4+1 View Model of Software Architecture

Basharat Hussain

23rd Nov. 2016

COMSTAS IIT - WAH

Outline

- Problem
- Solution
- 4+1 view model
    - Logical view
    - Process view
    - Development view
    - Physical view
    - Use-case view
- The Notations

The Problem

- Arch. documents over-emphasize an aspect of development (i.e. team organization) or do not address the concerns of all stakeholders.

- The stakeholders of software system:
  - end-user,
  - developers,
  - system engineers,
  - project managers
- Software engineers struggled to represent more on one blueprint, and so architecture documents contains complex diagrams

Different stakeholders – different prospective

- Architecture also means different things to different stakeholders.
- For example,
  - a Network Engineer would only be interested in the hardware and network configuration of the system;
  - a Project Manager in the key components to be developed and their timelines;
  - a Developer in classes that make up a component; and
  - a Tester in testable scenarios.

Solution

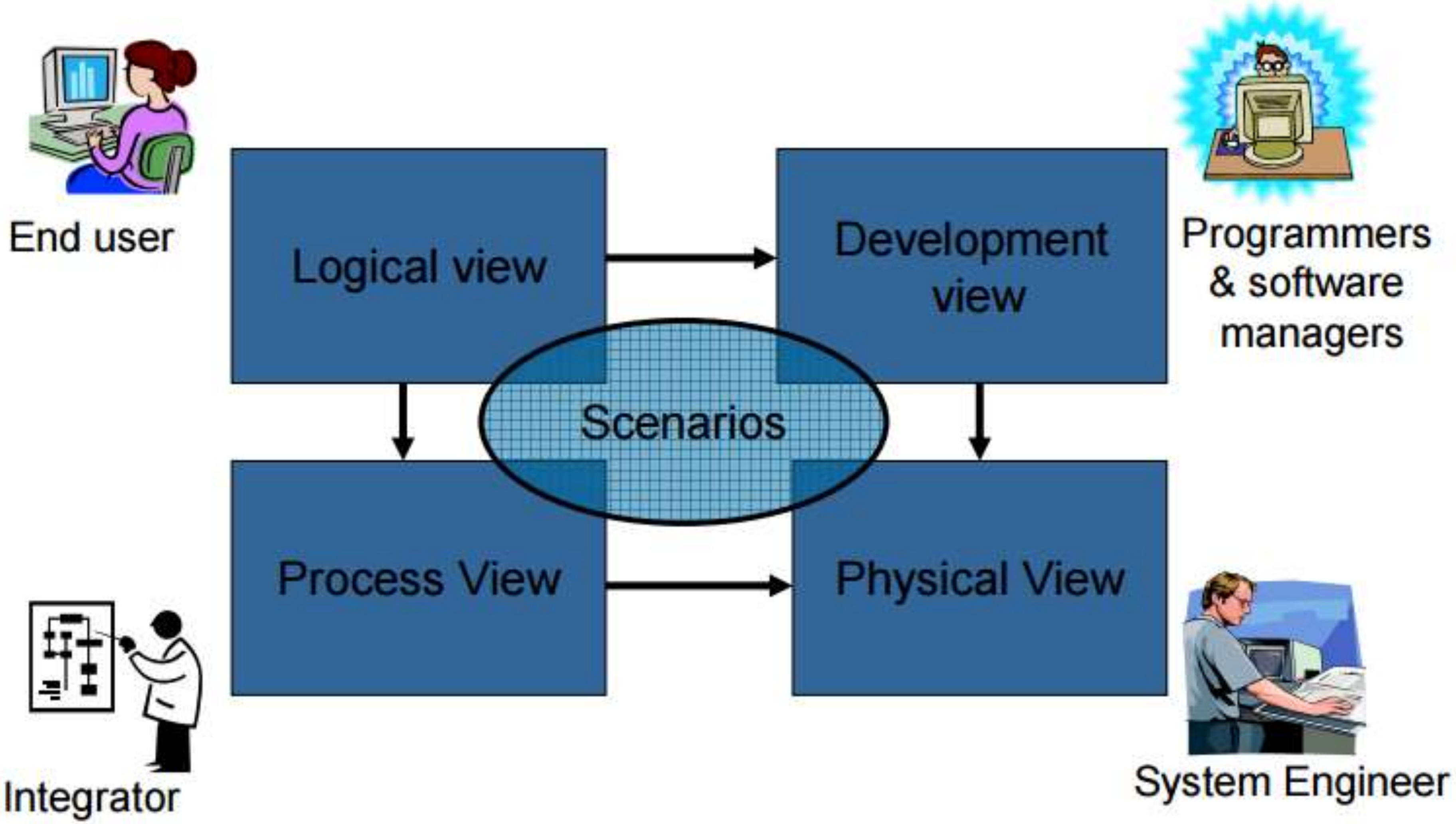
Solution came from “4+1” view model

- The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers.

- Suitable for large and challenging architectures

- Describe different aspects of the system into different views.
- (Why?) Because different stakeholders have different aspects
  - DEVELOPERS – Aspects of Systems like classes
  - SYSTEM ADMINISTRATOR – Deployment, hardware and network configuration
  - TESTER , PM, CUSTOMER -- ???
- “4+1” view model provides a “*better organization with better separation of concern*”.

4+1 View Model of Architecture

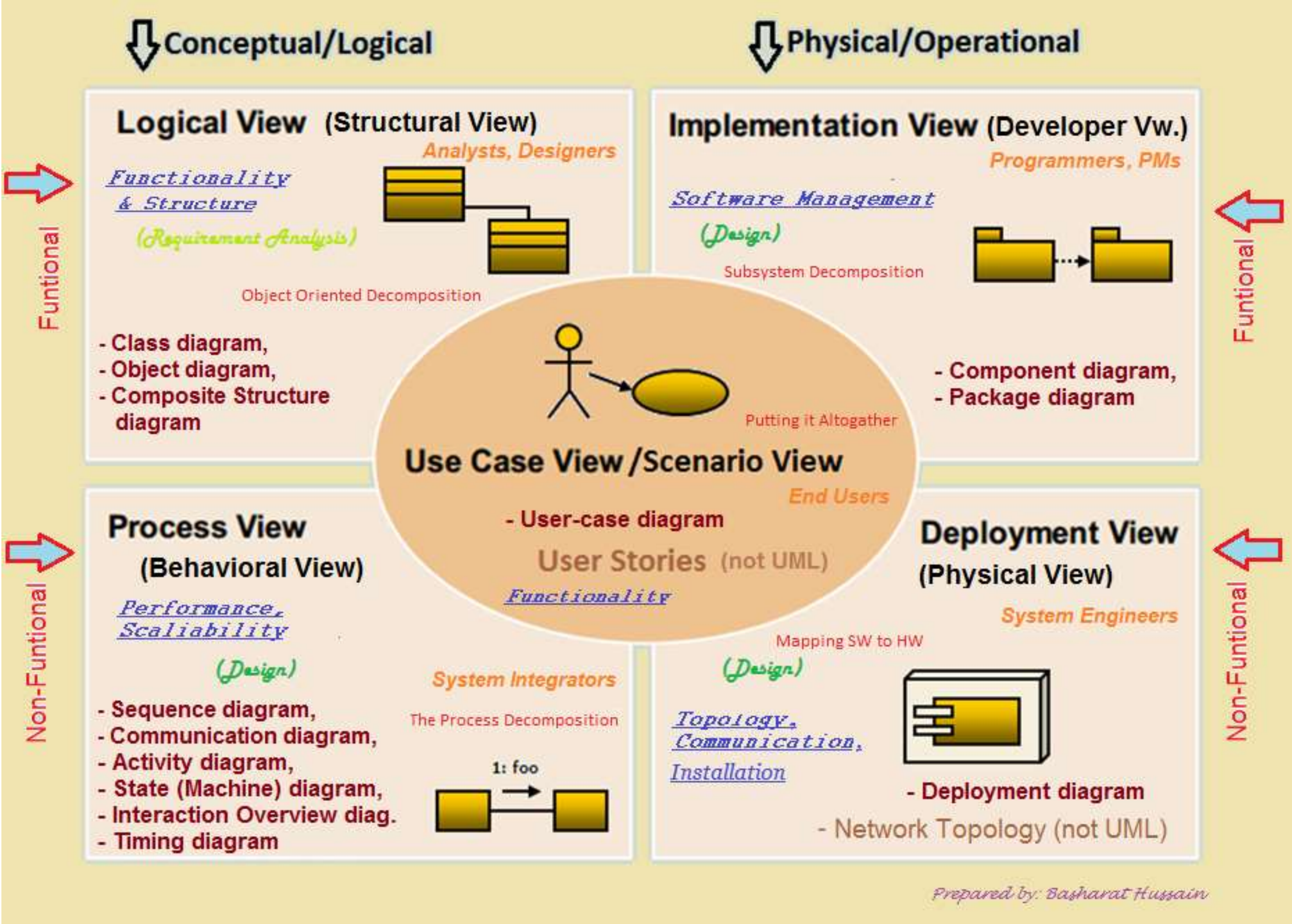


4+1 view model architecture *(for distributed systems)*

1. **Logical View (or Structural View)** - *an object model of the design*
2. **Process View (or Behavioral View)** - *concurrency and synchronization aspects*
3. **Development View (or Implementation View)** - *static organization (subset) of the software*
4. **Physical View (or Deployment View)** - *mapping of the software to the hardware*

+1. Use-cases View (or Scenarios) - *various usage scenarios*

13 UML2.0 diagrams in ‘4+1 View Model’ Architecture



Conceptual/Logical

Physical/Operational

Funtional

Logical View (Structural View)

Analysts, Designers

Functionality & Structure

(Requirement Analysis)

Object Oriented Decomposition

- Class diagram,
- Object diagram,
- Composite Structure diagram

Implementation View (Developer Vw.)

Programmers, PMs

Software Management

(Design)

Subsystem Decomposition

- Component diagram,
- Package diagram

Funtional

Putting it Altogather

Use Case View /Scenario View

End Users

- User-case diagram

User Stories (not UML)

Functionality

Non-Funtional

Process View

(Behavioral View)

Performance, Scaliability

(Design)

System Integrators

The Process Decomposition

1: foo

- Sequence diagram,
- Communication diagram,
- Activity diagram,
- State (Machine) diagram,
- Interaction Overview diag.
- Timing diagram

Deployment View

(Physical View)

System Engineers

Mapping SW to HW

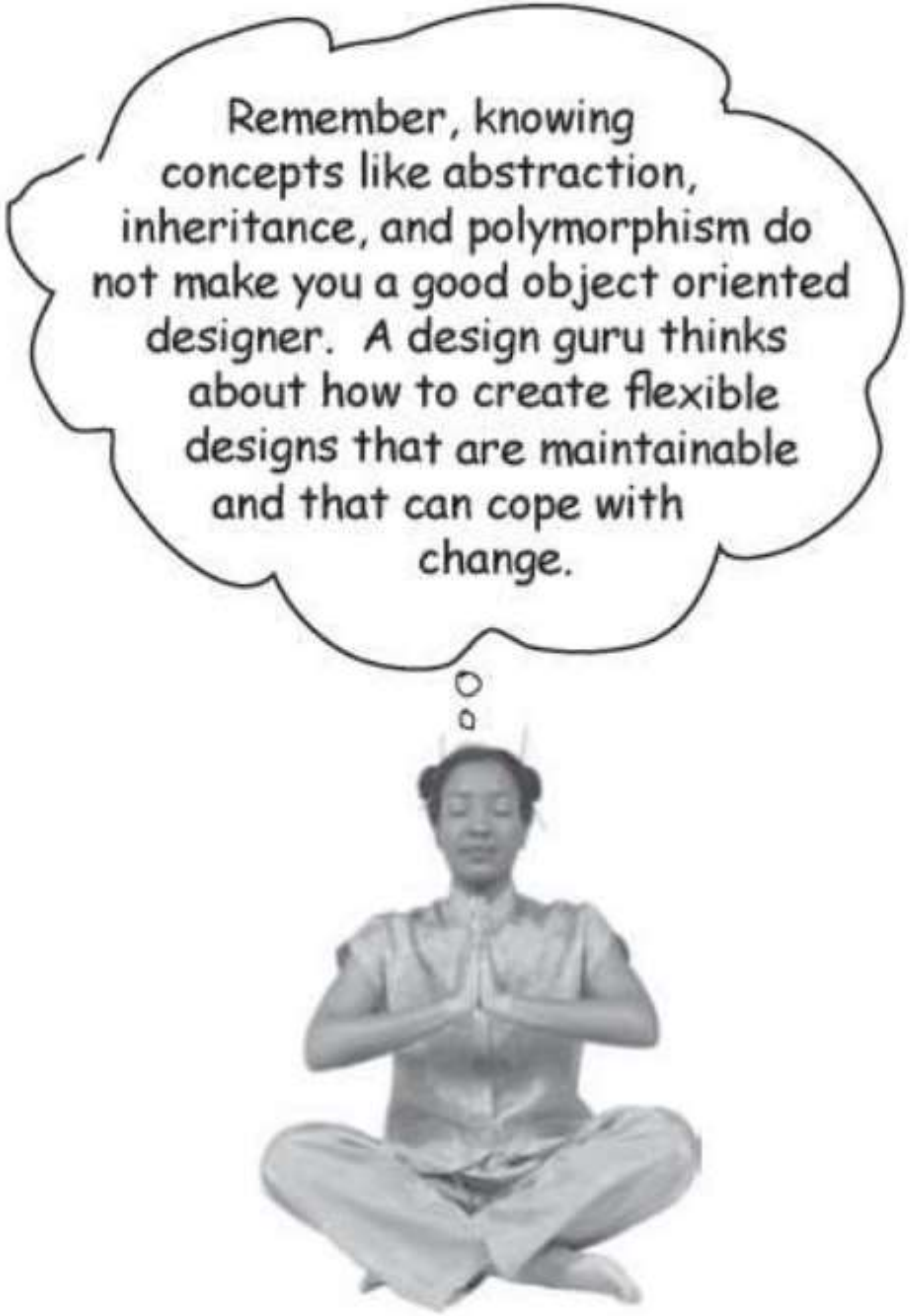
(Design)

Topology, Communication, Installation

- Deployment diagram
- Network Topology (not UML)

Non-Funtional

Prepared by: Basharat Hussain


Remember, knowing concepts like abstraction, inheritance, and polymorphism do not make you a good object oriented designer. A design guru thinks about how to create flexible designs that are maintainable and that can cope with change.

1. Logical View = (The Object-oriented Decomposition)

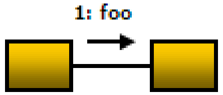
- **Viewer:** End-user
- **Considers:** Functional Requirements- What the system should provide in terms of services to its users.

