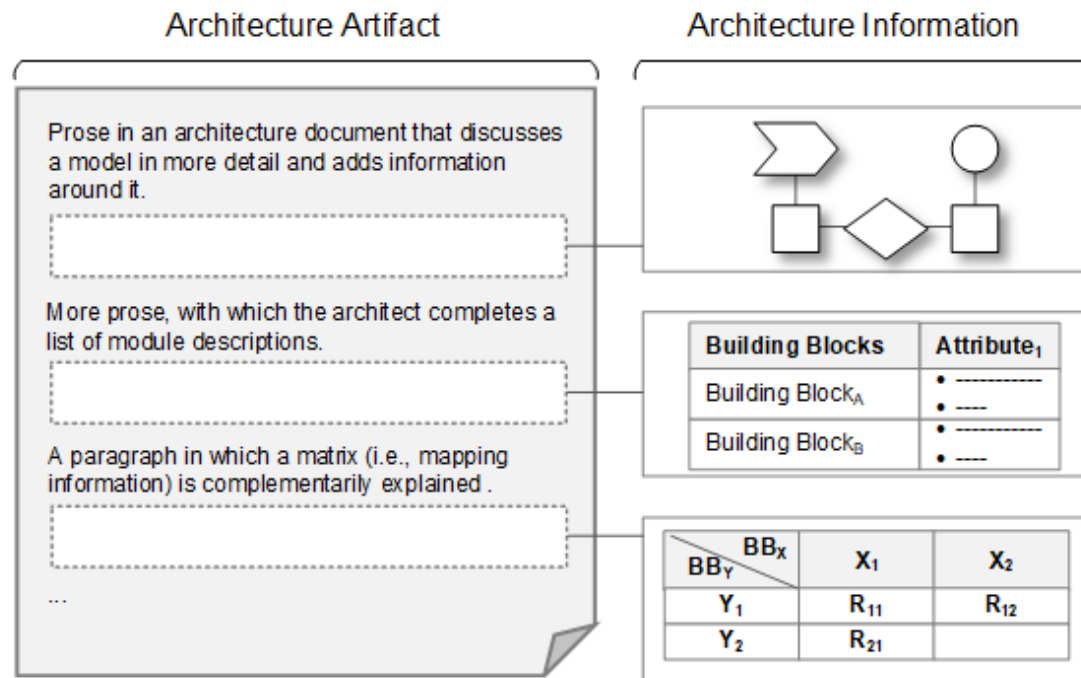
mappings, relationships (R<sub>ij</sub>) between dimensions X<sub>m</sub> and Y<sub>n</sub>



# Architecture Description

## Architecture artifacts and architecture information

Architecture **information** from many sources and in different forms is finally **combined** into an **overarching architecture artifact**.



# Lecture Agenda

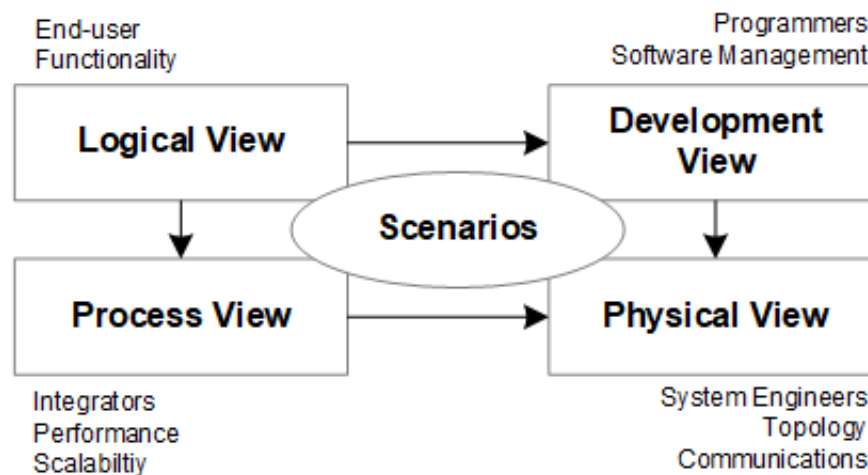


- Architecture View Model
- Architecture Description
- Kruchten 4+1 View Model

# Architecture – Kruchten 4+1 View Model Overview

The view of **software architecture manifestation** as a total of different views goes back to the work of Philippe Kruchten [Kruchten 1995].

