2IW80 Software specification and architecture

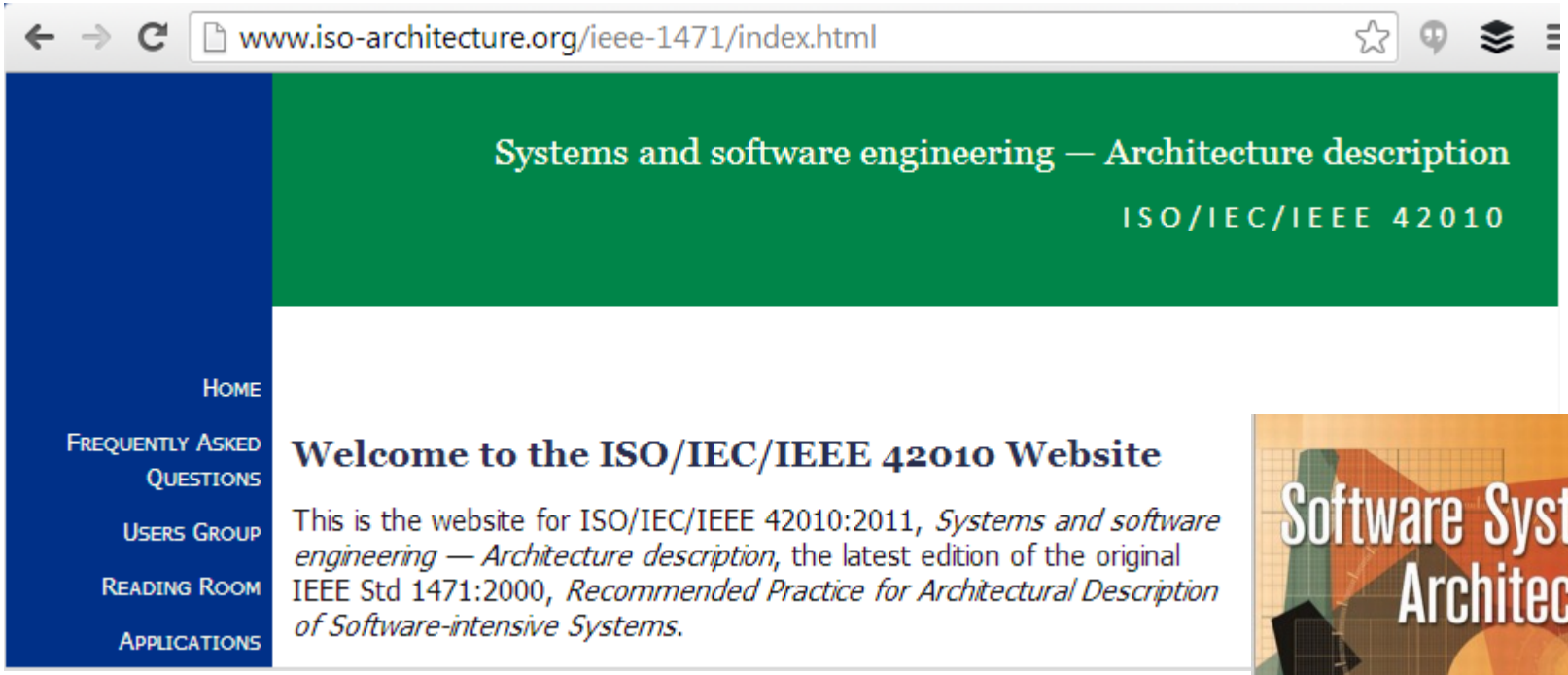
# Software architecture: Introduction

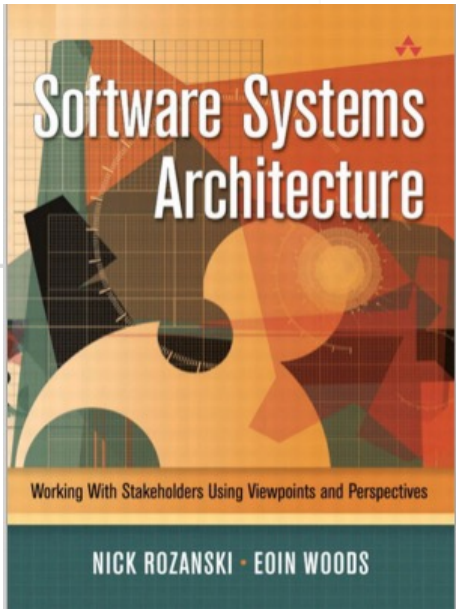
Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

# This week sources




Slides by Johan Lukkien and Rudolf Mak

# Software architecture

- Software architecture is the **fundamental organization** of a system embodied in
  - its **elements**,
  - **relationships**,
  - and in the principles of its **design** and **evolution**.

*[IEEE Std. 42010-2011]*

# “Fundamental organization”

- Less change-prone:
  - “Architecture of software is a collection of design decisions that are expensive to change.”

    Alexander Ran, Nokia Research

    European Conf on Software Engineering, 2001

- Not all aspects of system organization are fundamental
  - Hence, not all design decisions impact a system’s architecture
- How one defines “fundamental” depends on the **stakeholders’** perspectives on the system goals

# Stakeholders?

- **Stakeholder** is a person, group or organization with an interest in a project
  - we focus on the **roles** of stakeholders not their identities.
  - the same person can hold multiple roles.

# Stakeholders?

- **Stakeholder** is a person, group or organization with an interest in a project
  - we focus on the **roles** of stakeholders not their identities.
  - the same person can hold multiple roles.

| Class | Description [Rozanski Woods] |
|---|---|
| Acquirers | Oversee the procurement of the system |
| Assessors | Oversee conformance to standards and legal regulations |
| Communicators | Explain the system via documentation and training |
| Developers | Construct and deploy the system from its specifications |
| Maintainers | Manage the evolution of the system once operational |
| Suppliers | Provide hardware/software platform on which the system will run |
| Support staff | Assist users to make use of the running system |
| System administrators | Run the system once deployed |
| Testers | Check whether the system meets its specifications |
| Users | Define the system's functionality and use it once running |

# Example [Rozanski, Woods]

Company A, a manufacturer of computer hardware, wants an enterprise resource planning (ERP) system to better manage all aspects of its supply chain from ordering through delivery. Management anticipates the system being constructed from a custom off-the-shelf (COTS) package combined with some in-house development for specialized aspects of functionality. The new system must be deployed within one year, in time for a meeting where shareholders will consider future funding for the company.

- **Identify the stakeholders in this story.**

Company A, a manufacturer of computer hardware, wants an enterprise resource planning system to better manage all aspects of its supply chain from ordering through delivery. Management anticipates the system being constructed from a custom off-the-shelf package combined with some in-house development for specialized aspects of functionality. The new system must be deployed within one year, in time for a meeting where shareholders will consider future funding for the company.

| Class | Description [Rozanski Woods] |
|---|---|
| Acquirers | Oversee the procurement of the system |
| Assessors | Oversee conformance to standards and legal regulations |
| Communicators | Explain the system via documentation and training |
| Developers | Construct and deploy the system from its specifications |
| Maintainers | Manage the evolution of the system once operational |
| Suppliers | Provide hw/sw on which the system will run |
| Support staff | Assist users to make use of the running system |
| Sys. admins | Run the system once deployed |
| Testers | Check whether the system meets its specifications |
| Users | Define the system's functionality and use it once running |

# Solution

- **Acquirers**: the business sponsor (senior management), responsible for funding; the purchasing department and representatives of IT, who will evaluate a number of potential ERP packages.
- **Users**: internal staff, including those who work in order entry, purchasing, finance, manufacturing, and distribution.
- **Developers**, **system administrators**, and **maintainers:** the internal IT department
- **Assessors**: internal acceptance test team.
- **Communicators**: internal trainers.
- **Support**: an internal help desk (possibly in conjunction with the COTS supplier).

# Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**

# Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint** a way of looking at a system from the position of a certain *stakeholder* with a particular *concern*
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques

# Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint** a way of looking at a system from the position of a certain *stakeholder* with a particular *concern*
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques
  - **View**: whatever you see in a system from a particular viewpoint
    - collection of system models
    - conforming to the viewpoint

# Stakeholders, goals and concerns

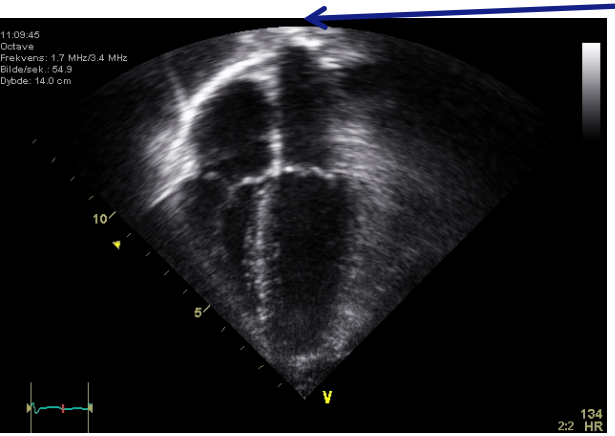
- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint** a way of looking at a system from the position of a certain *stakeholder* with a particular *concern*
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques
  - **View**: whatever you see in a system from a particular viewpoint
    - collection of system models
    - conforming to the viewpoint
  - **Model**:
    - leaves out details irrelevant to concerns
    - preserves the properties of interest with respect to those concerns