- **What this view shows?**
- *“This view shows the components (objects) of the system as well as their interaction/relationship”.*
- Notation: Object and dynamic models
- **UML diagrams:** Class, Object, State Machine, Composite Structure diagrams

2. Process View = (The Process Decomposition)

- **Viewer:** Integrator(s)
- **Considers:** Non-Functional Requirements - (concurrency, performance, scalability, usability, resilience, re-use, comprehensibility, economic and technology constraints, trade-offs, and cross-cutting concerns - like security and transaction management)

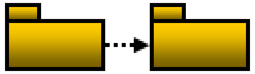
- **What this view shows?**
- *"This view shows the processes (workflow rules) of the system and how those processes communicate with each other".*
- **UML diagrams:** Sequence, Communication, Activity, Timing, Interaction Overview diagrams



3. Development/Implementation View

= (The Subsystem Decomposition)

- **Viewer:** Programmers and Software Managers
- **Considers:** Software module organization (Hierarchy of layers, software management, reuse, constraints of tools)
- **What this view shows?**
- *"This view shows the building blocks of the system".*
- **UML diagrams:** Component, Package diagrams
  - Package Details, Execution Environments, Class Libraries, Layers, Sub-system design



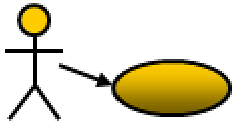
4. Physical/Deployment View

= (Software to Hardware Mapping)

- **Viewer:** System Engineers
- **Considers:** Non-functional Requirements for hardware (Topology, Communication)
- **What this view shows?** Non-functional
- *“This view shows the system execution environment”.*
- **UML diagrams:** Deployment diagrams
- **Non-UML diagrams:** Network Topology (not in UML)

5. Use-case View/Scenarios = (putting it altogether)

- **Viewer:** All users of other views and Evaluators
- **Considers:** System consistency, validity
- **What this view shows?**
- *"This view shows the Validation and Illustration of system completeness. This view is redundant with other views.".*
- **UML diagrams:** Use-case diagram, User stories



Relationships between Views

- The *Logical View* and the *Process View* are at a conceptual level and are used from analysis to design.
- The *Development View* and the *Deployment View* are at the physical level and represent the actual application components built and deployed.
- The *Logical* View and the Development View are tied closer to functionality (functional aspect) . They depict how functionality is modeled and implemented.
- The *Process* View and Deployment View realizes the non-functional aspects using behavioral and physical modeling.
- *Use Case* View leads to structural elements being ***analyzed*** in the Logical View and ***implemented*** in the Development View. The scenarios in the Use Case View are ***realized*** in the Process View and ***deployed*** in the Physical View.

Why is it called the 4 + 1 instead of just 5?

- **The Use-case View:** The use case view has a special significance.

Views are effectively redundant (i.e. Views are interconnected).

➢However, all other views would not be possible without use case view.

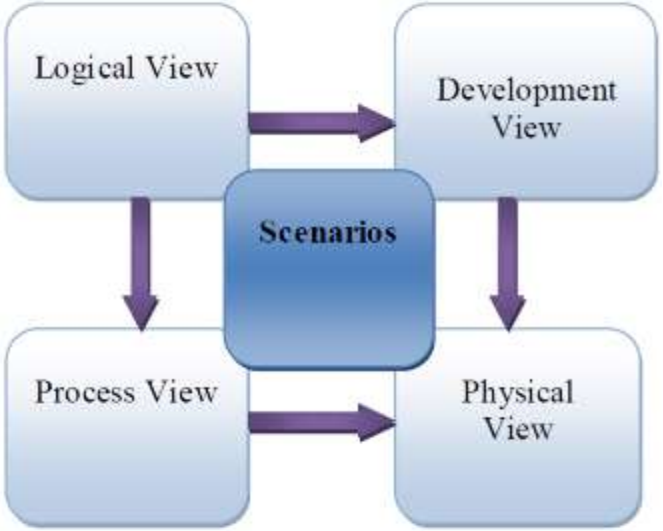
➢It details the high levels requirements of the system.

➢The other views detail how those requirements are realized.

Correspondence between views

- Views are interconnected.
- They are very close, but <u>the larger the project, the greater the distance.</u>
- Start with Logical view (Req. Doc) and Move to Development or Process view and then finally go to Physical view.

1. From logical to Process view
   - Two strategies to analyze level of concurrency:
     - Inside-out: starting from Logical structure
     - Outside-in: starting from physical structure
2. From Logical to development view
   - Grouping to subsystems is based on:
     - Classes
     - Class packages
     - Line of codes
     - Team organization

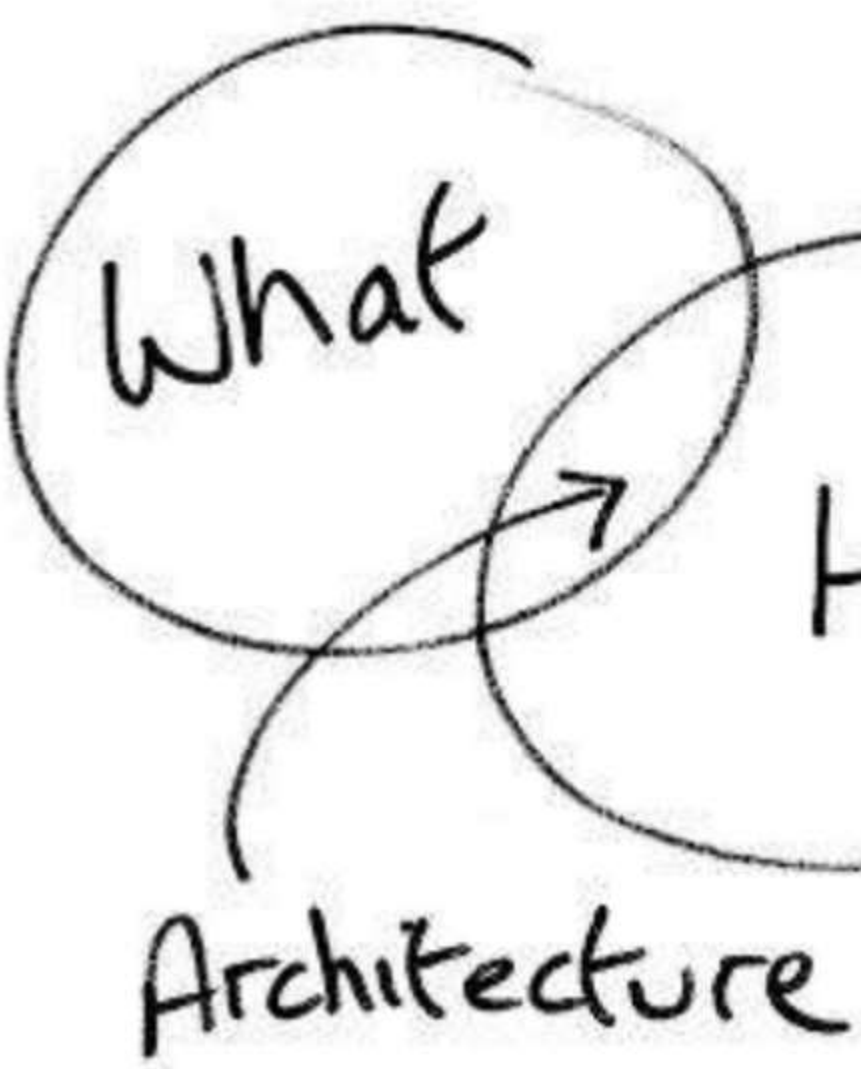


Software Architecture Definition by IEEE

“Software Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.”

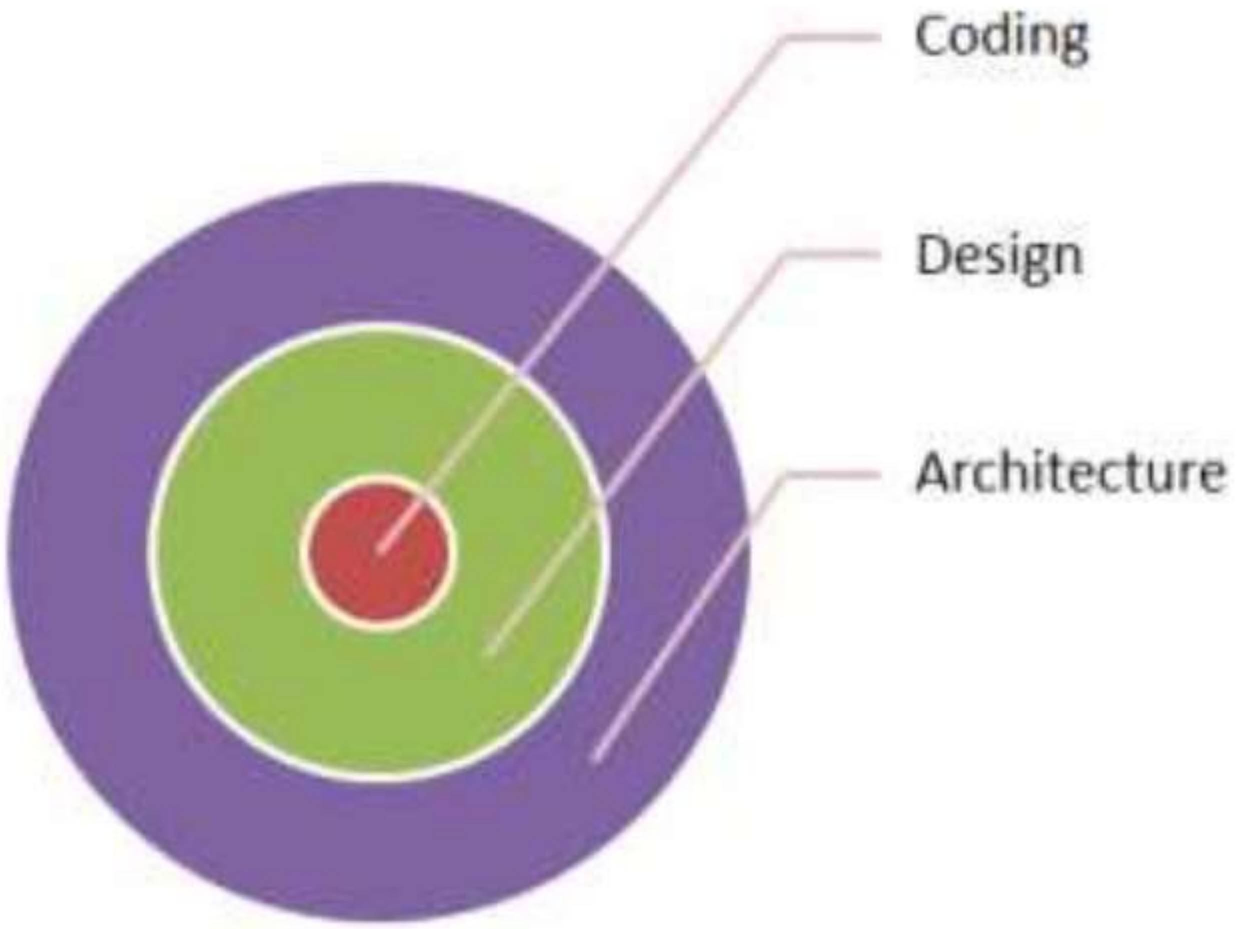
— The definition of Software Architecture as per IEEE Recommended Practice for Architectural Description of Software Intensive Systems (IEEE 1471-2000)

What is Software Architecture

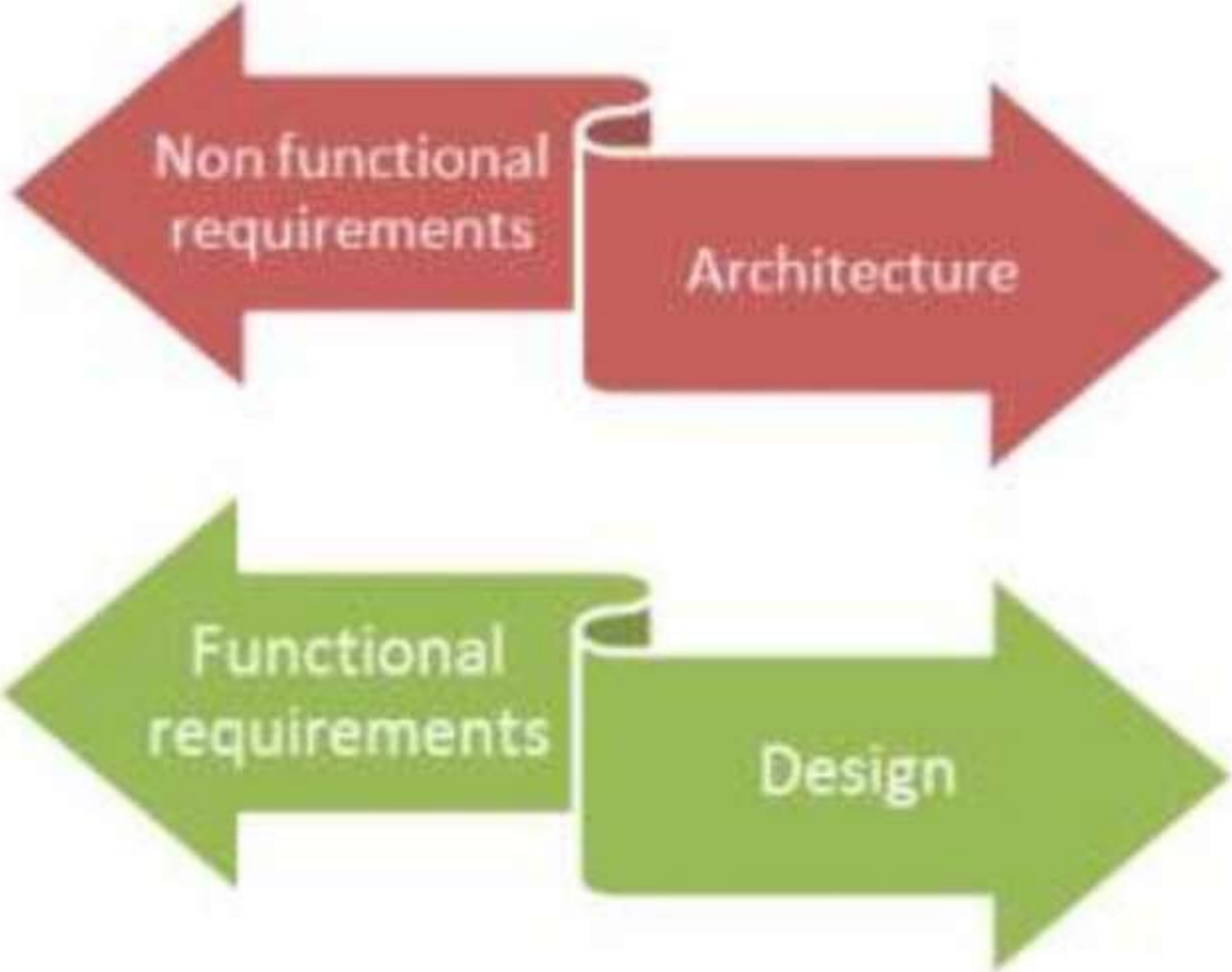


Software Architecture vs. Software Design

All architecture is design, but not all design is architecture.