The **four core views are complemented by one** (hence +1) specification view (scenarios). The arrows between two views mean that the view at the beginning of the arrow influences the view at the end of the arrow.



# Architecture – Kruchten 4+1 View Model

## Overview

The **logical view** represents cooperation relationships of components—that is, a representation of how functional components (e.g., objects) cooperate to realize scenarios.

The **development view** describes the static organization of components in their development environment—thus a representation of which design time structures (e.g., classes) factorize the logical view.

The **process view** captures concurrency and synchronization aspects of the design—that is, a representation of how components are grouped into processes and what cooperative relationships exist between processes.

The **physical view** describes mapping(s) of software to hardware, or an underlying operating layer, and corresponding aspects of software distribution—that is, a representation of how software and hardware ultimately form an overarching operational unit.

# Architecture – Kruchten 4+1 View Model

## Overview

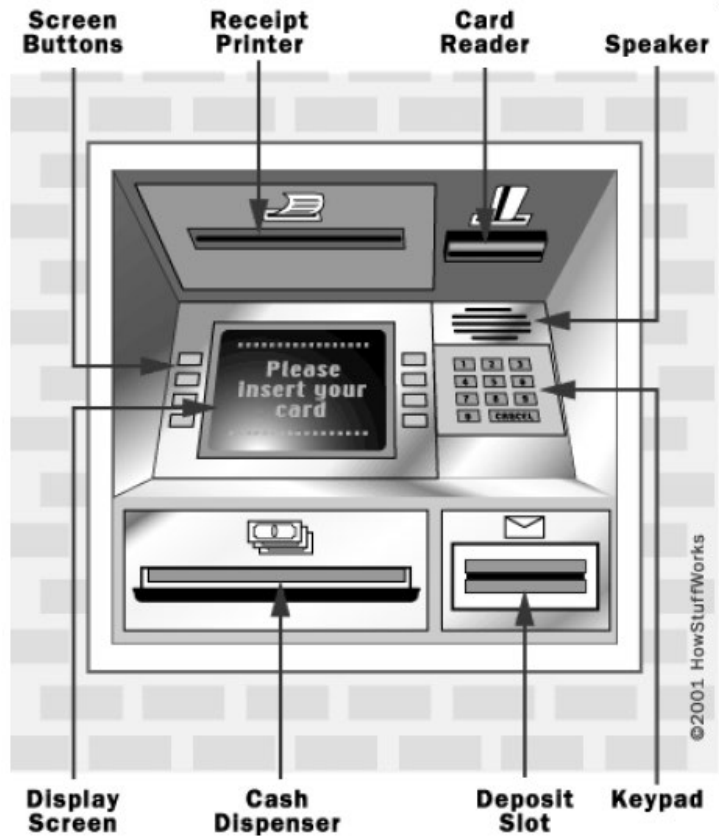
4+1 view model illustrated with an ATM example

- An ATM (Automated Teller Machine) is a computer
- It has a small display and something similar to a keyboard
- Users have access to ATMs on a bank's perimeter
- ATMs therefore need to be connected to the bank's Backoffice systems
- ATMs support a broad variety of functions – a common usage pattern is
  - ATM asks for the user's preferred language
  - ATM asks the user to insert his/her bank identification card
  - ATM reads information from the card to identify the user
  - ATM asks the user to authenticate
  - If the user successfully authenticated he is free to choose an ATM function
  - After the user is done with the ATM transaction the ATM spits out the card
  - The user leaves