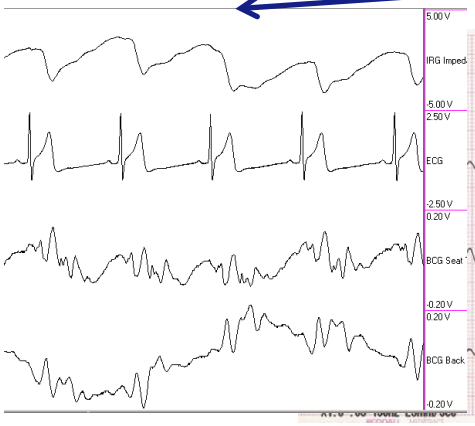
# Medical example

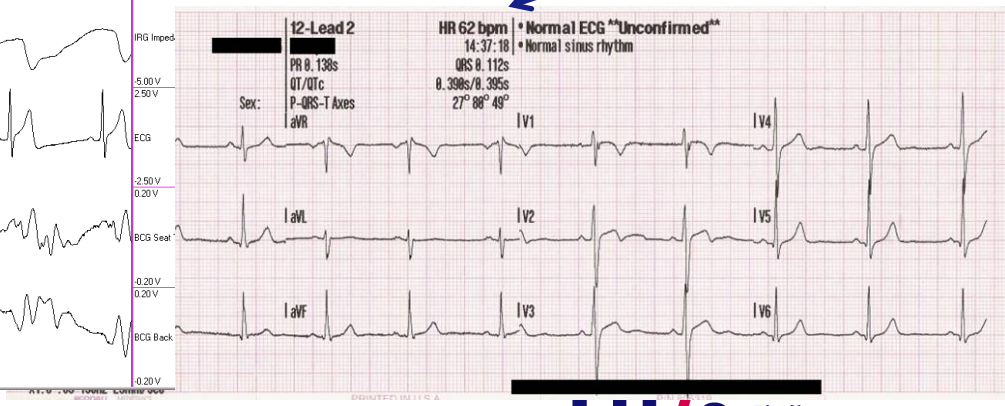
- **System**: human heart
- **Stakeholder**: cardiologist [*class*: Tester/Maintainer]
- **Concerns**: identify and treat disorders of the heart
- **Viewpoint**:
  - used models: electrocardiography (ECG), echocardiography, ballistocardiography
  - analysis techniques: ECG interpretation, …
- **View**:

**Model kinds**

**Models**

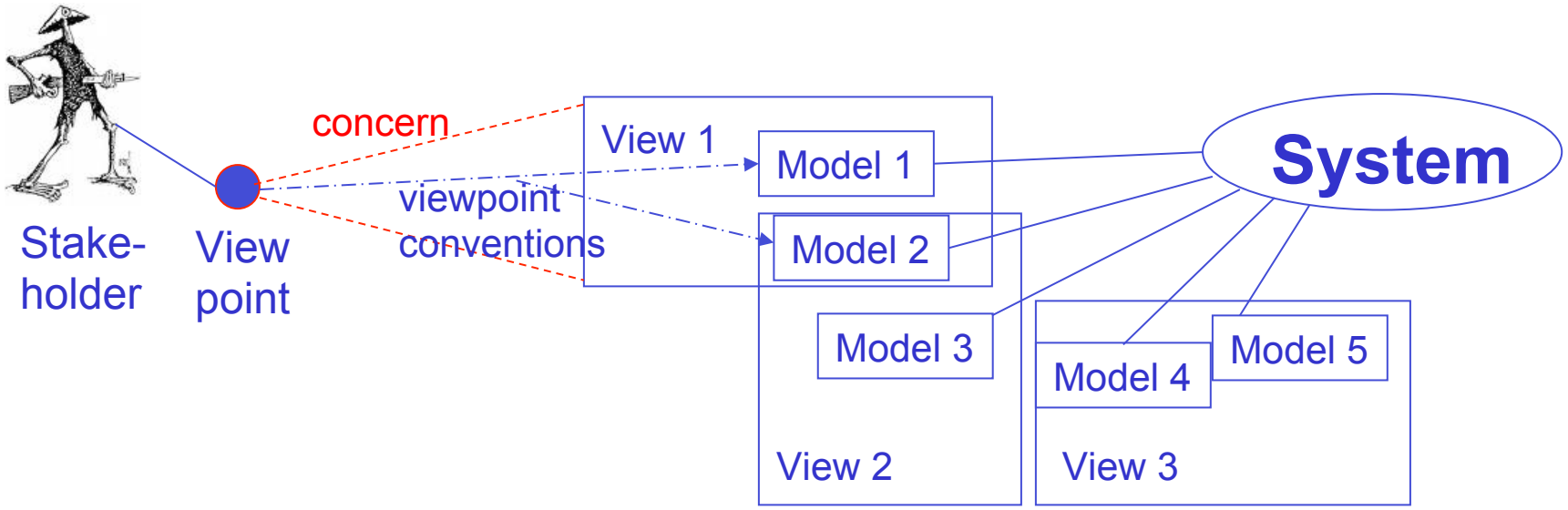


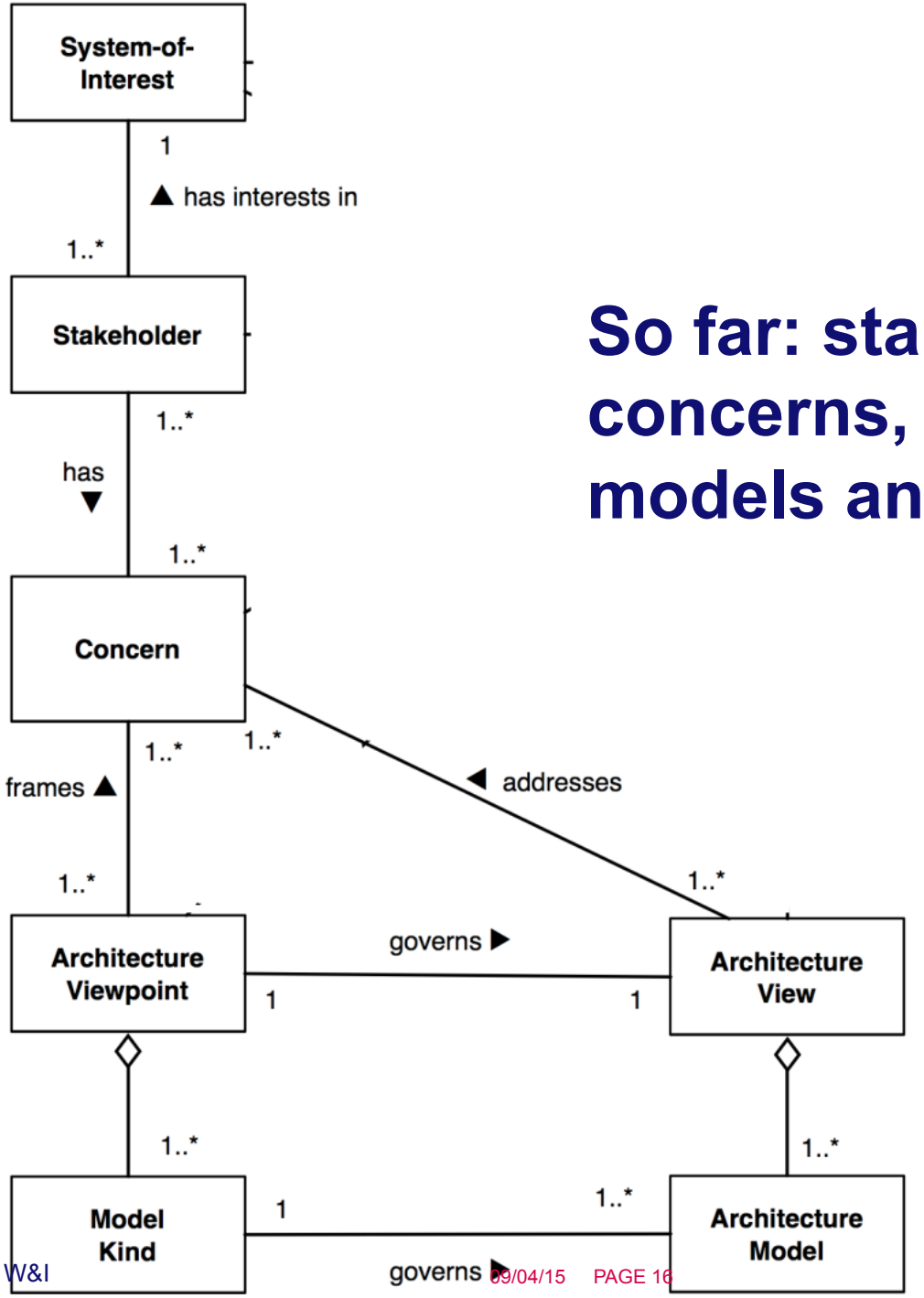



# Viewpoint vs View

- **Viewpoint** should govern the **View**
  - Relation similar to type/instance, grammar/language, …
  - Viewpoints are generic, and can be stored in libraries for re-use.
  - A view is always specific to the architecture for which it is created.