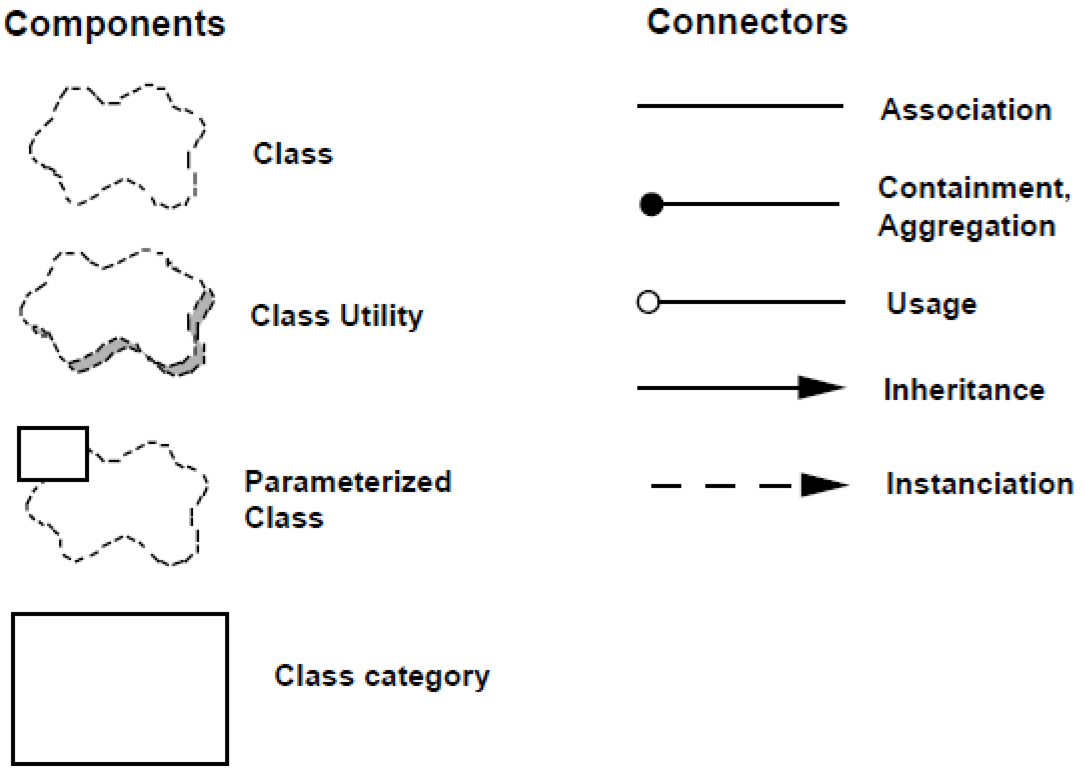
Software Architecture vs. Software Design



The Notations

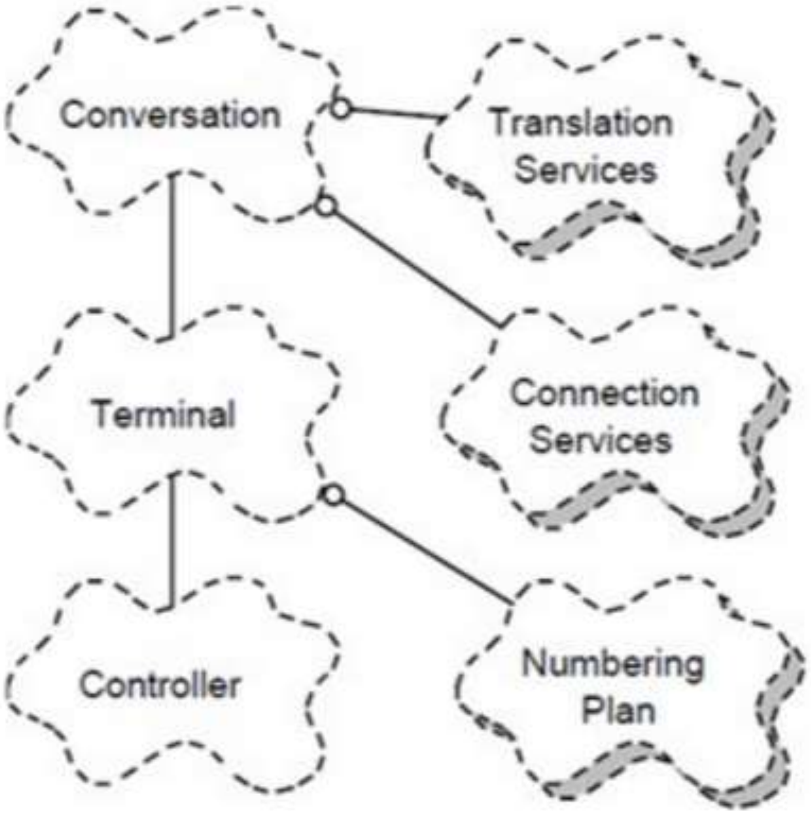
Logical View – Notations *(functional requirements)*

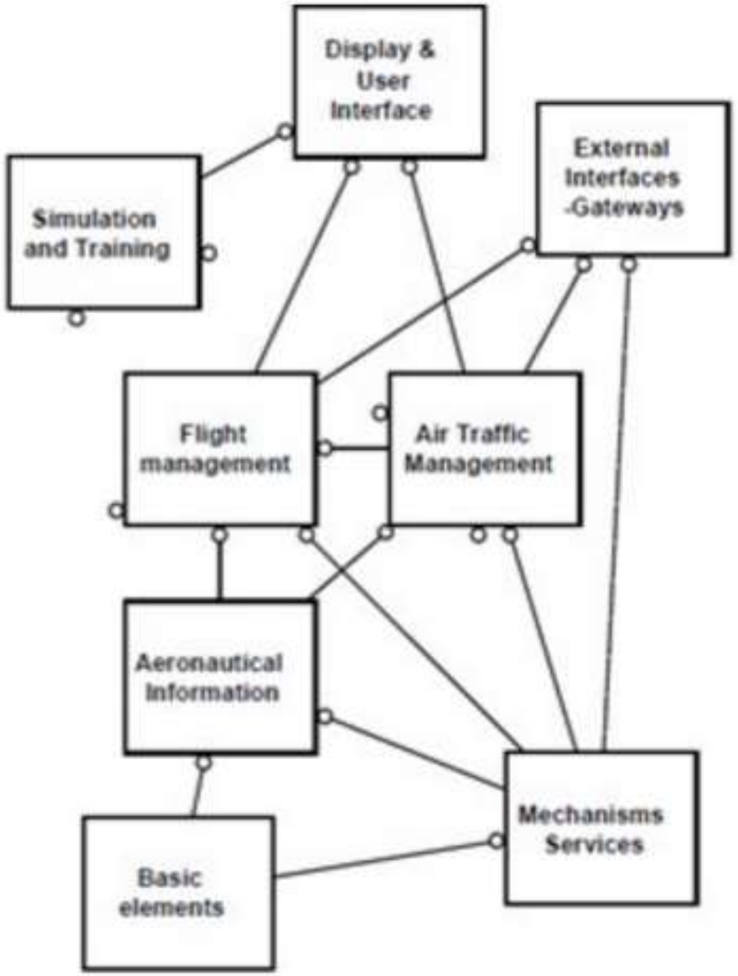
- Class diagrams and class templates are usually used to illustrate the abstraction. Common mechanisms or services are defined in *class utilities*.