# Architecture – Kruchten 4+1 View Model

## Overview

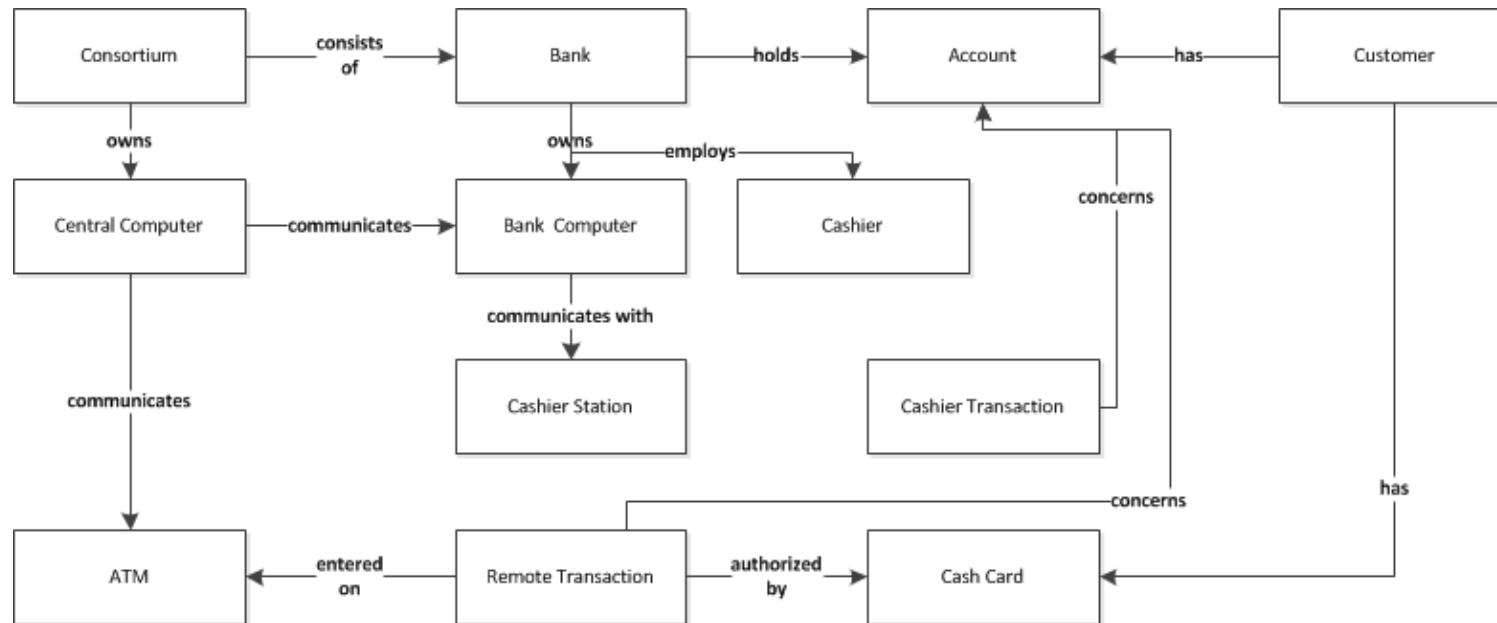
4+1 view model (ATM) – Real World



# Architecture – Kruchten 4+1 View Model

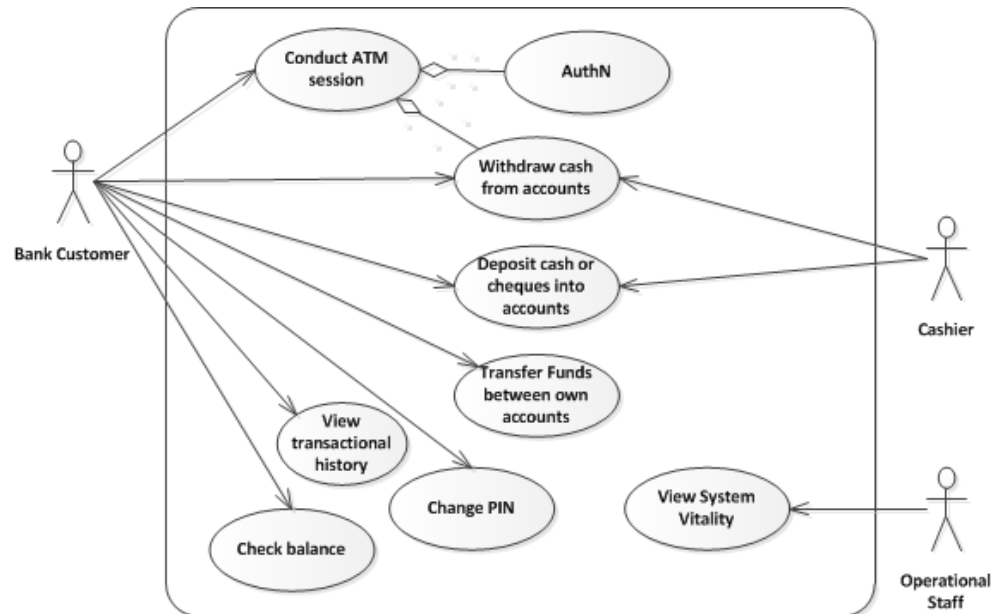
## Overview

4+1 view model (ATM) – Domain Model



# Architecture – Kruchten 4+1 View Model Overview

## 4+1 view model (ATM) – Scenarios Overview





# Architecture – Kruchten 4+1 View Model Overview

## 4+1 view model (ATM) – Scenario “Conduct ATM Session”

<b>Name</b>	<i>Conduct ATM session (including AuthN)</i>
<b>Brief Description</b>	
<b>Actor Descriptions</b>	<i>Bank Customer</i>
<b>Preconditions</b>	<i>Bank Customer physically present at ATM</i>
<b>Basic Flow of Events</b>	<ol style="list-style-type: none"> <li>1. ATM screen displays welcome and prompts the user to enter an account number</li> <li>2. User enters account number using the ATM keypad</li> <li>3. ATM screen prompts the user to enter the PIN associated with the specified account number</li> <li>4. User enters his/her personal PIN using the keypad</li> <li>5. If user enters a valid account number and the correct PIN for that account, the screen displays the main menu</li> <li>6. User performs transaction</li> <li>7. User closes ATM session</li> <li>8. User leaves ATM</li> </ol>
<b>Alternative Flows</b>	<u>Alternative Flow: Wrong account number entered</u> <ol style="list-style-type: none"> <li>1. Screen displays an appropriate message, then the ATM returns to Step 1 to restart the authentication process</li> </ol>
<b>Post-conditions</b>	
<b>Extension Points</b>	