*Schema by Johan Lukkien*




# So far: stakeholders, concerns, views, viewpoints, models and their kinds

# Library viewpoints

- “Viewpoints are generic, and can be stored in libraries for re-use”

- What are the popular library viewpoints?
  - IEEE Standards to not indicate specific viewpoints
  - Kruchten’s 4+1 [1995]
  - Rozanski, Woods [2005]

# Kruchten's 4+1

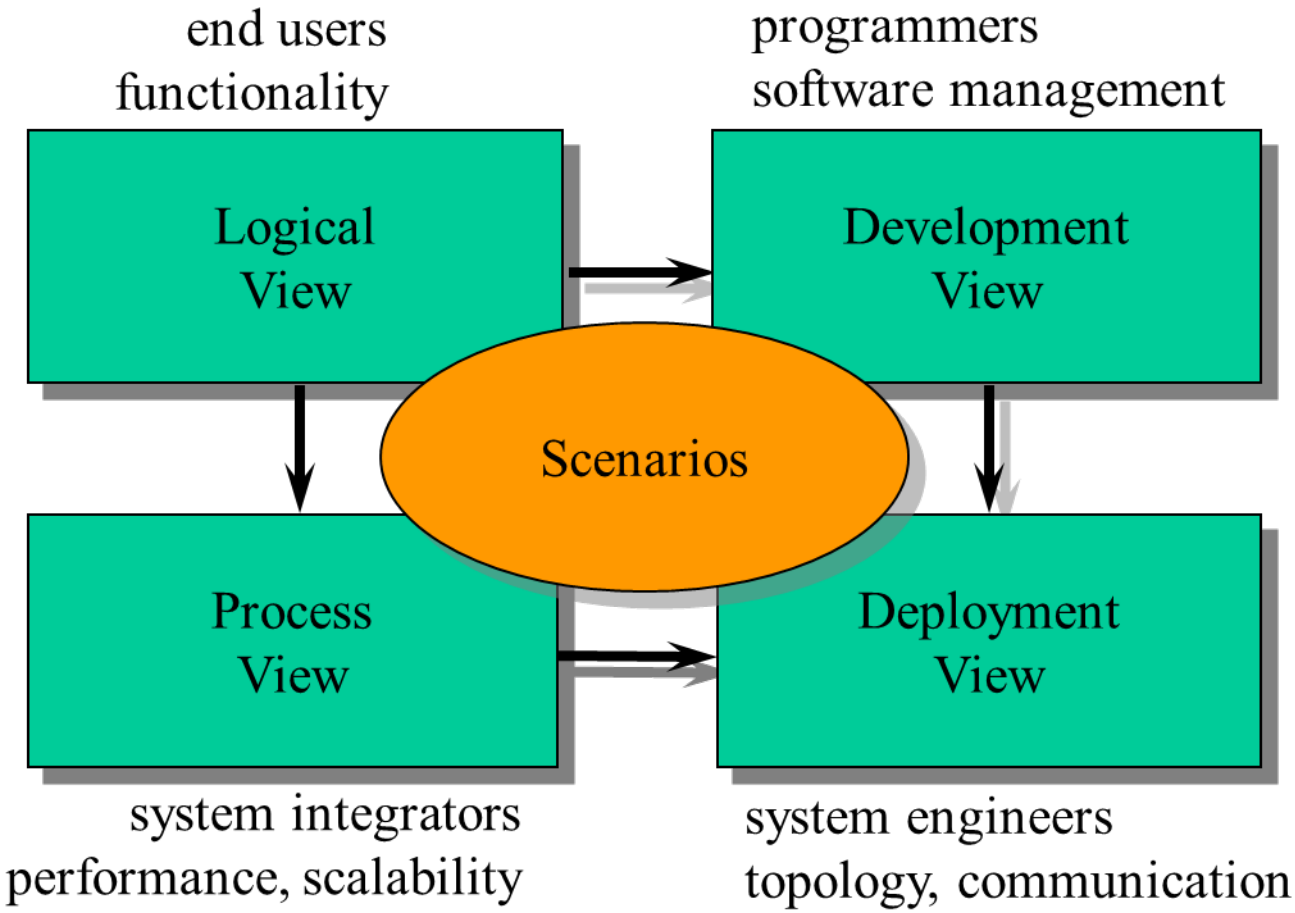
- **1995:** Kruchten, Philippe. *Architectural Blueprints — The "4+1" View Model of Software Architecture*.
  - popularized the "multiple views" idea

- **2000**: IEEE Standard 1471
  - formal conceptual model for architectural descriptions
  - standard terminology
  - distinguish between *views* and *viewpoints*

- Be careful: Kruchten's "views" are viewpoints
  - The word "view" is used for historical reasons

# Kruchten's "views" (viewpoints)

## A number of standard (library) views

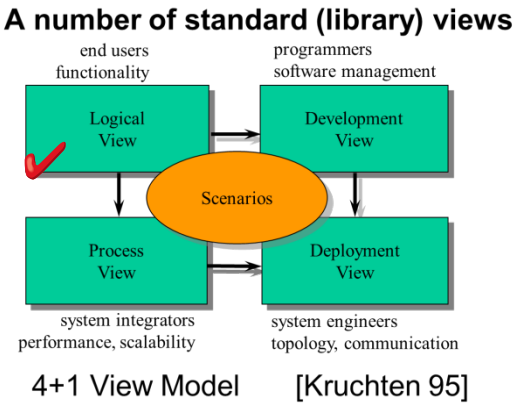



4+1 View Model [Kruchten 95]

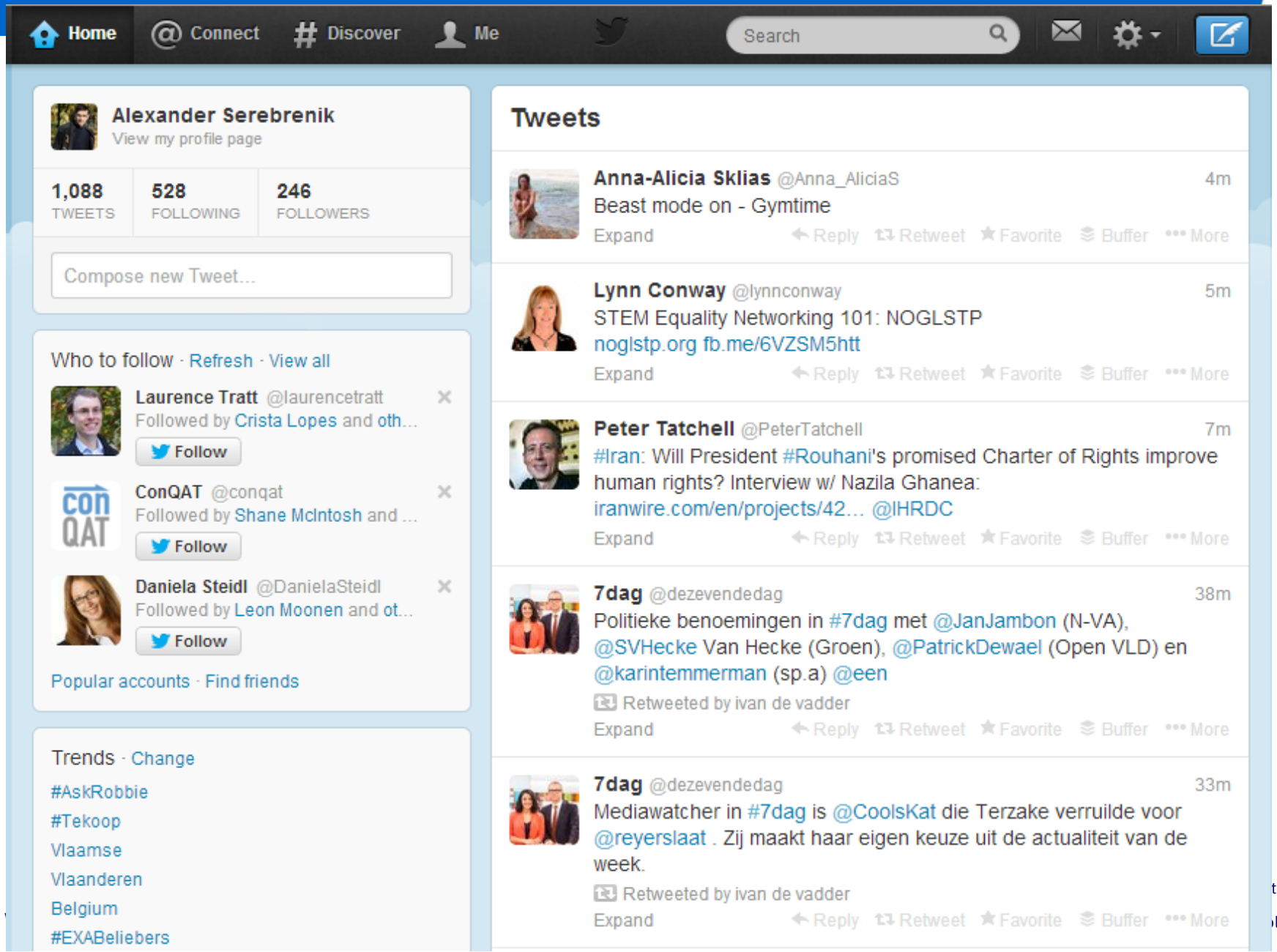
# Logical view

- The **user**'s view on the system, i.e. what a user encounters while using the system
  - **classes** and objects (class instances) documenting user-visible entities
    - including *interfaces,* that imply *responsibilities*
  - **interaction diagrams**: interactions describing usage scenario's
  - **state diagrams**: describing state changes as result of interactions

- Typical relations: is-a, has-a, …



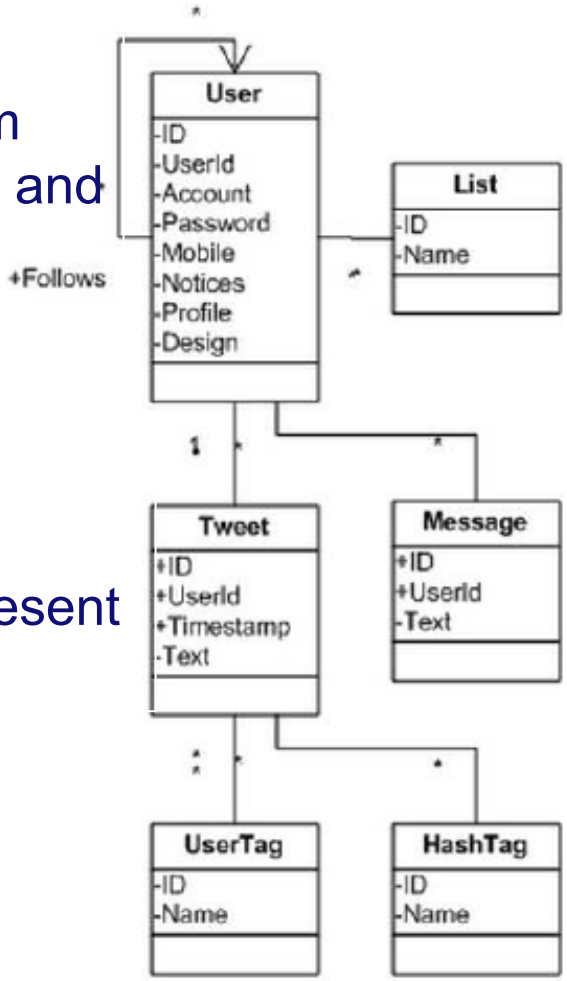
# Example: Twitter



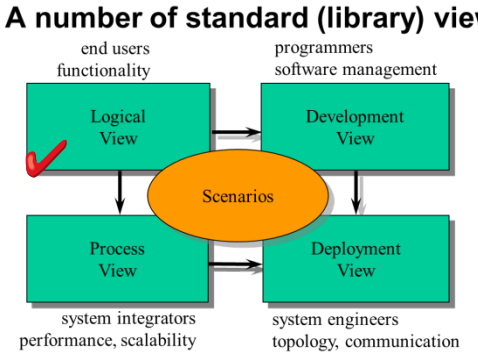
# Twitter example: Logical view

A **domain model**

- class diagram
- captures concepts from the application domain and their relationships




Additional models will typically be used to represent the logical view.