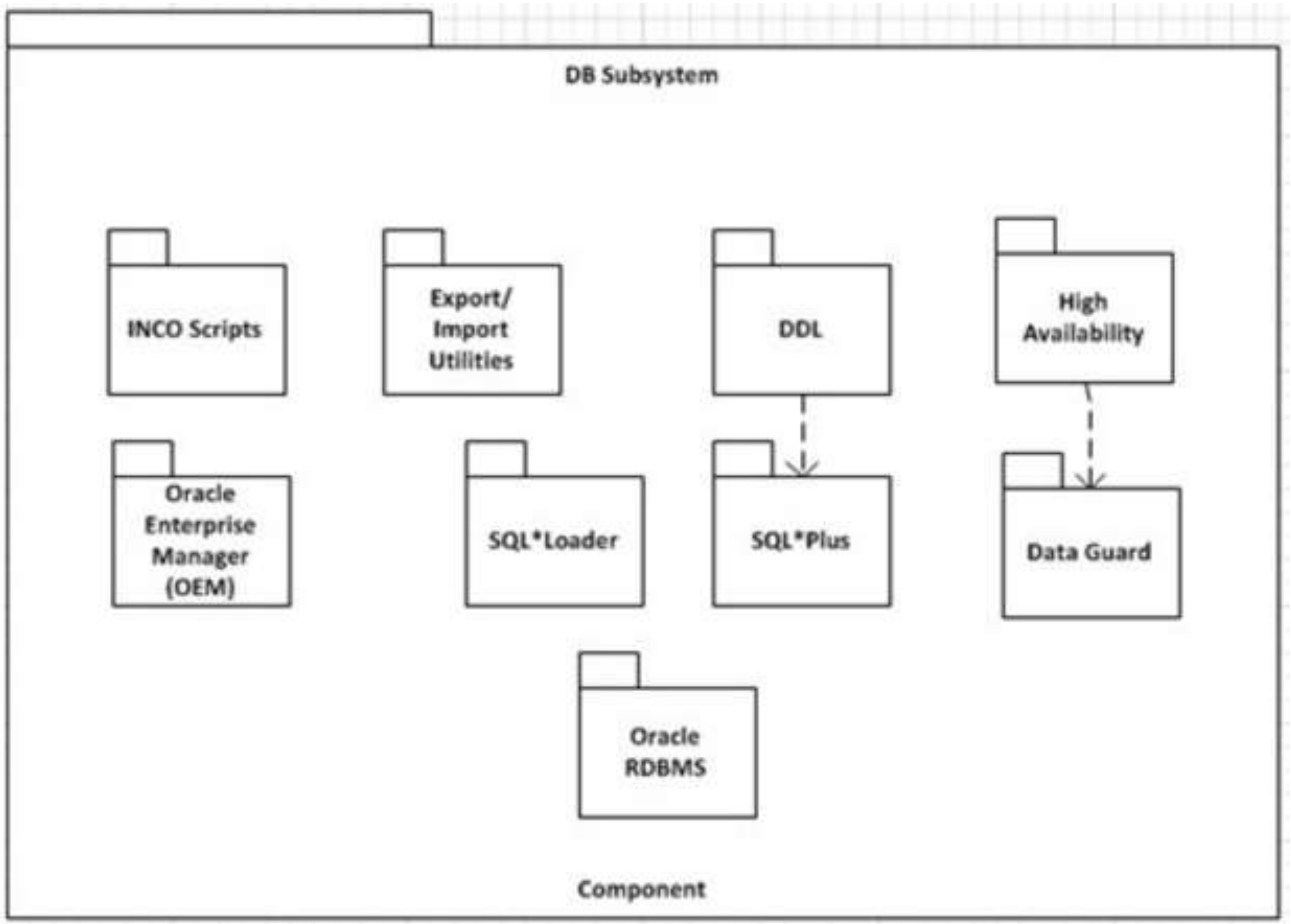
Notation for the logical blueprint

Logical View - Example



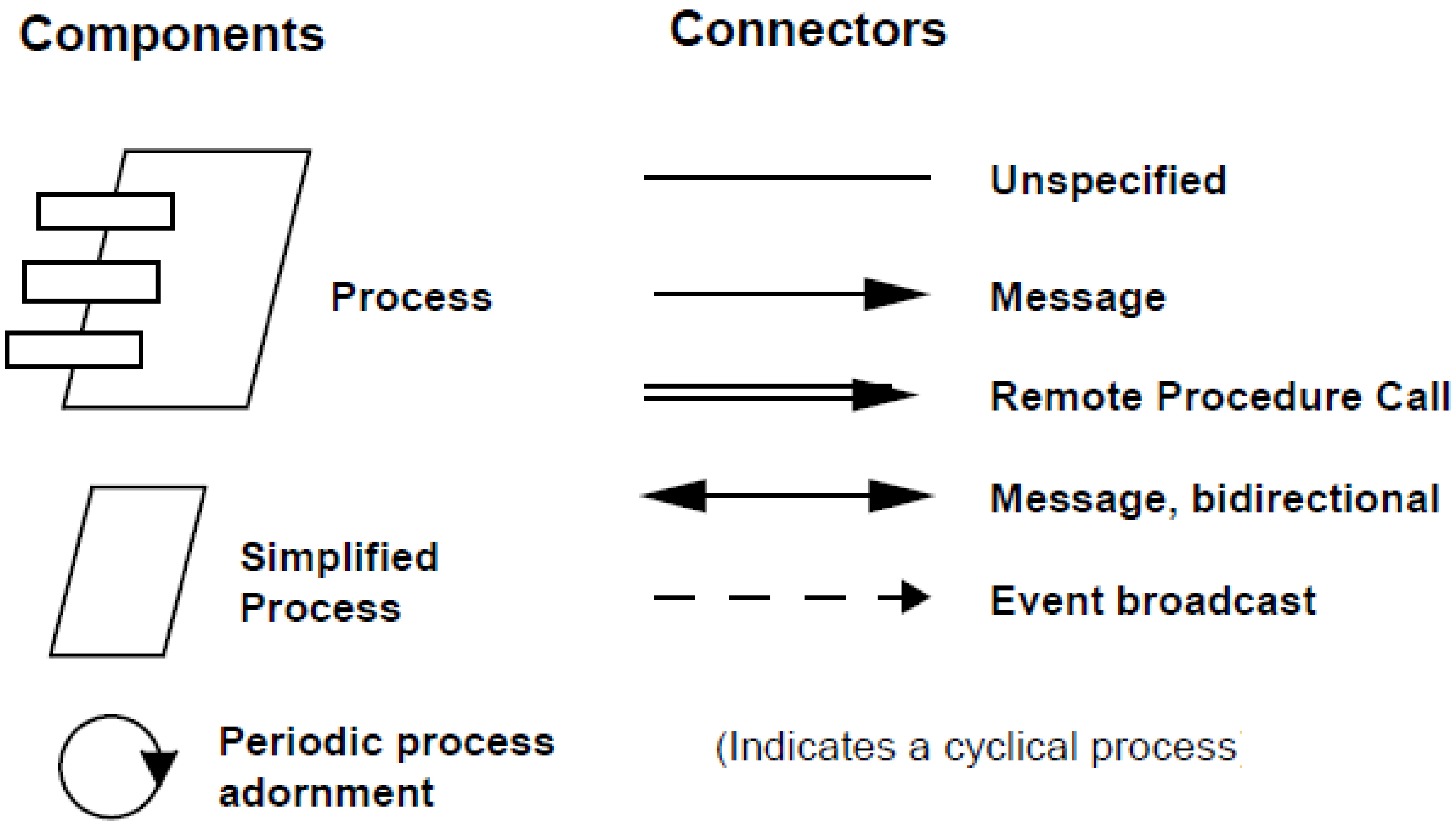


Logical View - Example



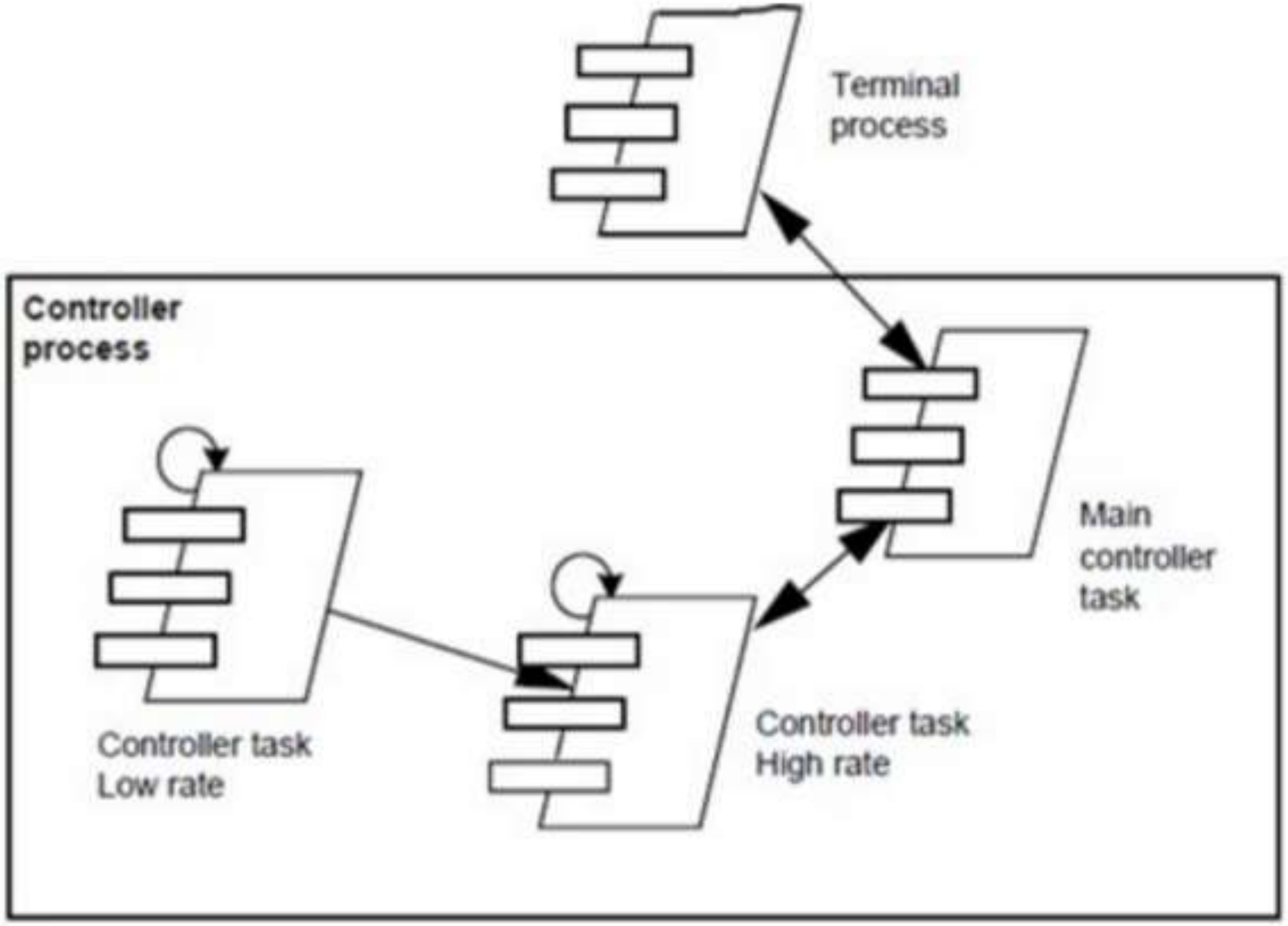
Process View – Notations *(Non-functional requirements)*

- A process is a group of tasks that form an executable unit and which can be (a) tactically controlled, (b) replicated, (c) partitioned into a set of independent tasks: major and minor (cyclic activities, buffering, time-outs).

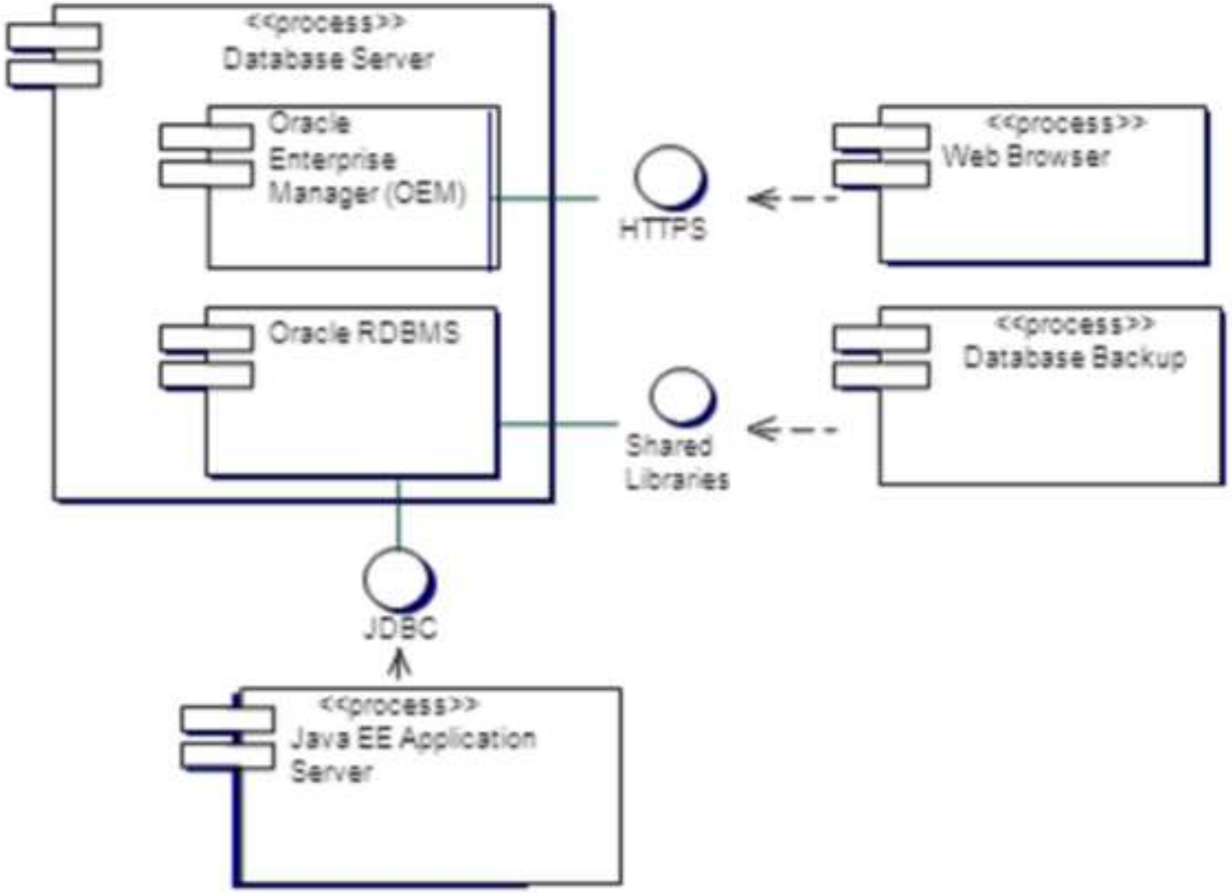


Notation for the Process blueprint

Process View - Example

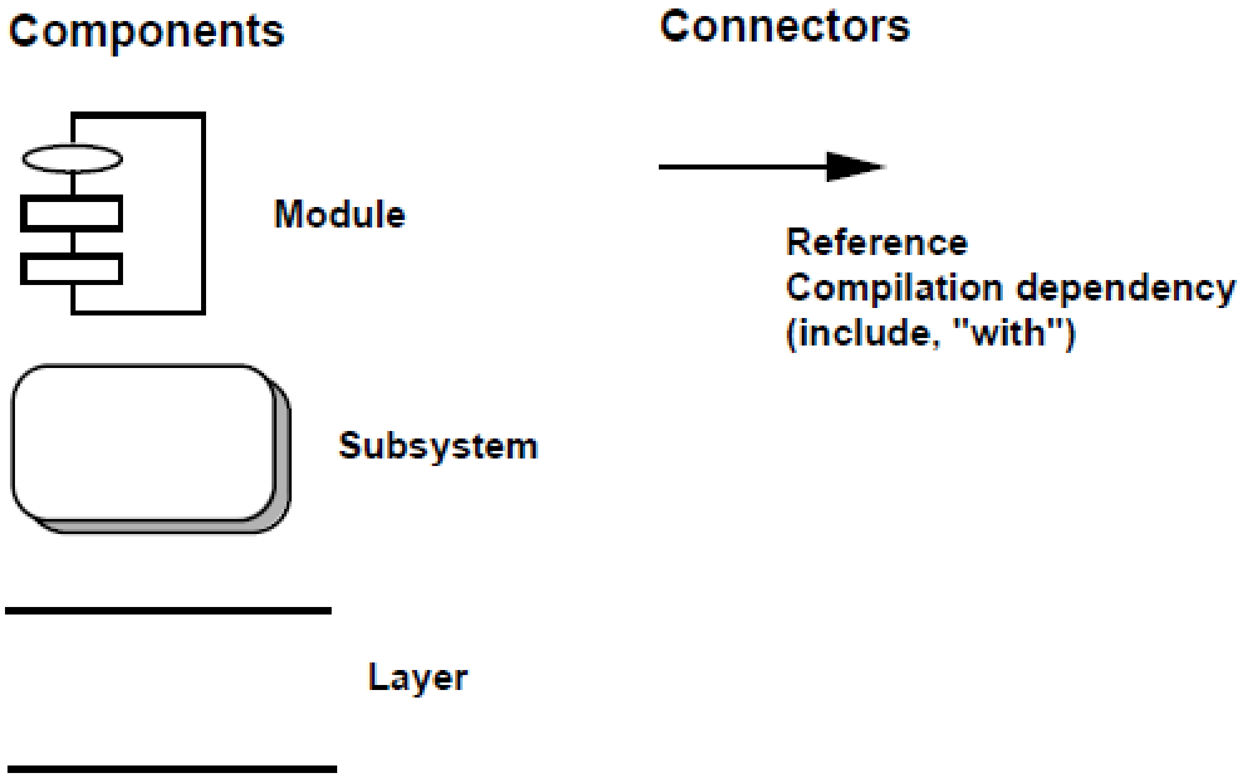


Process View - Example



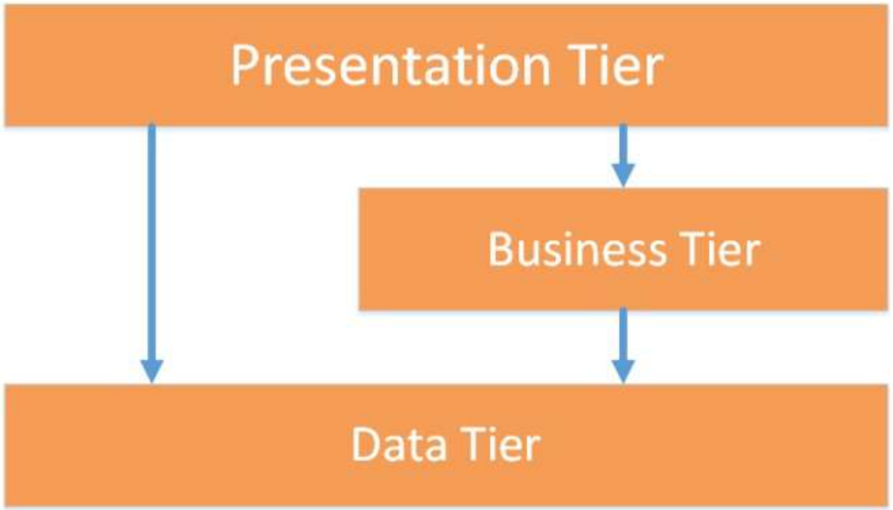
Development/Implementation View - Notation

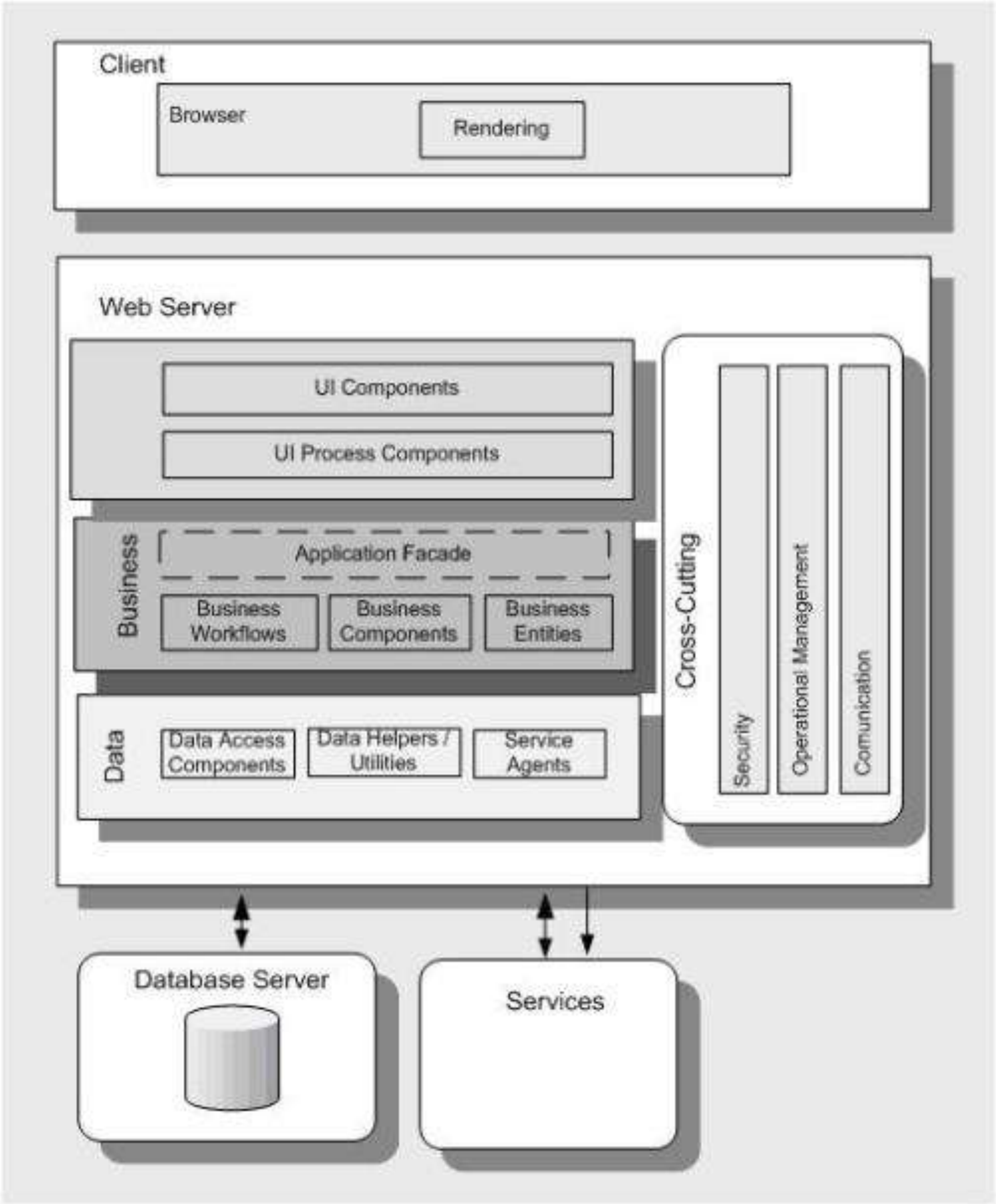
It consists of libraries and subsystems representation in dev environment. The subsystems are organized into the hierarchy of layers with well-defined interfaces.