<b>Special Requirements</b>	
<b>Supporting information</b>	

# Architecture – Kruchten 4+1 View Model

## Overview

### 4+1 view model (ATM) – Scenario “Withdraw Money from Bank Account”

<b>Name</b>	<i>Withdraw Money from Bank Account</i>
<b>Brief Description</b>	
<b>Actor Descriptions</b>	<i>Bank Customer</i>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>- in context of UC Conduct ATM Session</li> <li>- ATM cash dispenser contains enough cash</li> </ul>
<b>Basic Flow of Events</b>	<ol style="list-style-type: none"> <li>1. ATM screen displays a menu of standard withdrawal amounts and an option to cancel the transaction</li> <li>2. User enters a menu selection using the keypad</li> <li>3. If withdrawal amount is greater than the user's account balance <ol style="list-style-type: none"> <li>a. the screen displays a message stating this and telling the user to choose a smaller amount</li> <li>b. ATM returns to step 1</li> </ol> </li> <li>4. If the user chooses to cancel, the ATM displays the main menu and waits for user input</li> <li>5. If the withdrawal amount chosen is less than or equal to the user's account balance, the ATM proceeds</li> <li>6. ATM debits the withdrawal amount from the user's account in the bank's database</li> <li>7. ATM cash dispenser dispenses the desired amount of money to the user</li> <li>8. ATM screen displays a message reminding the user to take the money</li> </ol>
<b>Alternative Flows</b>	
<b>Post-conditions</b>	
<b>Extension Points</b>	
<b>Special Requirements</b>	
<b>Supporting information</b>	