4+1 View Model [Kruchten 95]

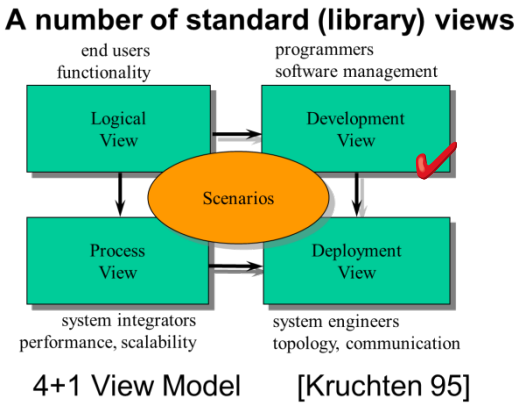


*With thanks to Rudolf Mak*

# Development view

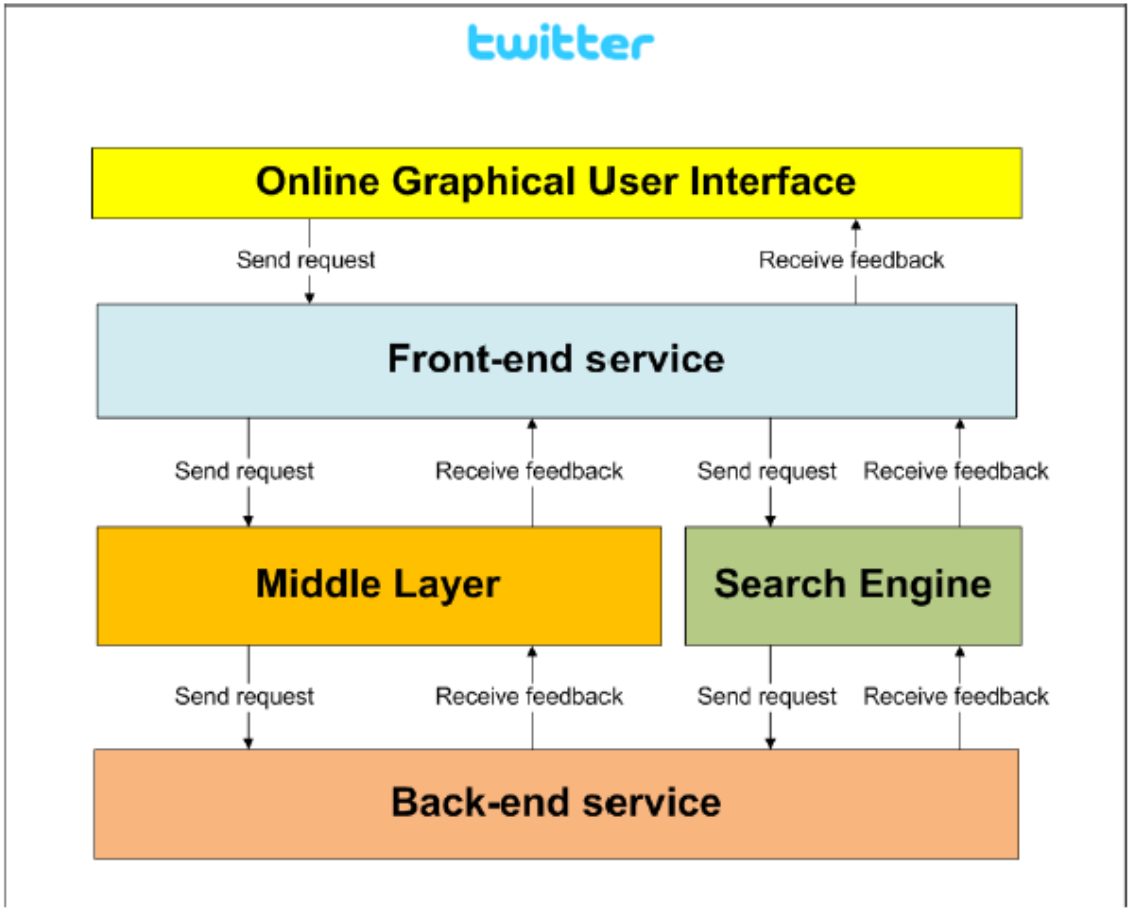
- **Components**, functions, **subsystems** organized in modules and **packages**
  - Component/module interface descriptions, access protocols
- Logical organization – **layering** of functionality, **dependencies**
  - Don't misunderstand the name 'logical'
- Organization into files and folders

- Typical relations: uses, contains, shares, part-of, depends-on



4+1 View Model [Kruchten 95]

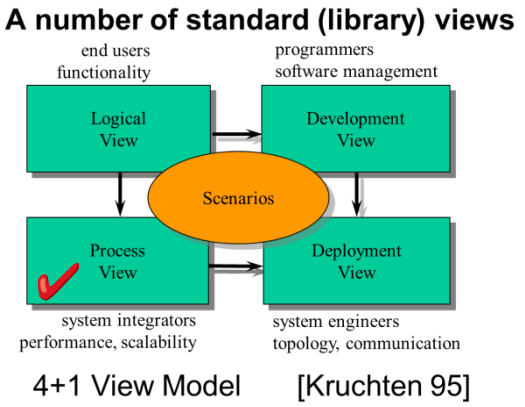
# Twitter example: Development view




*With thanks to Rudolf Mak*

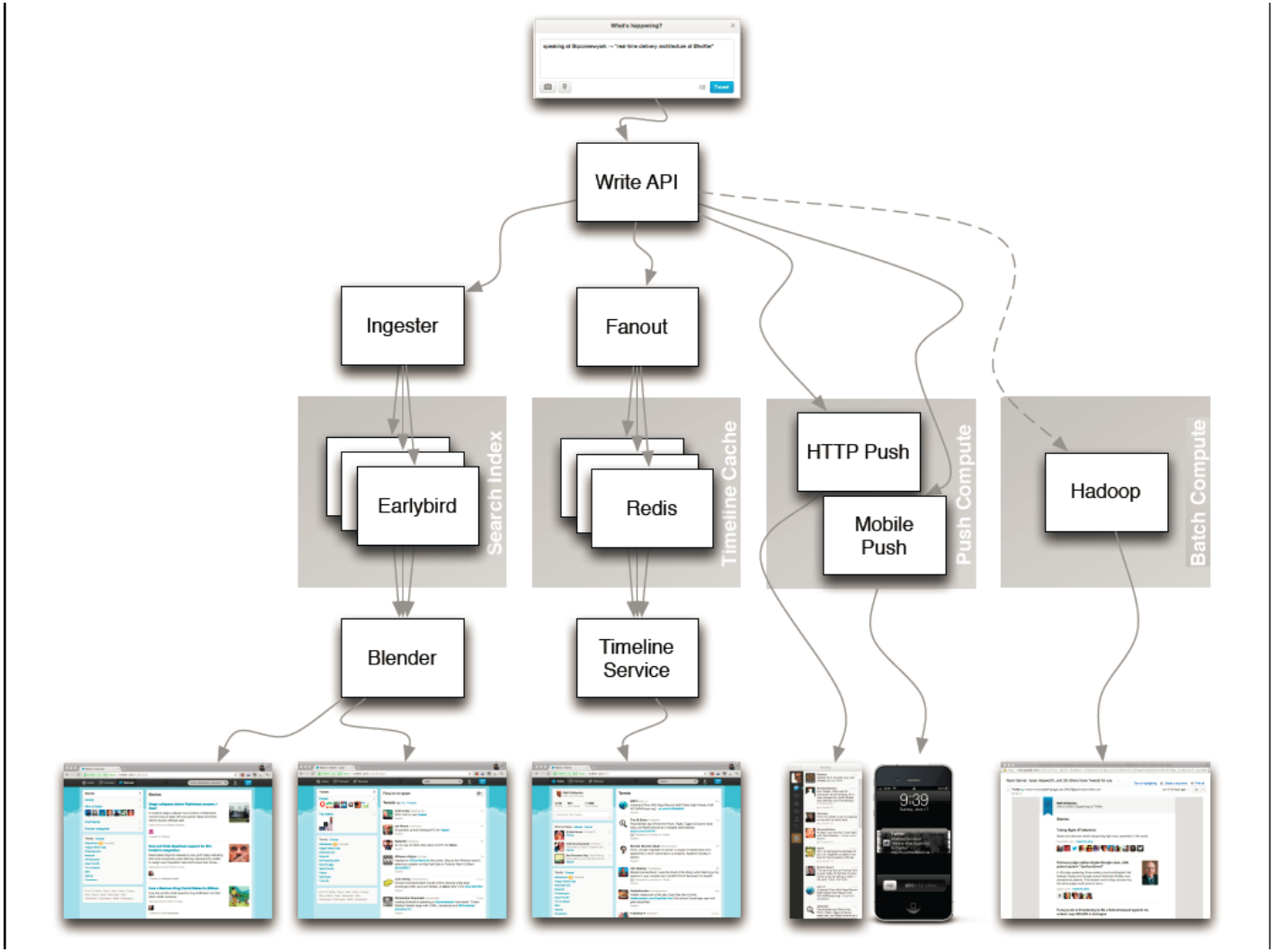
# Process view

- Concurrent **activities** inside the system
- Mapping of applications to distinct memory spaces and units of execution
  - unit of execution: process, thread
  - memory space: associated with a process
- Choice of communication protocols
- Scheduling of activities such as to satisfy **time** and resource constraints
  - Performance