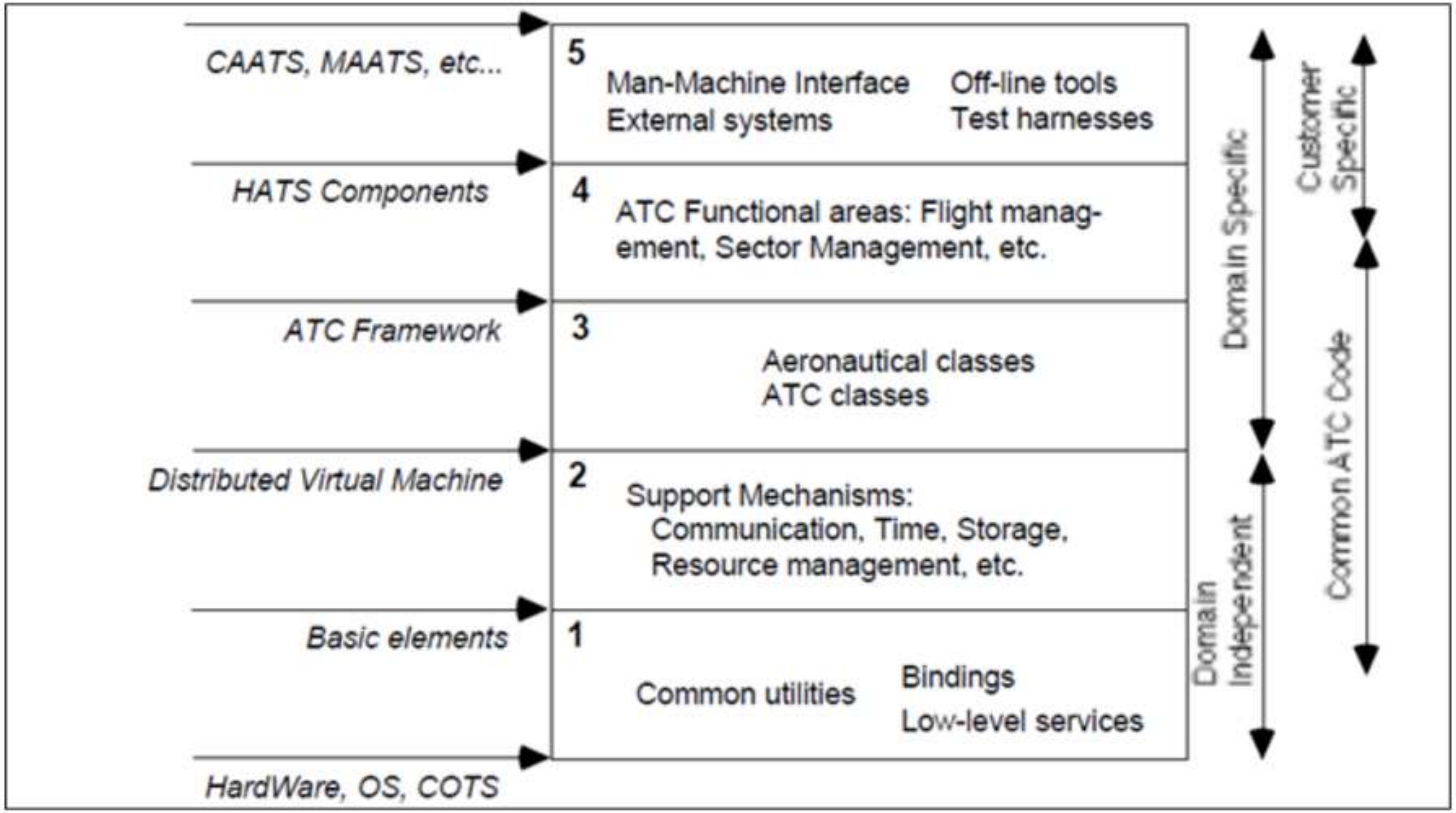
Notation for the Development blueprint

A Typical Layer Model

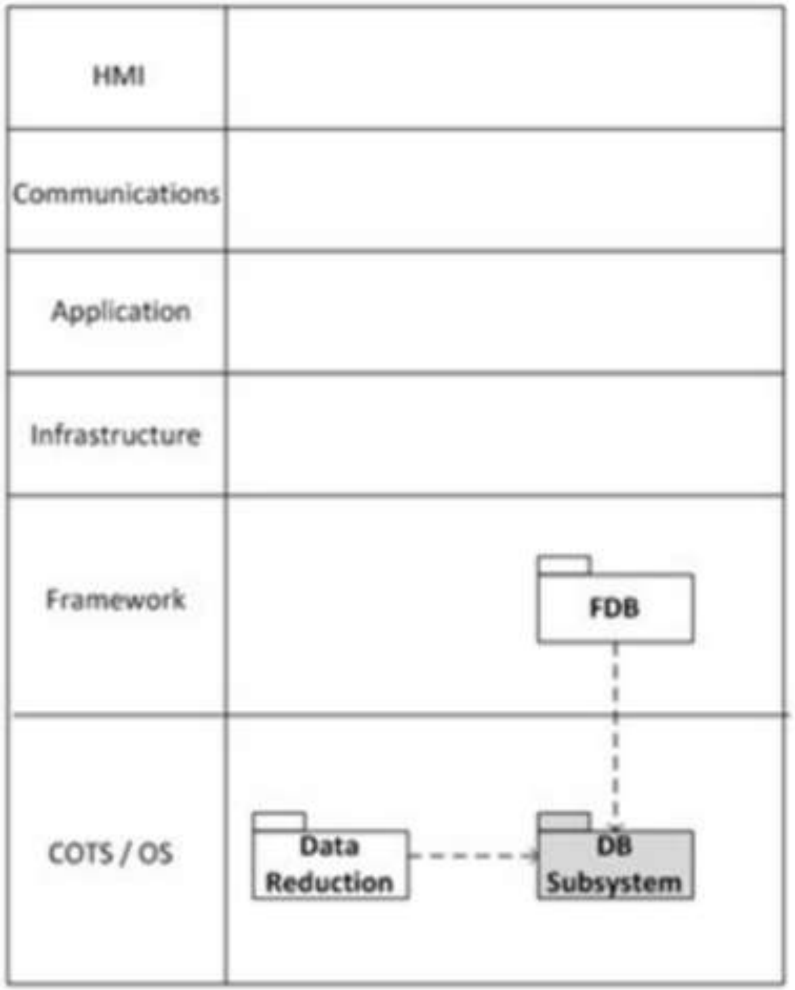




Development View / Implementation View - Example

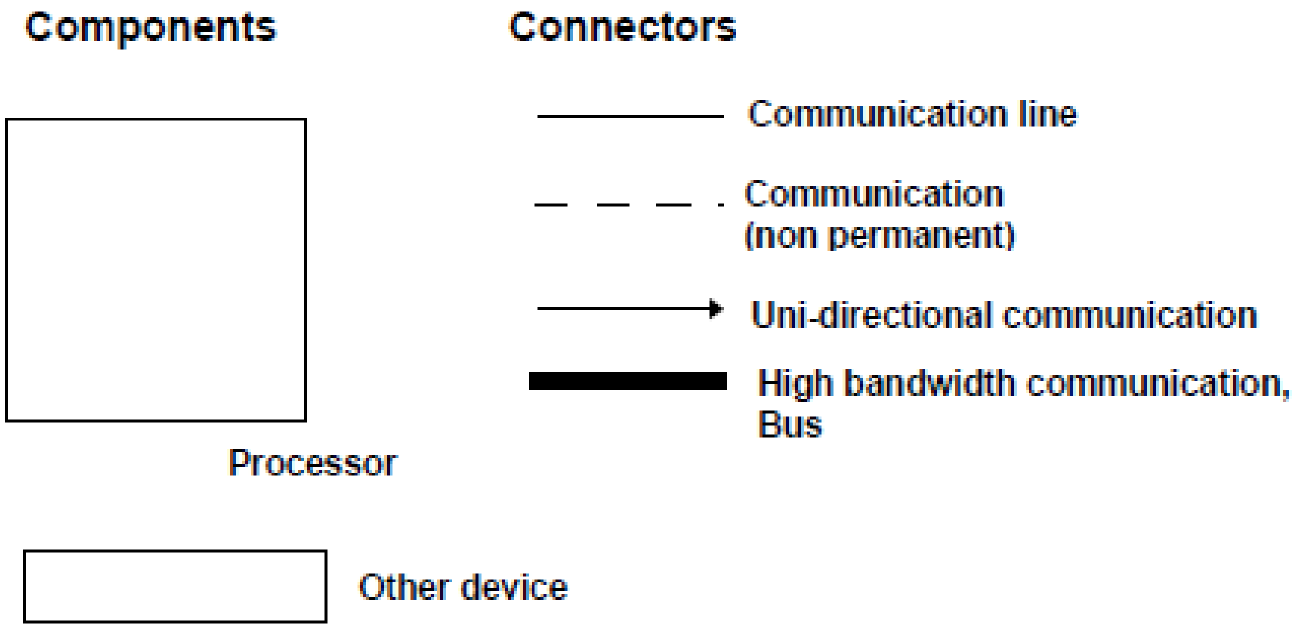


Development View / Implementation View - Example



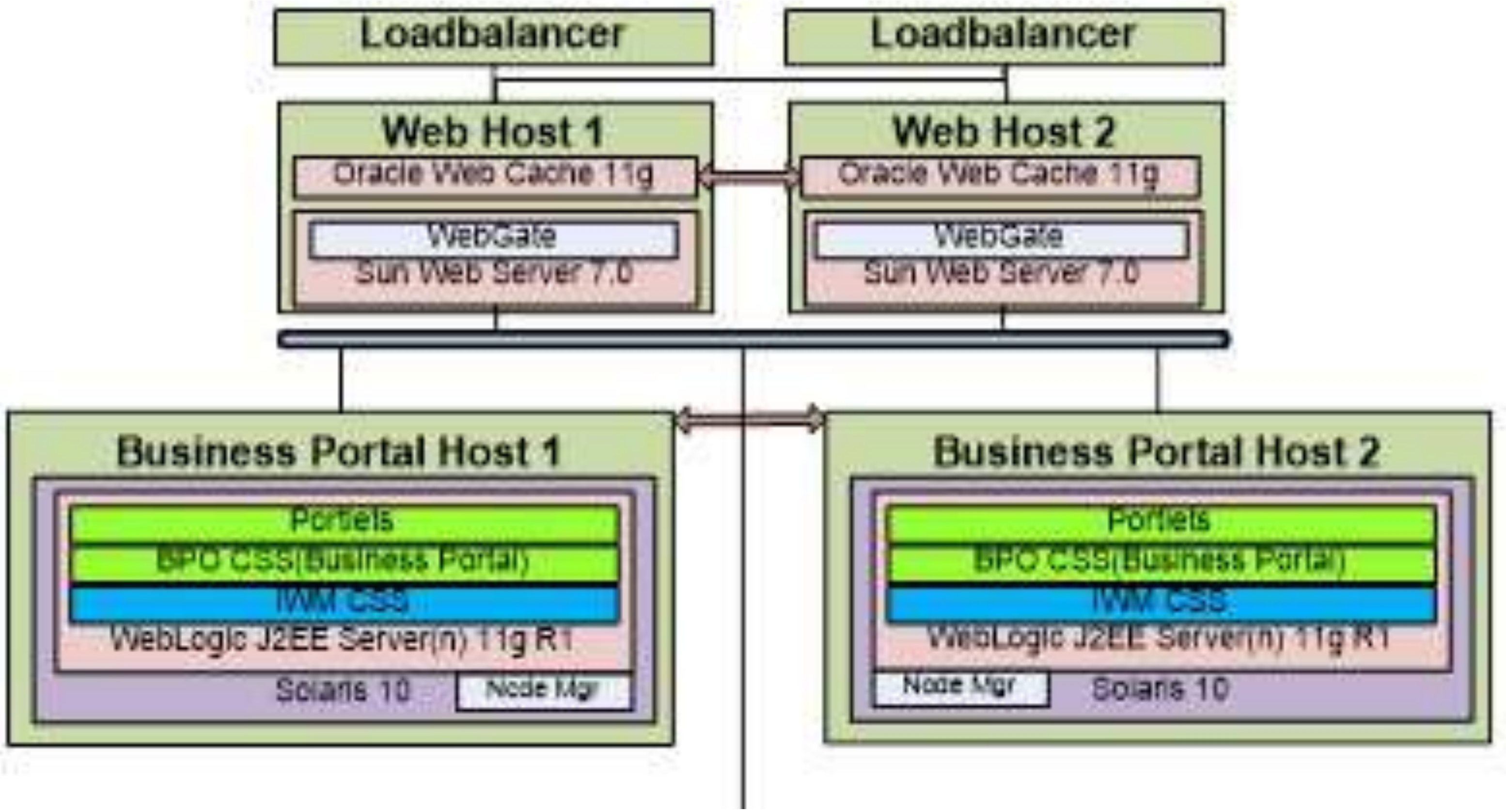
Deployment/Physical View - Notations

Represents non-functional requirements such as availability, reliability, scalability and performance. It shows how networks, processes, tasks and objects are mapped onto the various nodes.

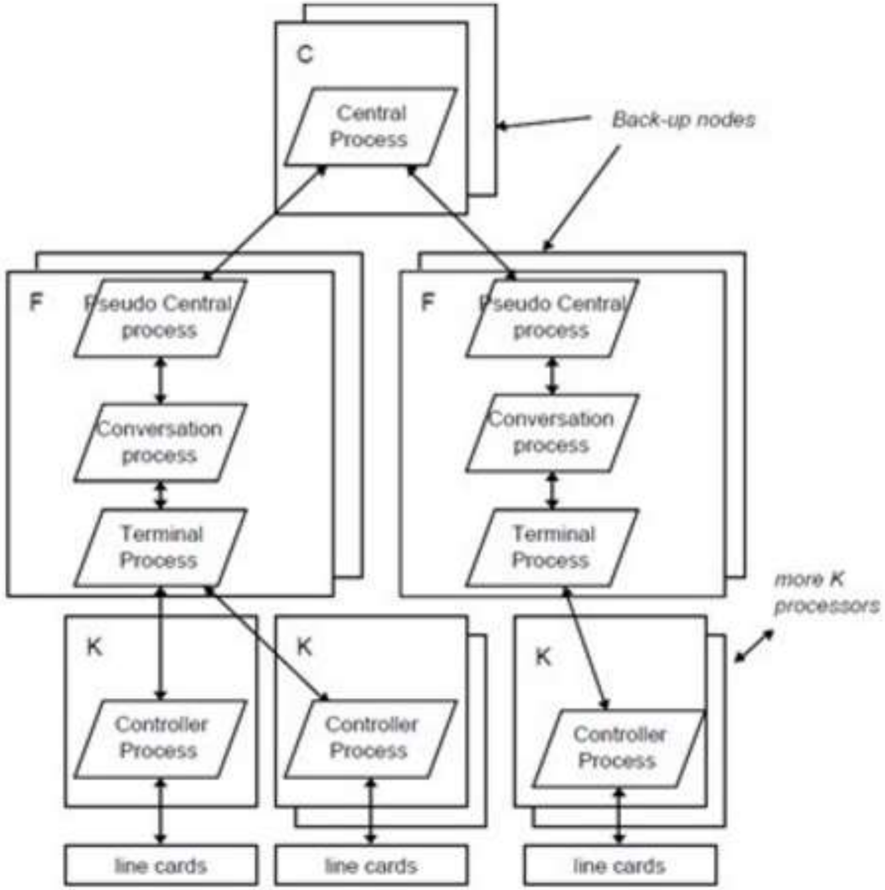

Components
Connectors
Communication line
Communication (non permanent)
Uni-directional communication
High bandwidth communication, Bus
Processor
Other device