# Architecture – Kruchten 4+1 View Model

## Overview

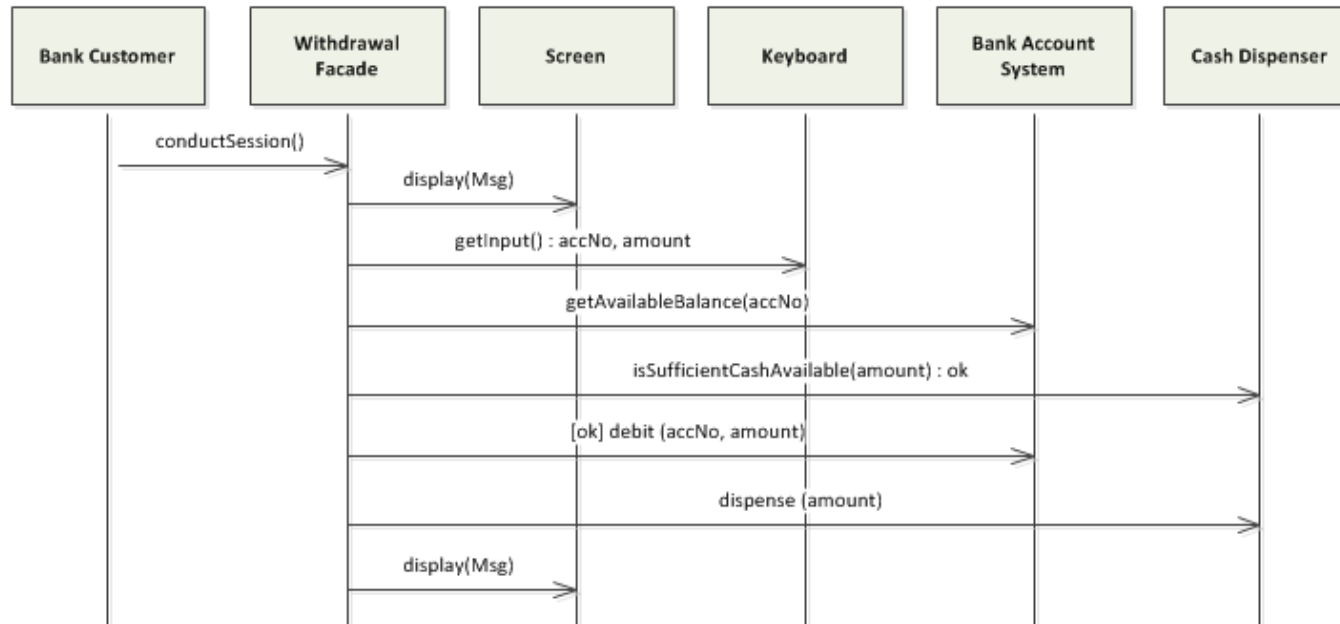
### 4+1 view model (ATM) – Scenario “Deposit Money to Bank Account”