# Twitter example: process view

http://www.slideshare.net/raffikrikorian/qcon-nyc-2012-twitters-real-time-architecture

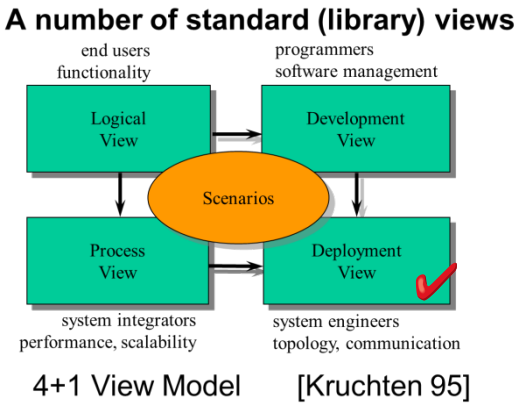





*With thanks to Rudolf Mak*

# Deployment view

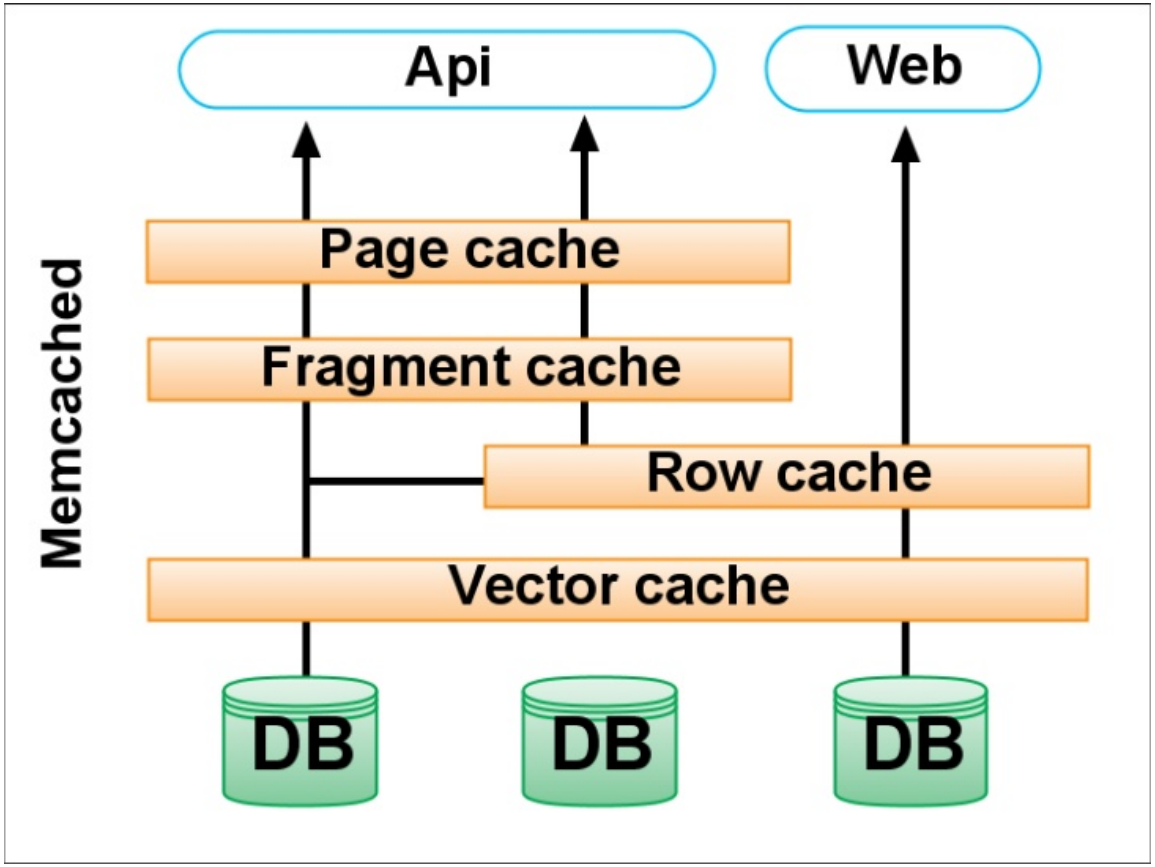
- **Machines** (processors, memories), networks, conncetions
  - including specifications, e.g. speeds, sizes
- **Deployment**: mapping of elements of other views to machines
- Typical relations: connects-to, contains, maps-to
- Concerns: performance (throughput, latency), availability, reliability, etc., together with the process view



# Twitter example: Deployment view

- Not very convincing
- No physical components

- It is hard to find a twitter deployment model

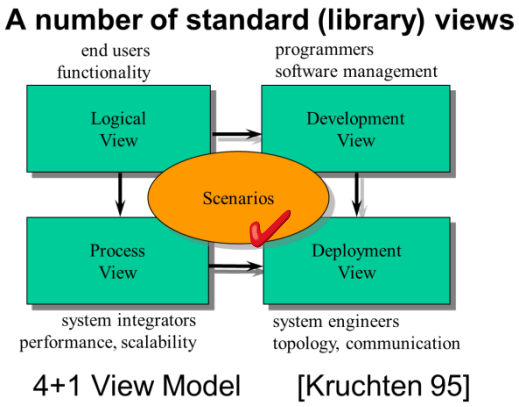





*With thanks to Rudolf Mak*

# Scenarios

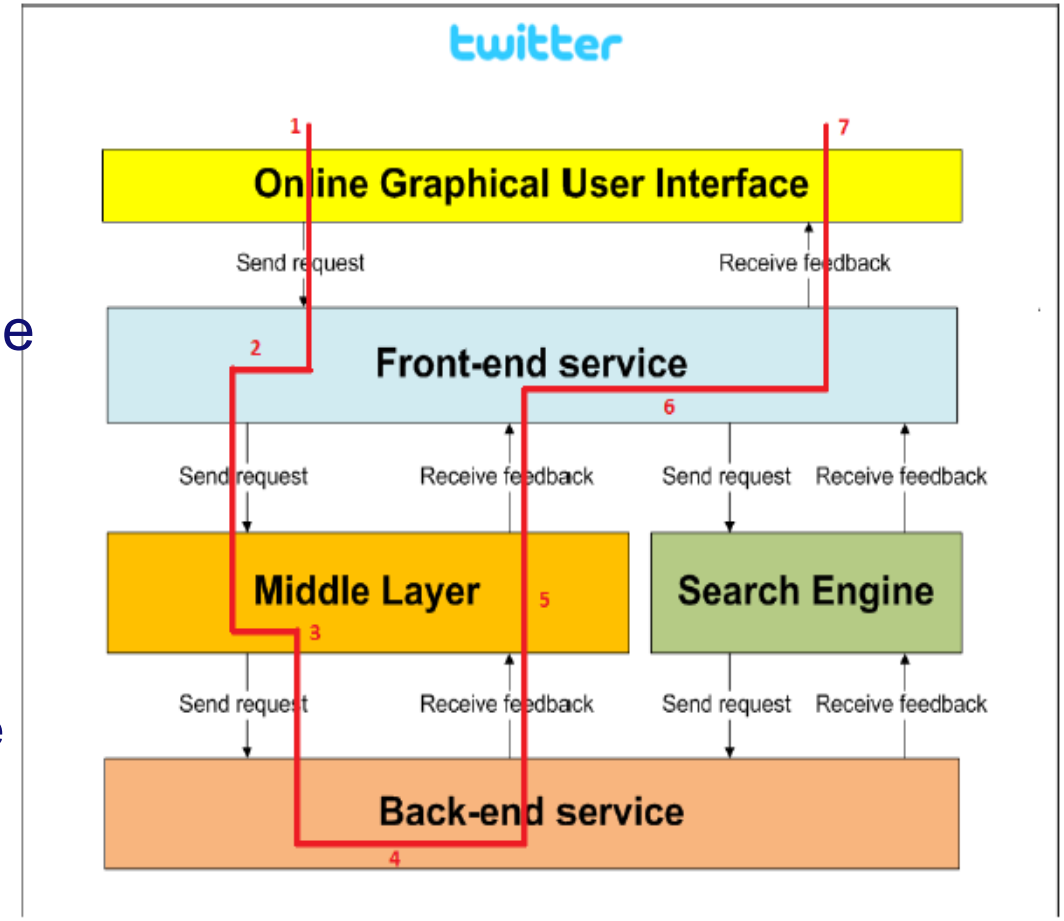
- Set of **use cases**
  - Small set
  - Representative use cases




4+1 View Model [Kruchten 95]

# Twitter scenario example: insertion of a tweet

1. User:
   a) Logon to GUI
   b) Send tweet
2. Process the insertion request
3. Put the request in the queue system
4. Back-end
   a) Get request from front of queue
   b) Insert tweet into memory.
   c) Update the search engine with an index entry
5. Send feedback to queuing system
6. Notify GUI