Notation for the Physical blueprint

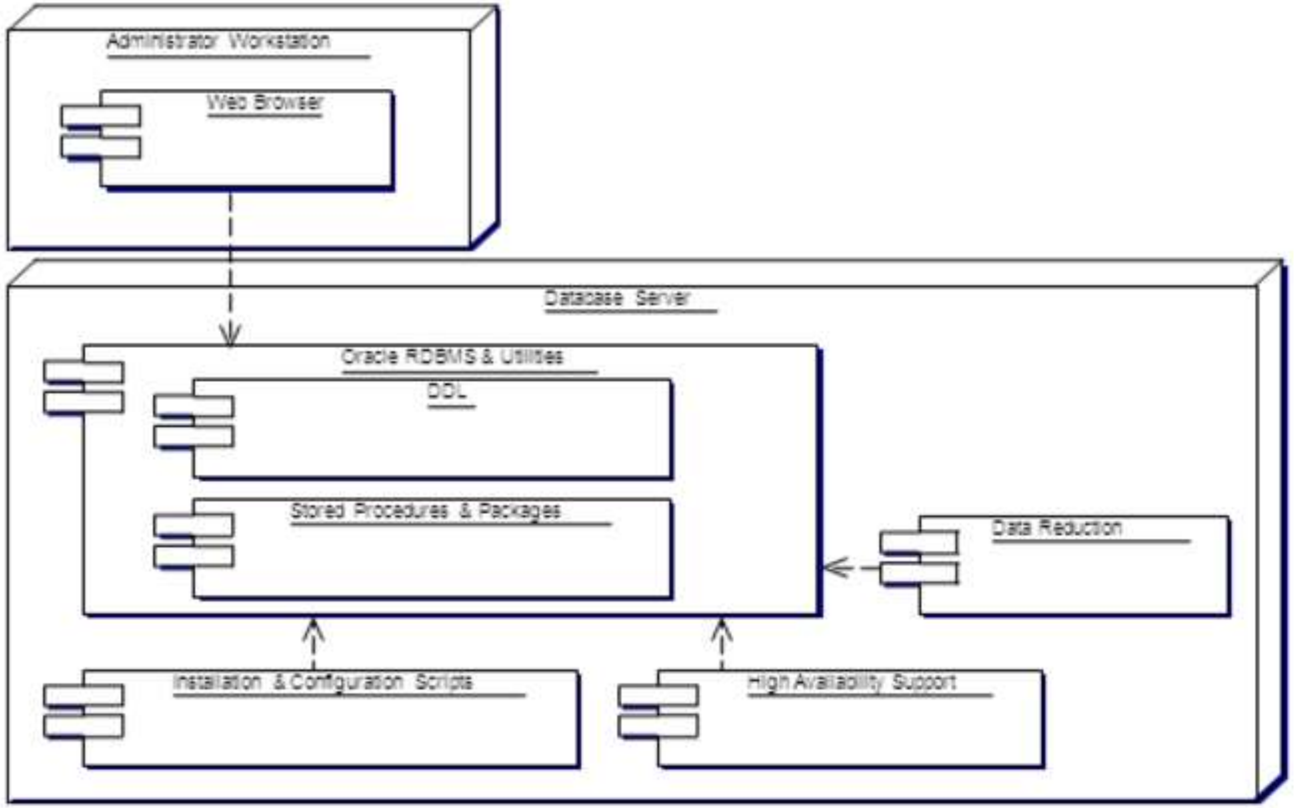
A typical Implementation Model



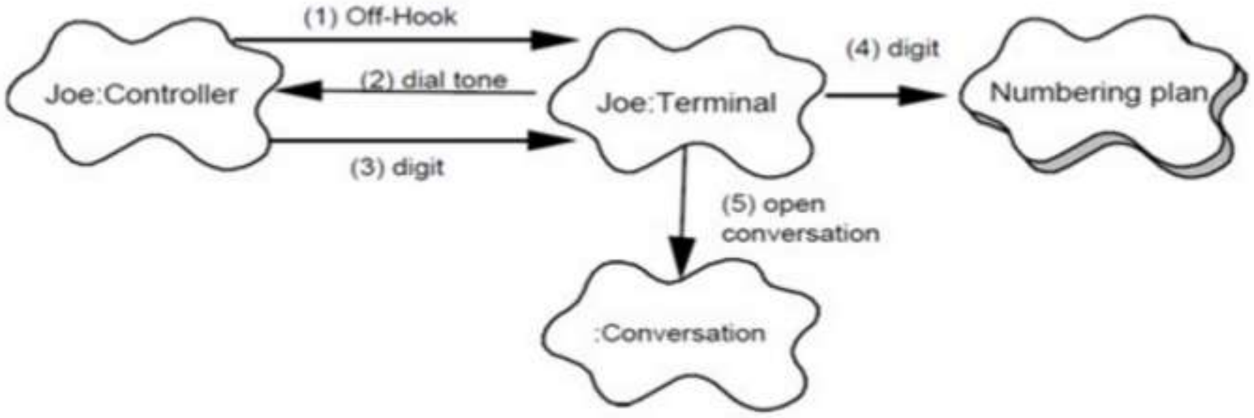
Deployment/Physical View - Example

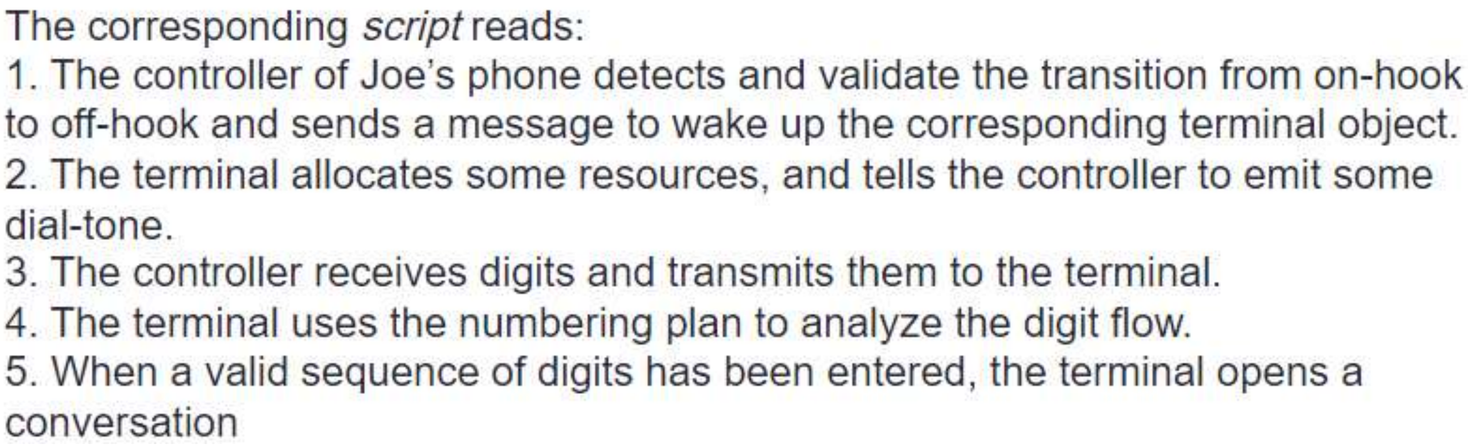


Deployment/Physical View - Example


Administrator Workstation
Web Browser
Database Server
Oracle RDBMS & Utilities
DDL
Stored Procedures & Packages
Data Reduction
Installation & Configuration Scripts
High Availability Support

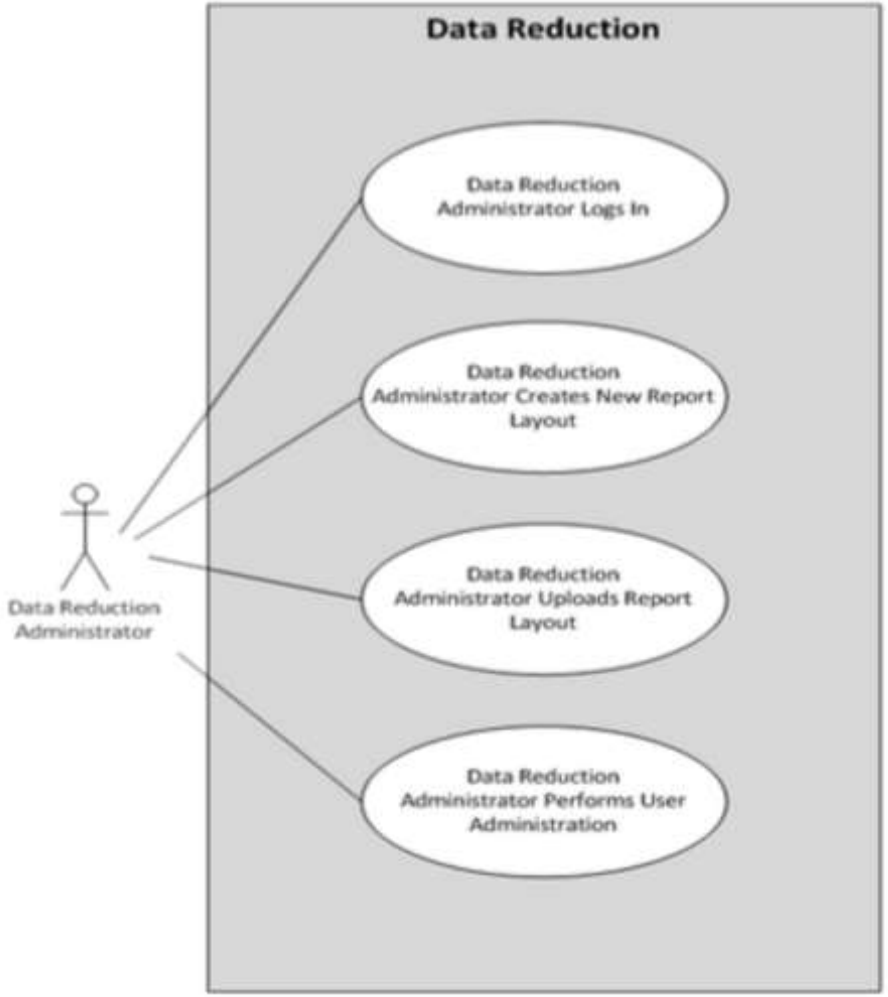
Scenarios / Use Case Model (Small PABX)



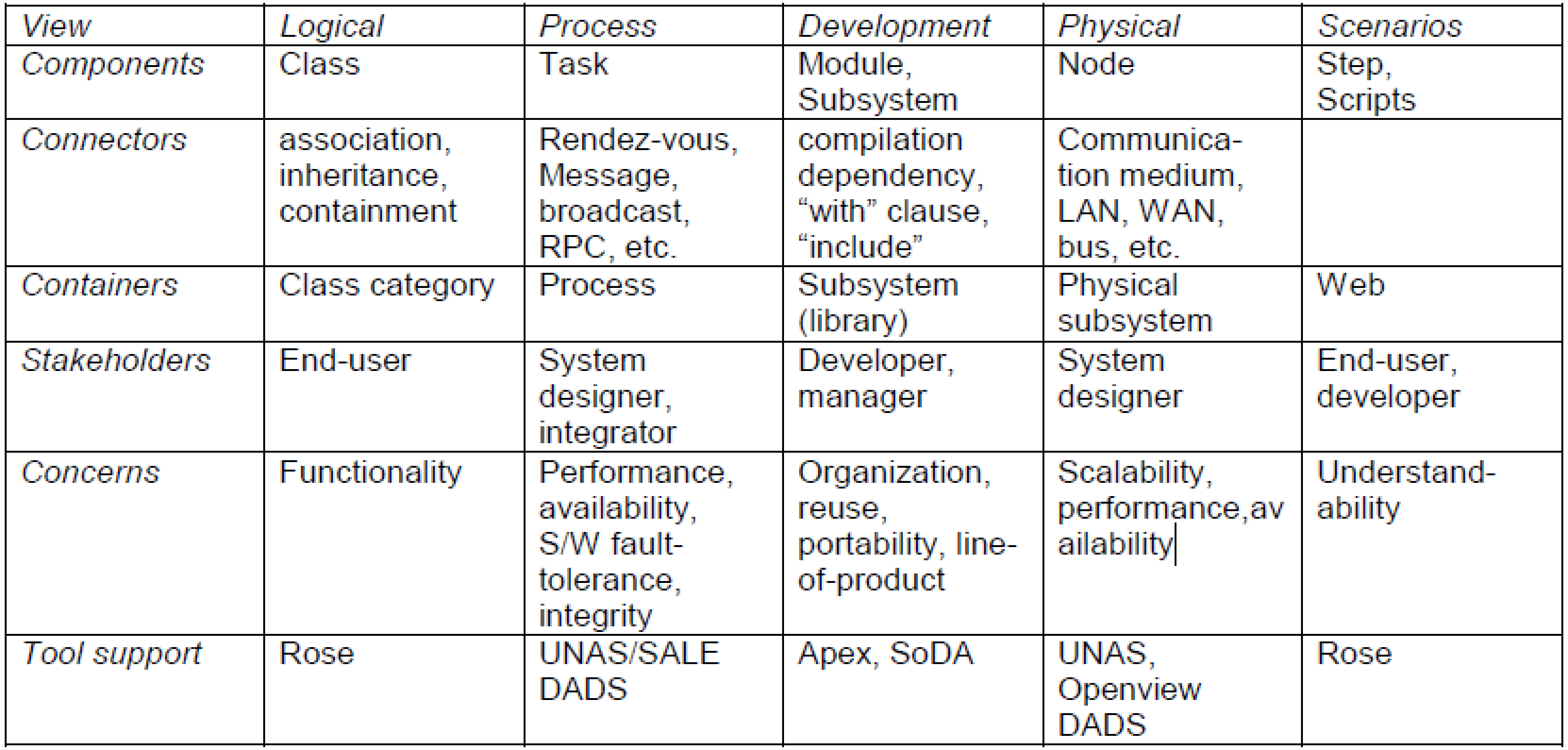


The corresponding *script* reads:
1. The controller of Joe’s phone detects and validate the transition from on-hook to off-hook and sends a message to wake up the corresponding terminal object.
2. The terminal allocates some resources, and tells the controller to emit some dial-tone.
3. The controller receives digits and transmits them to the terminal.
4. The terminal uses the numbering plan to analyze the digit flow.
5. When a valid sequence of digits has been entered, the terminal opens a conversation

Scenarios / Use Case Model (Small PABX)

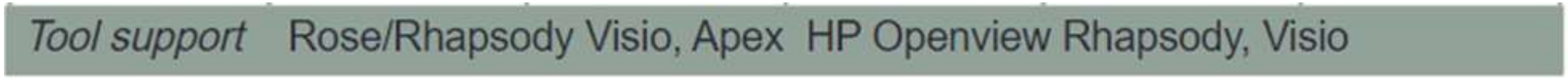

Data Reduction
Data Reduction Administrator Logs In
Data Reduction Administrator Creates New Report Layout
Data Reduction Administrator Uploads Report Layout
Data Reduction Administrator Performs User Administration
Data Reduction Administrator