<b>Name</b>	<i>Deposit Money to Bank Account</i>
<b>Brief Description</b>	
<b>Actor Descriptions</b>	<i>Bank Customer</i>
<b>Preconditions</b>	
<b>Basic Flow of Events</b>	<ol style="list-style-type: none"> <li>1. ATM screen prompts the user to enter a deposit amount or type 0 to cancel</li> <li>2. User enters a deposit amount or 0 using the keypad</li> <li>3. If user chooses to cancel the transaction (by entering 0), the ATM displays the main menu and waits for user input</li> <li>4. If user specifies a deposit amount, the ATM proceeds</li> <li>5. ATM screen displays a message telling the user to insert a deposit envelope</li> <li>6. If the deposit slot receives a deposit envelope within two minutes, the ATM credits the deposit amount to the user's account in the bank's database (i.e., adds the deposit amount to the user's account balance)</li> </ol>
<b>Alternative Flows</b>	
<b>Post-conditions</b>	
<b>Extension Points</b>	
<b>Special Requirements</b>	
<b>Supporting information</b>	

# Architecture – Kruchten 4+1 View Model

## Overview

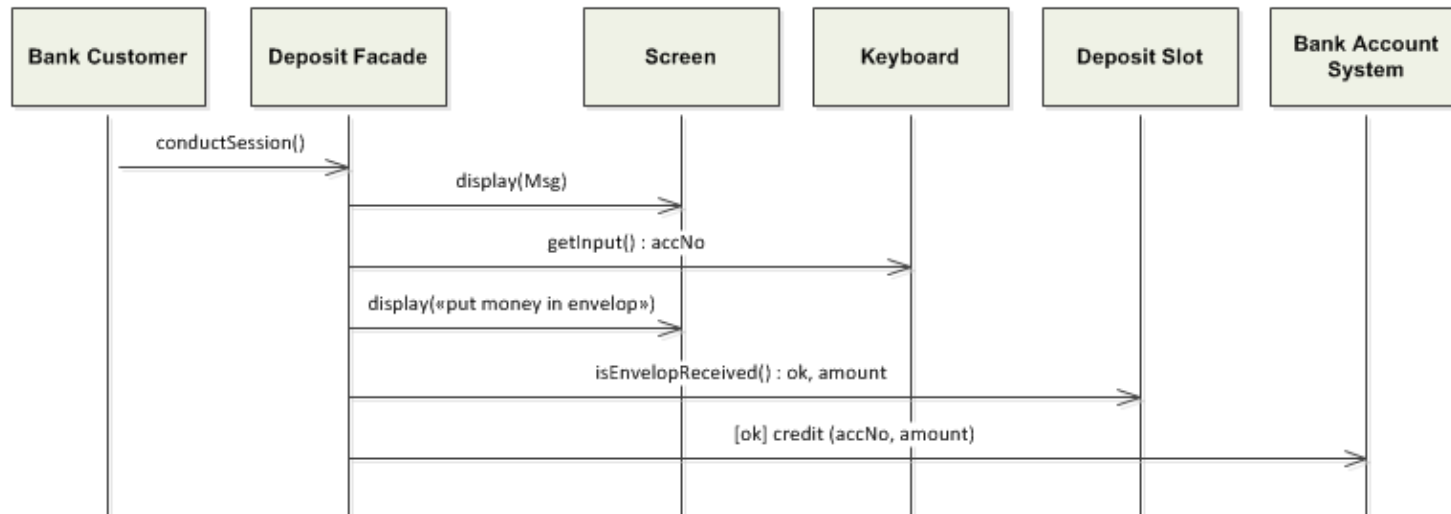
4+1 view model (ATM) – Logical View “Withdraw Money from Bank Account”