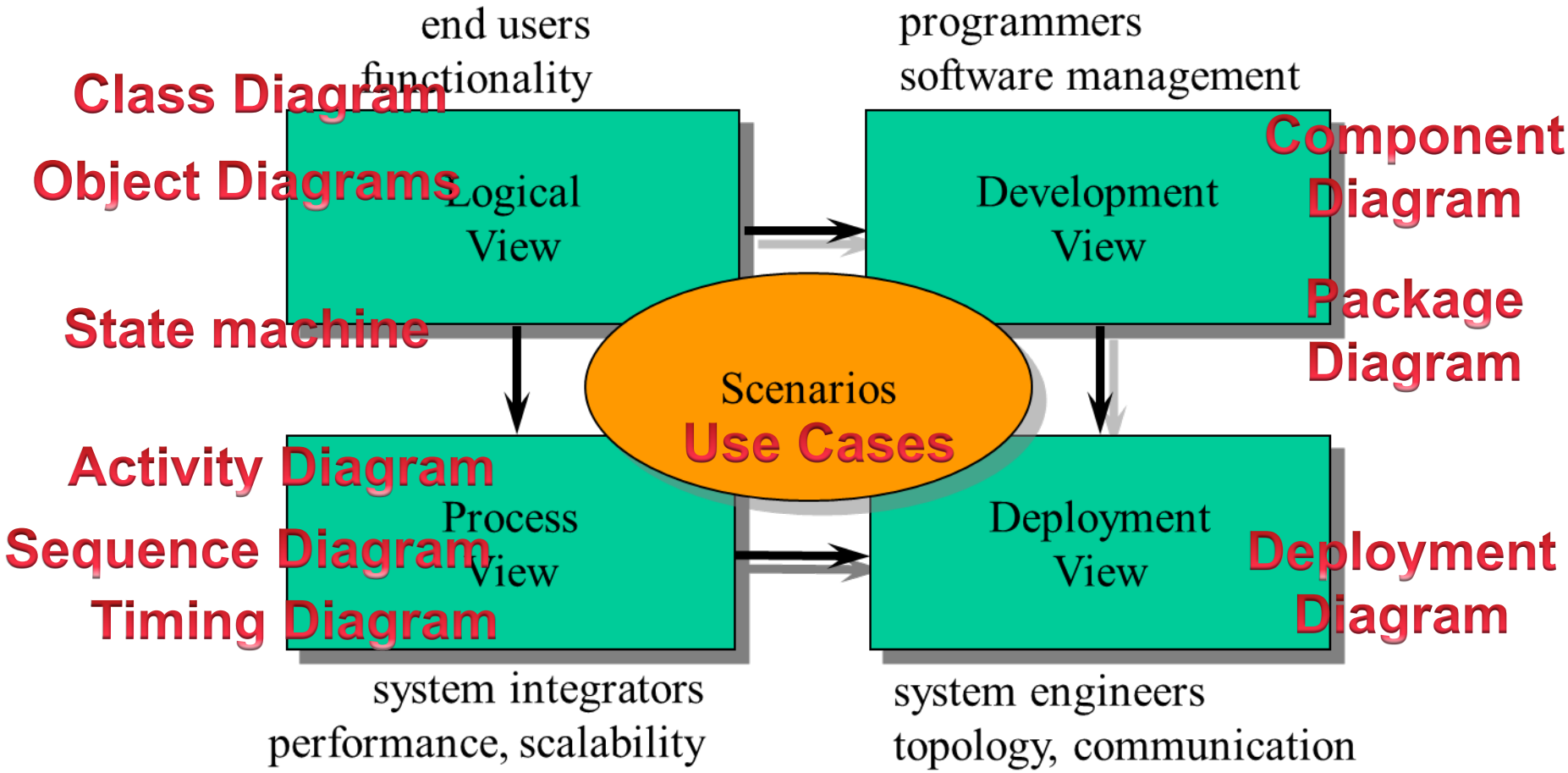


# More examples?

- **The original paper by Philippe Kruchten**
  - http://www.win.tue.nl/~aserebre/2IW80/2014-2015/4+1view-architecture.pdf
- **Hewlett Packard template for documenting software and firmware architectures**
  - http://www.win.tue.nl/~aserebre/2IW80/2014-2015/HP_arch_template.pdf

# UML diagrams and Kruchten's view(point)s?

## A number of standard (library) views



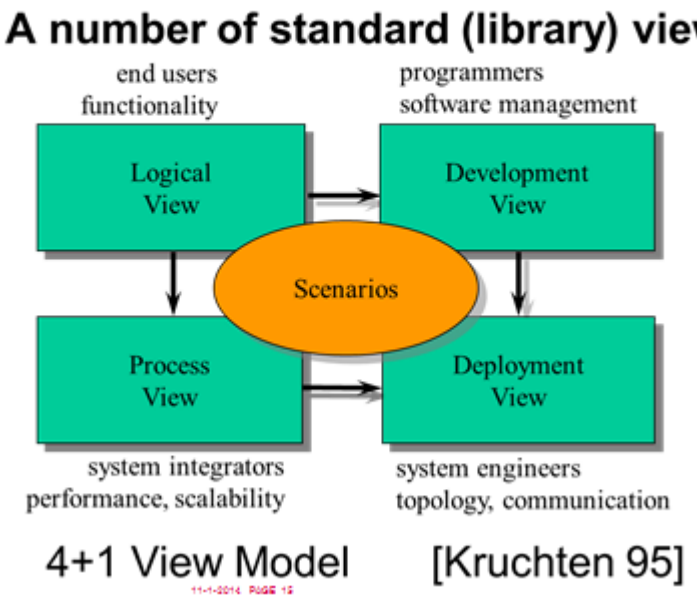

4+1 View Model [Kruchten 95]

# Taylor, Medvidovic, Dashofy: Alternative classification of viewpoints

- **Logical**: capture the logical (often software) entities in a system and how they are interconnected.
- **Physical**: capture the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment**: Capture how logical entities are mapped onto physical entities.
- **Concurrency**: Capture how concurrency and threading will be managed in a system.
- **Behavioral**: Capture the expected behavior of (parts of) a system.

# How would we map different classifications?

## Kruchten's "views" (viewpoints)



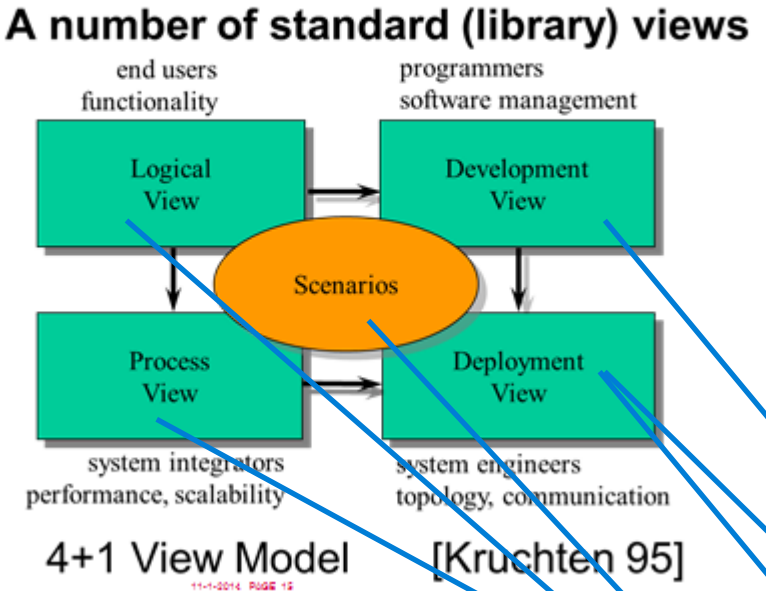

Link Kruchten's views and viewpoints of Taylor, Medvidovic Dashofy to each other

## Taylor, Medvidovic, Dashofy: Alternative classification ov viewpoints

- **Logical**: capture the logical (often software) entities in a system and how they are interconnected.
- **Physical**: capture the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment**: Capture how logical entities are mapped onto physical entities.
- **Concurrency**: Capture how concurrency and threading will be managed in a system.
- **Behavioral**: Capture the expected behavior of (parts of) a system.

# How would we map different classifications?

## Kruchten's "views" (viewpoints)




## Taylor, Medvidovic, Dashofy: Alternative classification ov viewpoints

- **Logical**: capture the logical (often software) entities in a system and how they are interconnected.
- **Physical**: capture the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment**: Capture how logical entities are mapped onto physical entities.
- **Concurrency**: Capture how concurrency and threading will be managed in a system.
- **Behavioral**: Capture the expected behavior of (parts of) a system.