Summary



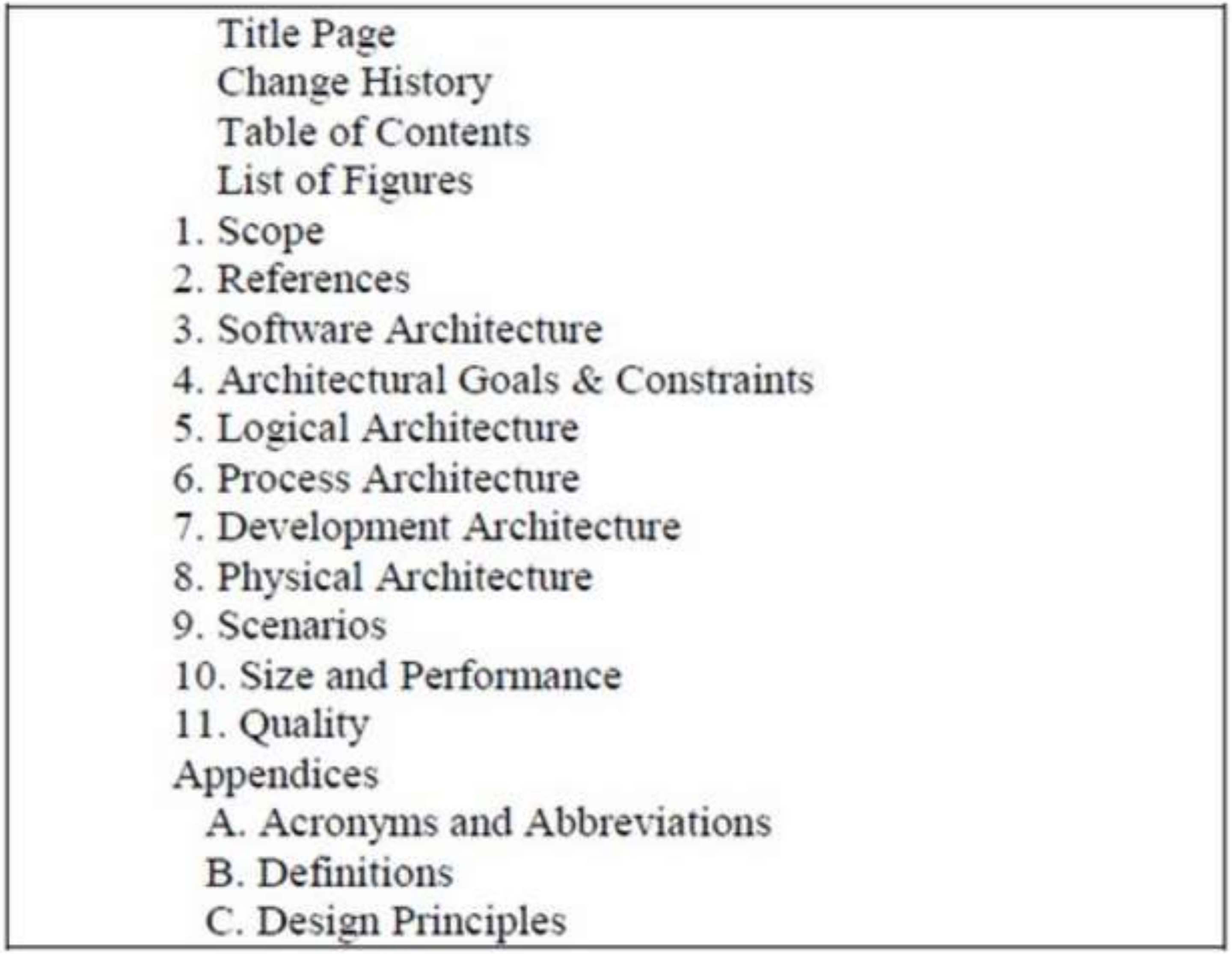
View	*Logical*	*Process*	*Development*	*Physical*	*Scenarios*
Components	Class	Task	Module, Subsystem	Node	Step, Scripts
Connectors	association, inheritance, containment	Rendez-vous, Message, broadcast, RPC, etc.	compilation dependency, “with” clause, “include”	Communication medium, LAN, WAN, bus, etc.	
Containers	Class category	Process	Subsystem (library)	Physical subsystem	Web
Stakeholders	End-user	System designer, integrator	Developer, manager	System designer	End-user, developer
Concerns	Functionality	Performance, availability, S/W fault-tolerance, integrity	Organization, reuse, portability, line-of-product	Scalability, performance,availability	Understandability
Tool support	Rose	UNAS/SALE DADS	Apex, SoDA	UNAS, Openview DADS	Rose

Summary of the “4+1” view model



Tool support	Rose/Rhapsody	Visio, Apex	HP Openview	Rhapsody, Visio

SW Architecture Document - a typical outline



Title Page
Change History
Table of Contents
List of Figures

1. Scope
2. References
3. Software Architecture
4. Architectural Goals & Constraints
5. Logical Architecture
6. Process Architecture
7. Development Architecture
8. Physical Architecture
9. Scenarios
10. Size and Performance
11. Quality

Appendices

A. Acronyms and Abbreviations
B. Definitions
C. Design Principles

The Architecture must be Documented

Architectural Document should include (my version):

- Architectural goals & constraints
- Software Architectural Views:
  - Logical Views (representing user functionalities)
  - Process views (representing system execution)
  - Development Views (representing implementation breakdown)
  - Physical Views (representing deployment/hardware assignments)
  - Data view (representing the key files and tables)
- Scenarios
- Rationale
- Software Architecture (combined all the views and rationale) = {elements, forms, rationale}

This author includes more such as: change history, scope, references, performance & size, quality, etc.

Enterprise Architecture

- If Solution Architecture is analogy to Building Blueprint, then Enterprise Architecture will be analogy to City Blueprint.
- The scope of Enterprise Architecture covers the entire enterprise rather than just 1 business unit.

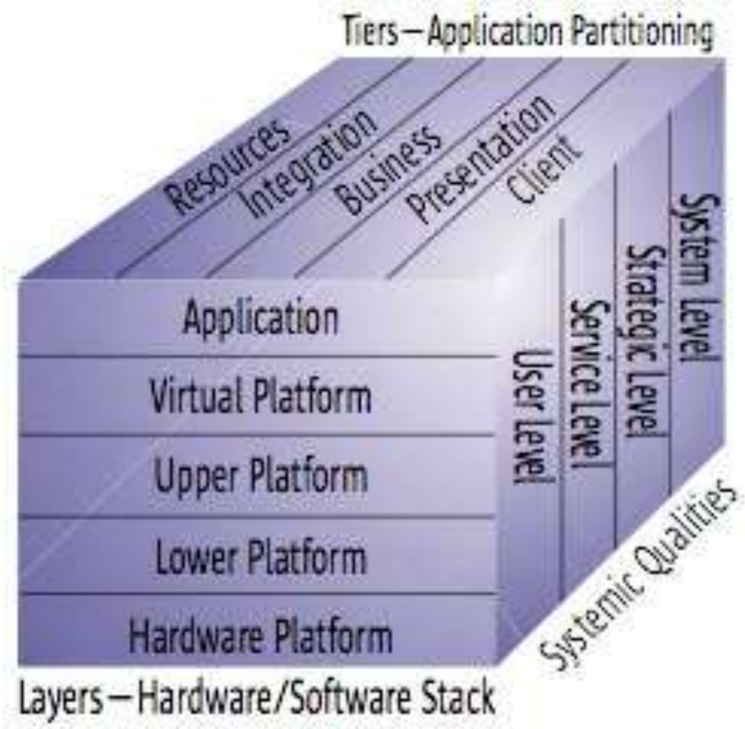
Definition of Enterprise Architecture

- The MIT Center for Information Systems Research (MIT CISR) in 2007 defined enterprise architecture as:

“Enterprise architecture is the organizing logic for business processes and IT infrastructure reflecting the integration and standardization requirements of the company’s operating model. The operating model is the desired state of business process integration and business process standardization for delivering goods and services to customers”.

Sun Tone Cube Architecture (Partitioning)

- The Sun Tone cubic shows a 3 dimensional cube that represent 3 important aspect of technology architecture.



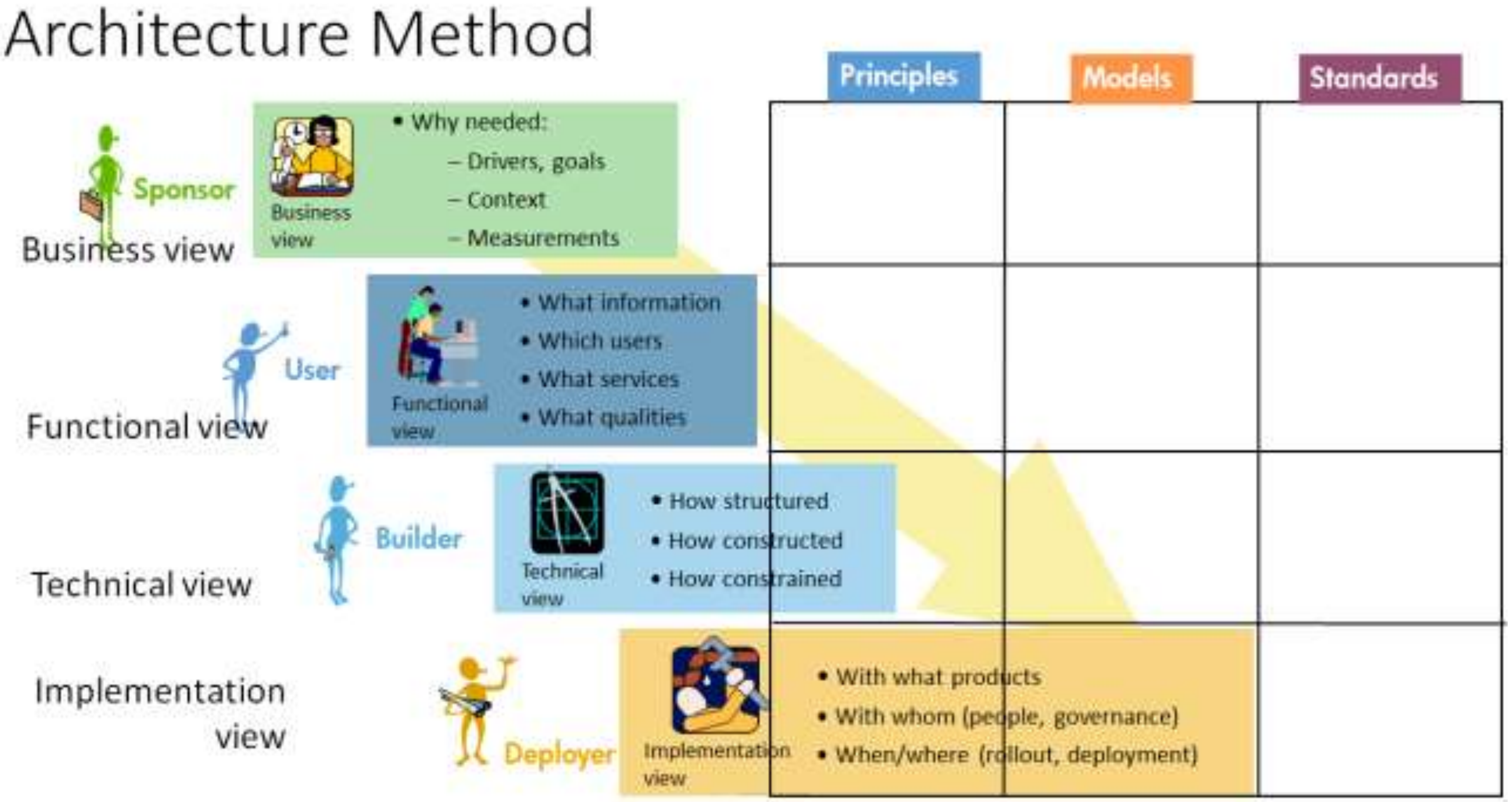
- Tiers: Represent the separation of concern
- Layers: Hardware and Software Stack
- Systemic Qualities: cross functional

requirements of the IT System.

Still - One Size Does Not Fit All

- The architect community hasn’t agreed on a “Unified Architecture Method”.

HP Global Method for ITSA



Architecture Method

Principles

Models

Standards

Sponsor

Business view

Business view

- Why needed:
  - Drivers, goals
  - Context
  - Measurements

User

Functional view

Functional view

- What information
- Which users
- What services
- What qualities

Builder

Technical view

Technical view

- How structured
- How constructed
- How constrained

Implementation view

Deployer

Implementation view

- With what products
- With whom (people, governance)
- When/where (rollout, deployment)

This architecture method comprises of the following views:

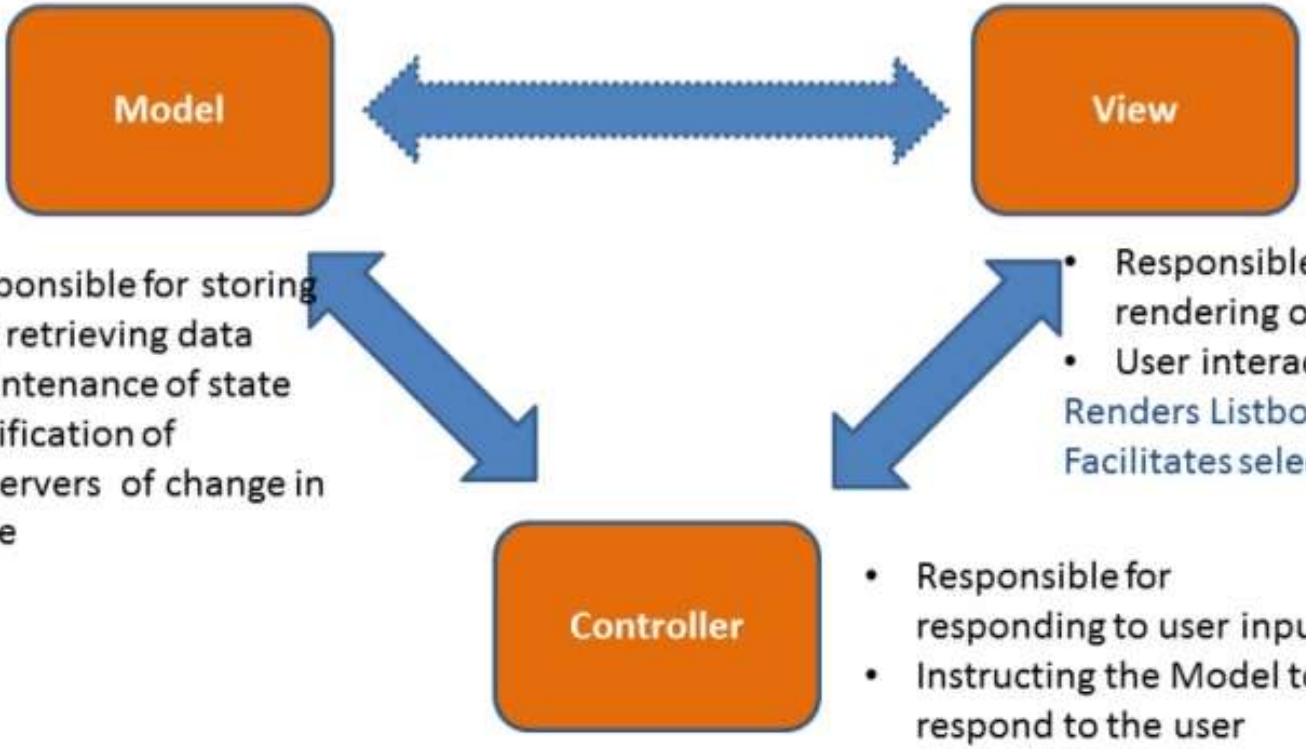
- ***Business view*** : The business view gives the business sponsor’s prospective on ‘why we are doing it?’ The drivers, goals, context and measurements.
- ***Functional view*** : The functional view illustrates a system from a user’s perspective. It gives answers to the ‘What’ questions: What is the information, for which users, what are the services and at what qualities.
- ***Technical view*** : The application view illustrates a system from a builder’s perspective and answer the ‘How’ questions: how the system will be structured, how it will be constructed, how are the constraint handled?
- ***Implementation view*** : The infrastructure view depicts the system from a Deployer’s point-of-view. It answers the ‘With’ questions: With what product, with whom (people, governance), When/Where (rollout, deployment).

Each view defines Architecture Principle, Architecture Model and Architecture Standards.