# Architecture – Kruchten 4+1 View Model

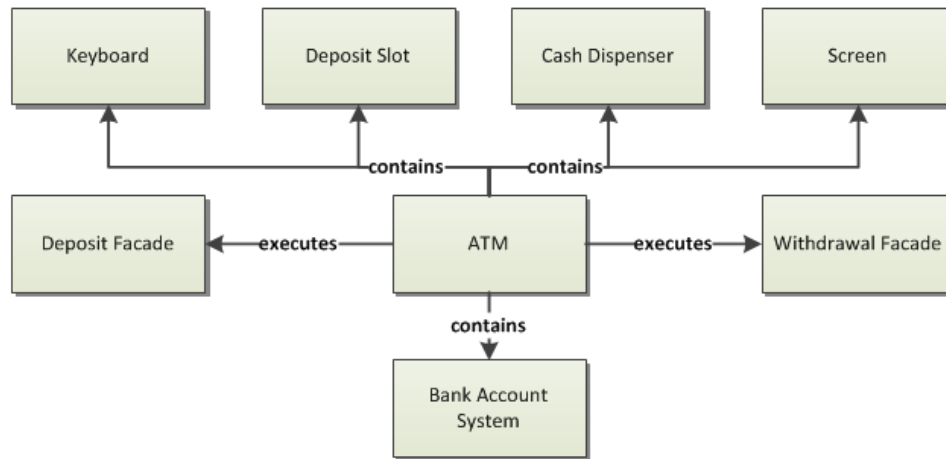
## Overview

4+1 view model (ATM) – Logical View “Deposit Money to Bank Account”



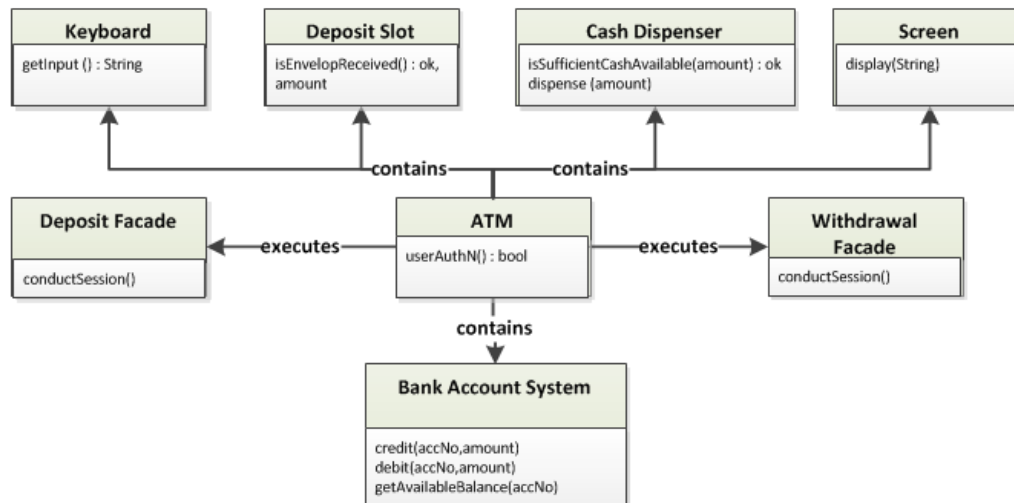
# Architecture – Kruchten 4+1 View Model Overview

4+1 view model (ATM) – Development View (Overview)



# Architecture – Kruchten 4+1 View Model Overview

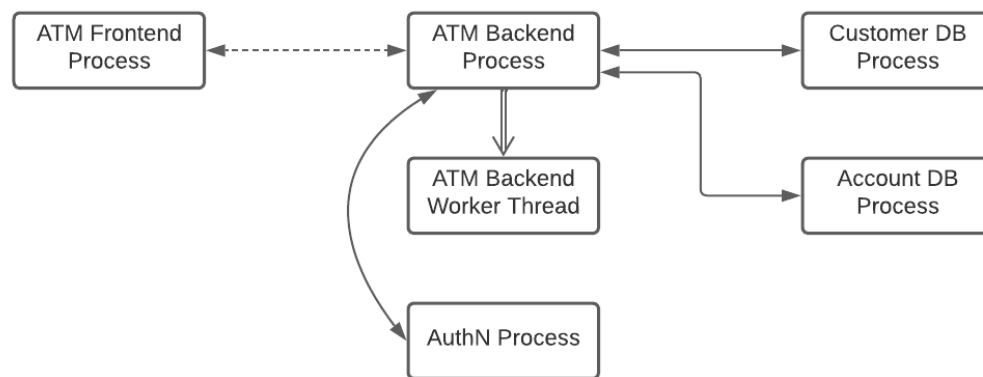
## 4+1 view model (ATM) – Development View (Details)