# Rozanski Woods: Alternative classification

- **Viewpoint library**
  - Functional viewpoint
  - Information viewpoint
  - Concurrency viewpoint
  - Development viewpoint
  - Deployment viewpoint
  - Operational viewpoint

Not explicitly addressed by Kruchten's 4+1



*Slide inspired by Rudolf Mak's slides*

# Rozanski Woods: Alternative classification

- **Viewpoint library**
  - Functional viewpoint
  - Information viewpoint
  - Concurrency viewpoint
  - Development viewpoint
  - Deployment viewpoint
  - Operational viewpoint

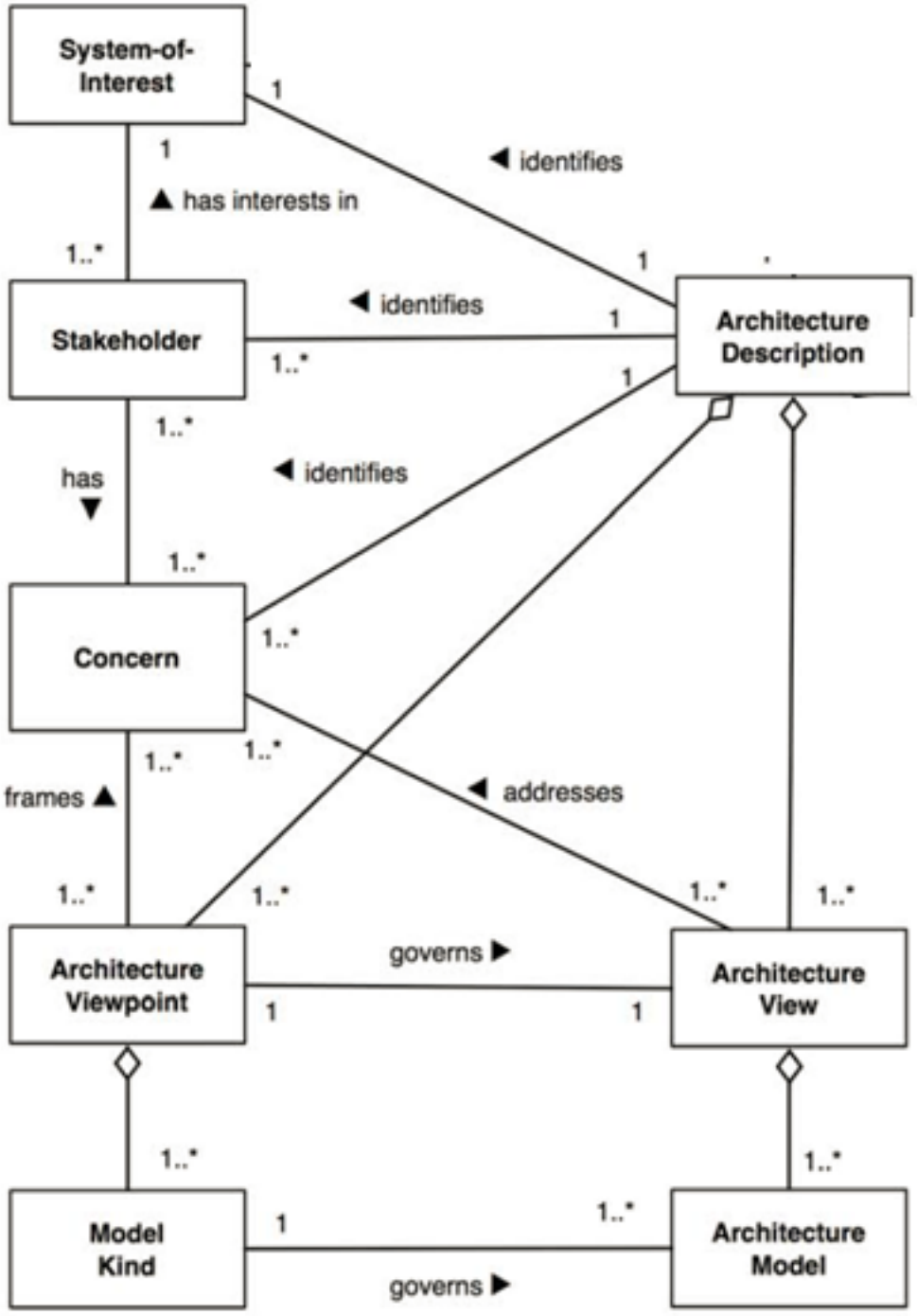
Not explicitly addressed by Kruchten's 4+1

- **Perspectives** (a.k.a. non-functional requirements)
  - Security
  - Performance and scalability
  - Availability and resilience
  - Evolution
  - and many more …

Orthogonal to viewpoints/views

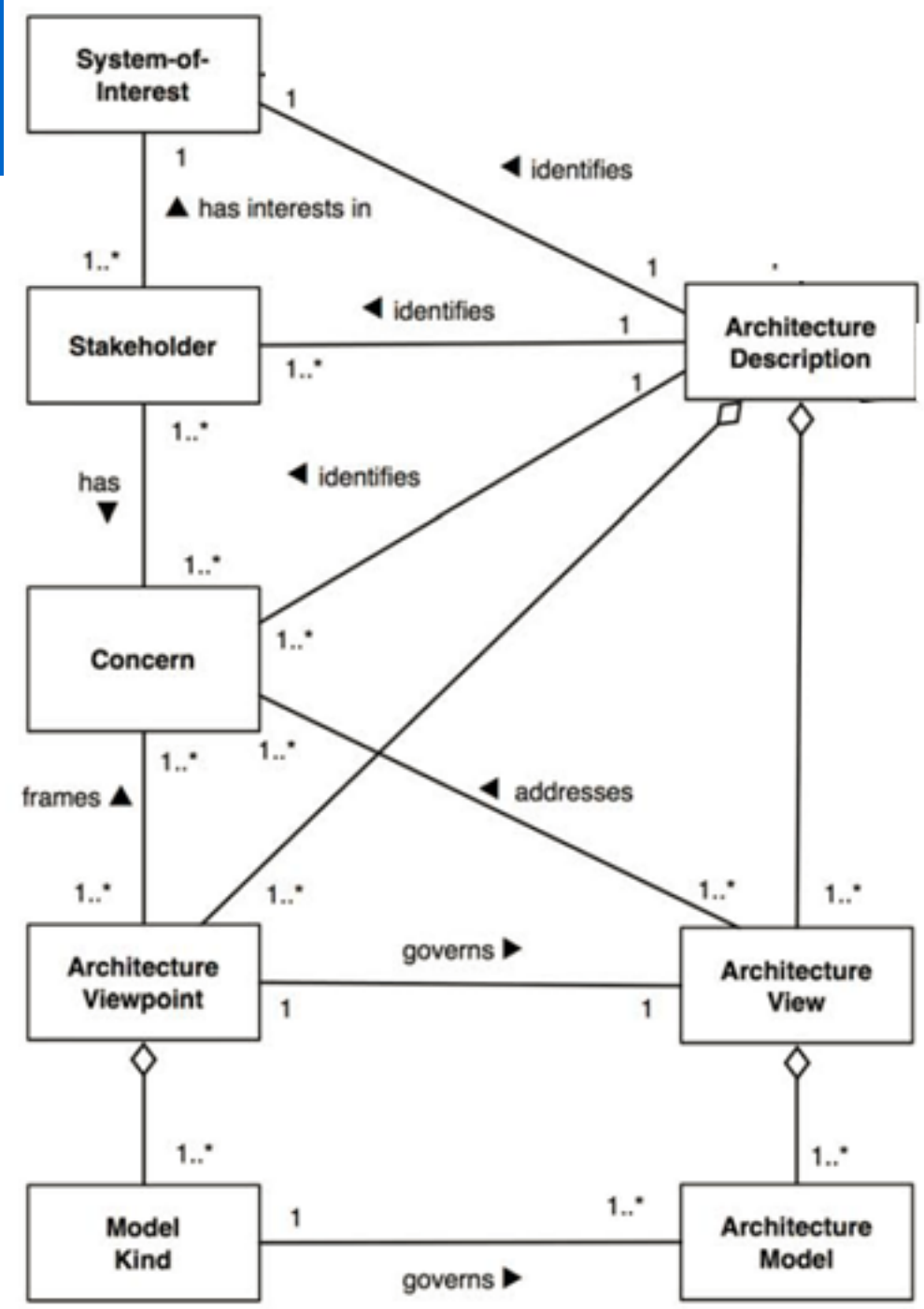


*Slide inspired by Rudolf Mak's slides*




**Recall:** Kruchten popularized “multiple views” idea

Architectural description aggregates **multiple** viewpoints and views.




**Recall:** Kruchten popularized “multiple views” idea

Architectural description aggregates **multiple** viewpoints and views.

What **risks** associated with multiplicity of views can you identify?

# Risks due to multiplicity of views


*Wrong views*

*Fragmentation*


*Inconsistency*

# Wrong views

- Which views are suitable?
- Kruchten, Rozanski-Woods provide guidance but **specifics of the system** should be taken into account
  - Is concurrency important? Is security important?
- Matter of experience and skill

# Fragmentation

- Kruchten: 4+1, Rozanski-Woods: 6
- What about 50 views?

- Each view has to be created and maintained ⇒ costs!
- Views have to be kept consistent, the more views the more potential inconsistencies ⇒ costs!

- Consider combining less crucial views
  - e.g. deployment & concurrency
  - do not overdo, combined views are more difficult to understand

# Inconsistency

- All views are related
  - They describe the same system!

# Inconsistency

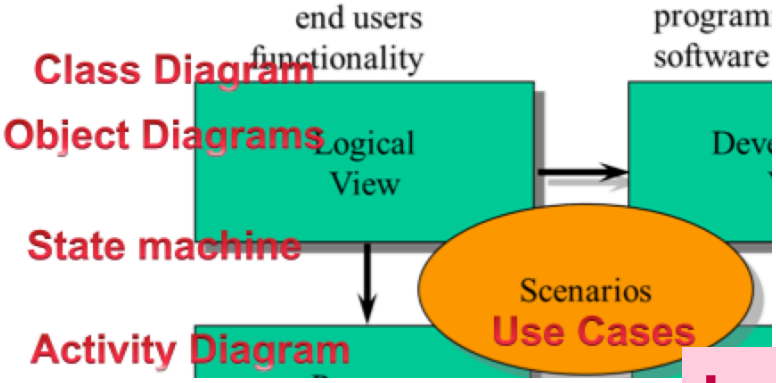
- All views are related
  - They describe the same system!

- Recall: ***interaction diagrams*** *should be consistent with the corresponding* ***class diagrams*** *and* ***use case diagrams***

# Inconsistency

- All views are related
  - They describe the same system!