- 1. Architecture Principle: Principles are general rules and guidelines, intended to be enduring and seldom amended, that inform the way in which an organization sets about fulfilling its mission. For example, if a person's *principle* is to "have integrity". That will guide his actions and decisions; He would most likely have a behavior *standard* not to steal. Therefore architecture principle provides the guidance to architecture standards that is actionable. (Standard may be more likely subjected to changed than Principle). The format for Defining Principles is:
- Name: Principle Name
- Statement: Unambiguous statement that communicate the fundamental rule.
- Rationale: Describe why the principle is established.
- Implications: Describe the requirements, both business and IT, for carrying out the principle.
- 2. Model: Diagram that describe the architecture view.
- 3. Architecture Standard: Standards are rules for how to define the structure throughout the architecture. Taking the same example on 'Integrity' principle, the standard may be "When I found a wallet that does not belongs to me, I will return it to the police station." Notice that the standard is actionable and design directly take reference to this standard and must comply with the architecture standard.

A good architecture method should give guidance on how to address the coherence between architecture views. The HP Global Method achieves views coherence through Architecture Principles. Architect first defines the Business Architecture Principle, then derives the Functional Architecture Principle and followed by Technical and Implementation Architecture Principles. These principles once finalized, forms the guiding principles for the definitions of model and standard in each of its respective view.

Model View Controller (MVC) Arch Pattern



- Responsible for storing and retrieving data
- Maintenance of state
- Notification of Observers of change in state

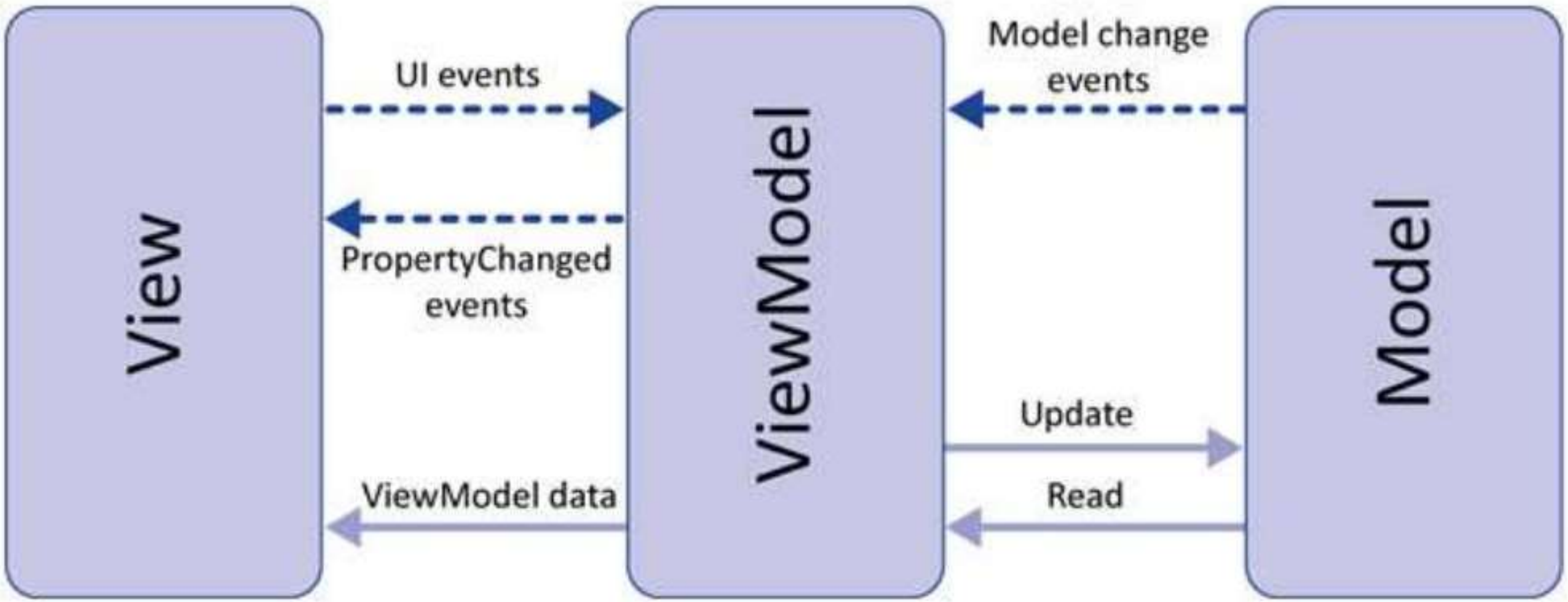
- Responsible for rendering of model - UI
- User interaction

Renders Listbox Control, Facilitates selection

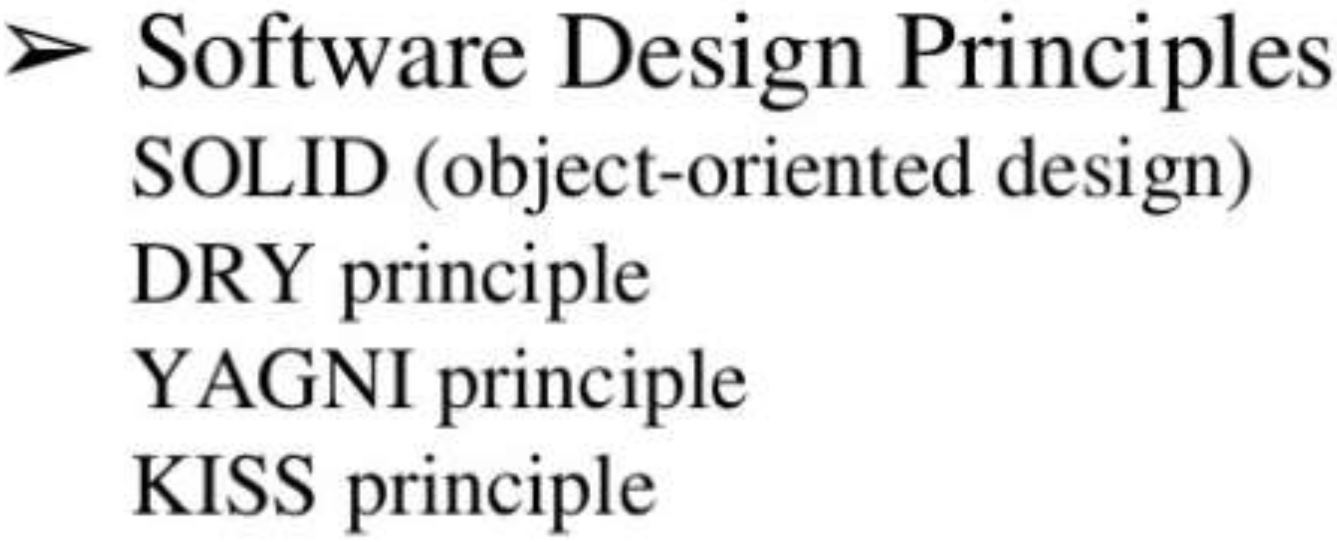
- Responsible for responding to user input
- Instructing the Model to respond to the user input

Handles Listbox Selection event, sends information to Model

Model View ViewModel



Software Design Principles



➢ Software Design Principles

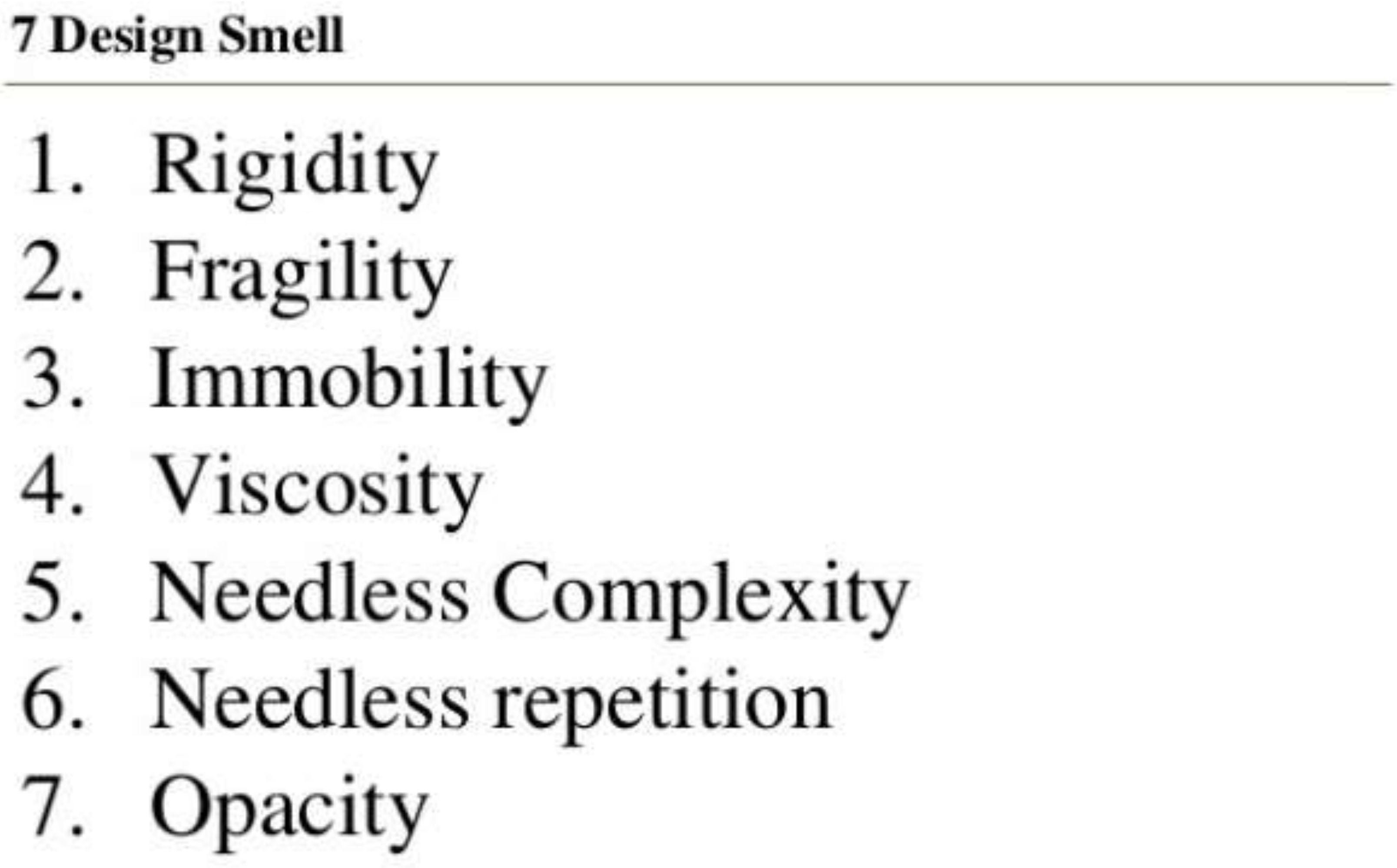
SOLID (object-oriented design)

DRY principle

YAGNI principle

KISS principle

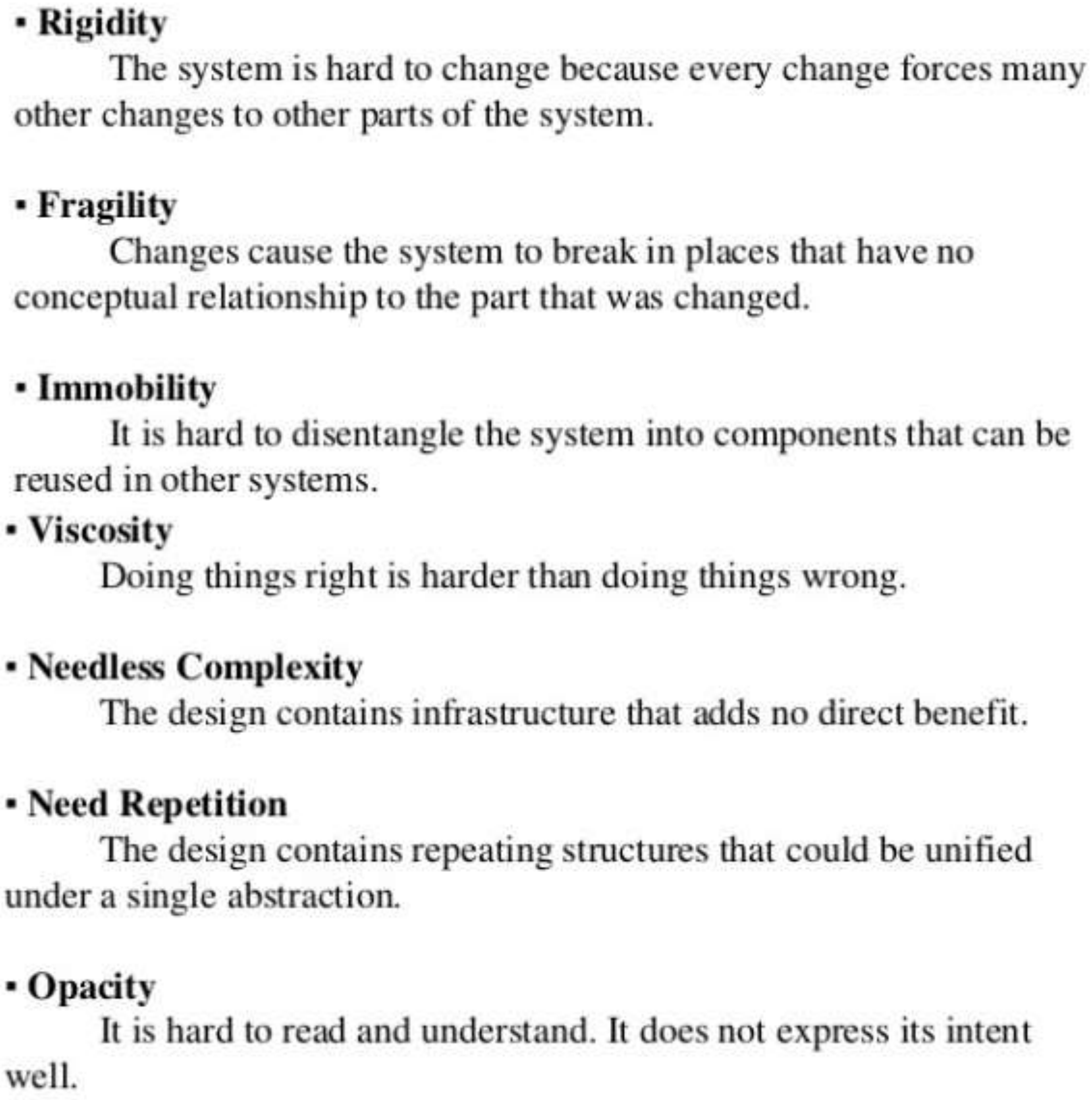
Design Smells



7 Design Smell

1. Rigidity
2. Fragility
3. Immobility
4. Viscosity
5. Needless Complexity
6. Needless repetition
7. Opacity

7 Design Smells



- **Rigidity**

  The system is hard to change because every change forces many other changes to other parts of the system.

- **Fragility**

  Changes cause the system to break in places that have no conceptual relationship to the part that was changed.

- **Immobility**

  It is hard to disentangle the system into components that can be reused in other systems.

- **Viscosity**

  Doing things right is harder than doing things wrong.

- **Needless Complexity**

  The design contains infrastructure that adds no direct benefit.

- **Need Repetition**

  The design contains repeating structures that could be unified under a single abstraction.

- **Opacity**

  It is hard to read and understand. It does not express its intent well.

The End