# Architecture – Kruchten 4+1 View Model

## Overview

### 4+1 view model (ATM) – Process View



- ←- - - - -> Messaging (2PC Tx, queue, asynchronous)
- ← - - - - -> RPC (request/response, synchronous)
- ← - - - - -> Multithreading (fork work threads for incoming requests)

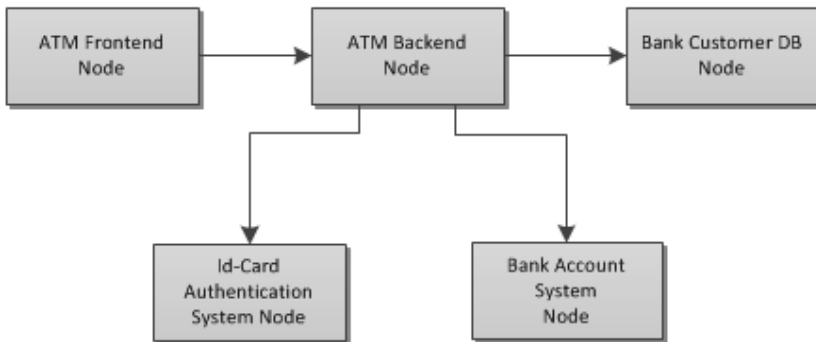
Process (hosts)	Component
ATM Frontend	Keyboard, Deposit Slot, Cash Dispenser, Screen, Deposit Facade, Withdrawal Facade
ATM Backend Process	ATM
Customer DB Process	<i>Bank Customer System</i>
AuthN Process	<i>AuthN System</i>
Account DB Process	Bank Account System



# Architecture – Kruchten 4+1 View Model

## Overview

4+1 view model (ATM) – Physical View



Physical Node (hosts)	Process
ATM Frontend	ATM Frontend Process
ATM Backend	ATM Backend Process, ATM Backend Worker Thread
Bank Customer DB	Customer DB Process
Id-Card AuthN System	AuthN Process
Bank Account System	Account DB Process

# Architecture – Kruchten 4+1 View Model Exercise



# Architecture – Kruchten 4+1 View Model Exercise



# Architecture – Architecture Description Example



\* ArchitectureDescription-Healthcare Company-V1.0

# Bibliography

## Lecture

### [ArchiMate® 2021]

Open Group, *ArchiMate*, 2021, <https://www.opengroup.org/archimate-home>

### [Apostel 1960]

Apostel, Leo, Towards the formal study of models in the non-formal sciences, *Synthese* 12, 1960

### [Kruchten 1995]

Kruchten, Philippe, *Architectural Blueprints – The '4+1' View Model of Software Architecture*, IEEE Software 12 (6), November 1995, pp. 42-50,  
<http://dx.doi.org/10.1109/52.469759>,  
[https://www.researchgate.net/publication/220018231\\_The\\_41\\_View\\_Model\\_of\\_Architecture](https://www.researchgate.net/publication/220018231_The_41_View_Model_of_Architecture)

### [TOGAF® 2021]

Open Group, *The Open Group Architecture Framework*, 2021,  
<https://www.opengroup.org/togaf>

# Bibliography

## Lecture

### [Vogel, Arnold et al 2011]

Vogel, Oliver; Arnold, Ingo; Chugtai, Arif; Kehrer, Timo, *Software Architecture: A Comprehensive Framework and Guide for Practitioners*, Springer Science and Business Media, Berlin Heidelberg, 2011

### [Zachman 2021]

Zachman, Zachman; *the Zachman framework*, 2021, <https://www.zachman.com/>

# Questions