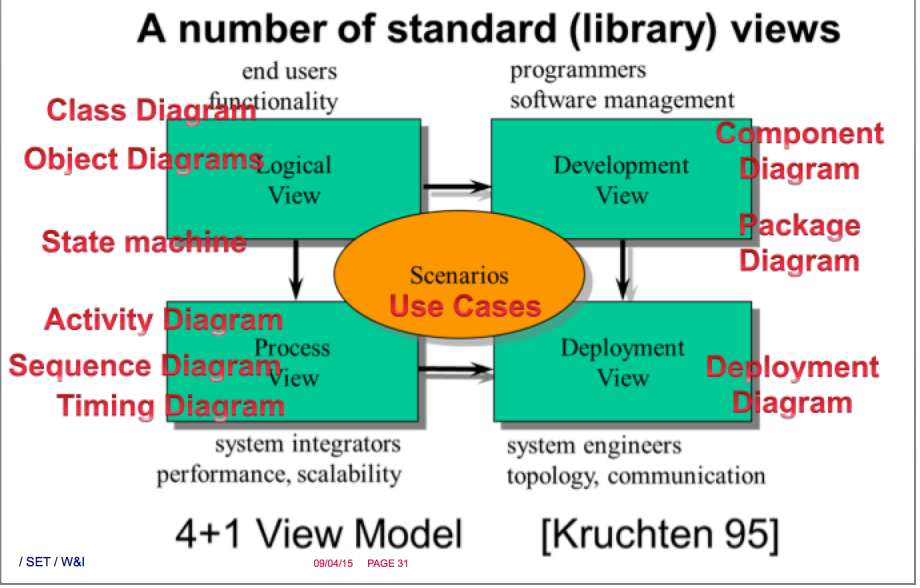

- Recall: ***interaction diagrams*** *should be consistent with the corresponding* ***class diagrams*** *and* ***use case diagrams***




**Logical view should be consistent with scenarios**

# Of course, there are more rules

**UML diagrams and Kruchten's view(point)s?**

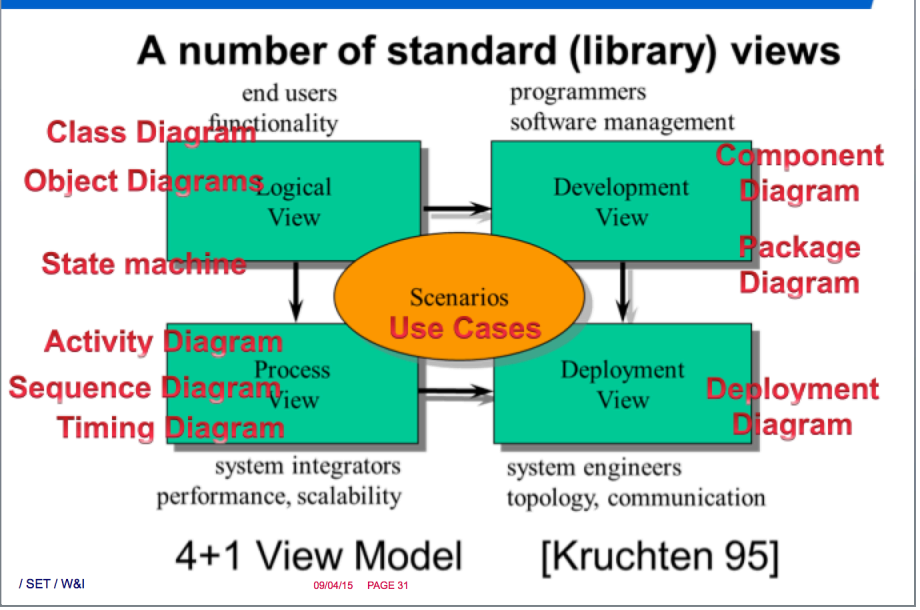



Quote: *"Package A **depends** on package B if A contains a class which depends on a class in B"*

⇒ **Development view should be consistent with the logical view**

# Of course, there are more rules

**UML diagrams and Kruchten's view(point)s?**




/ SET / W&I 09/04/15 PAGE 31

Quote: *"Package A **depends** on package B if A contains a class which depends on a class in B"*

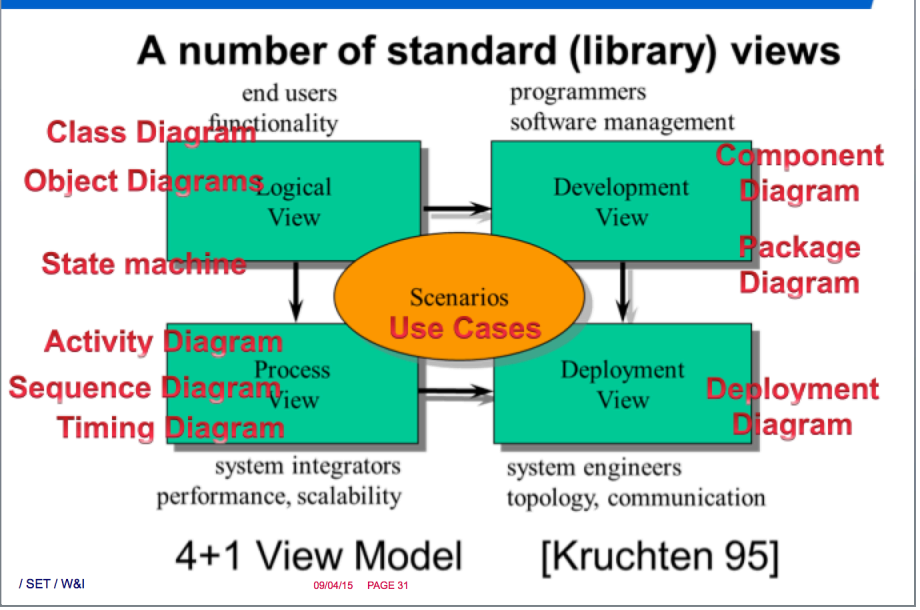
⇒ **Development view should be consistent with the logical view**

Quote: "Manifestation maps artifacts to … / components / packages"

⇒ **Deployment view should be consistent with the development view**

# Of course, there are more rules

**UML diagrams and Kruchten's view(point)s?**




Quote: *"Package A **depends** on package B if A contains a class which depends on a class in B"*

⇒ **Development view should be consistent with the logical view**

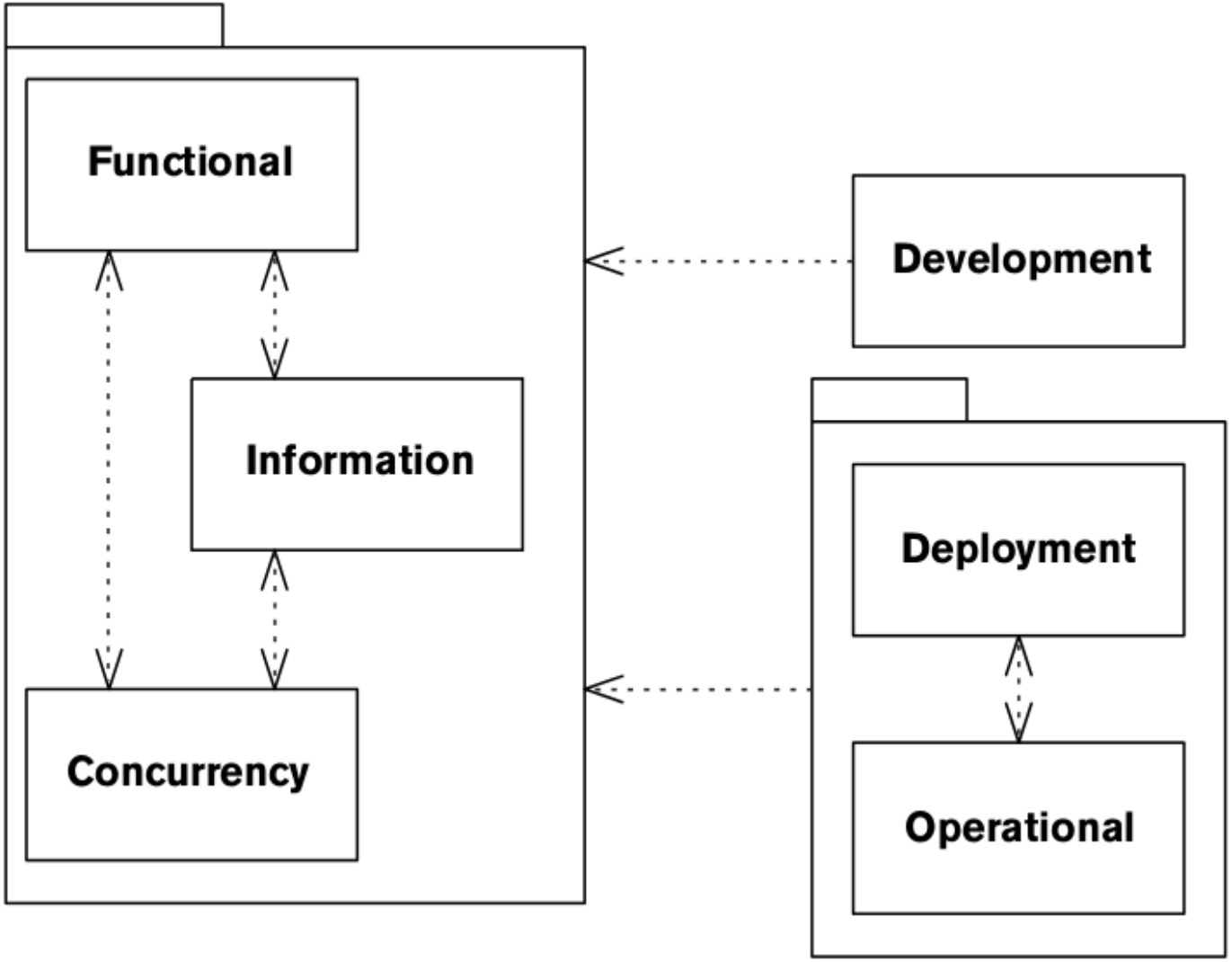
Quote: "Manifestation maps artifacts to … / components / packages"

⇒ **Deployment view should be consistent with the development view**

Quote: "Manifestation maps artifacts to use cases"

⇒ **Deployment view should be consistent with scenarios**

# Consistency for Rozanski and Woods



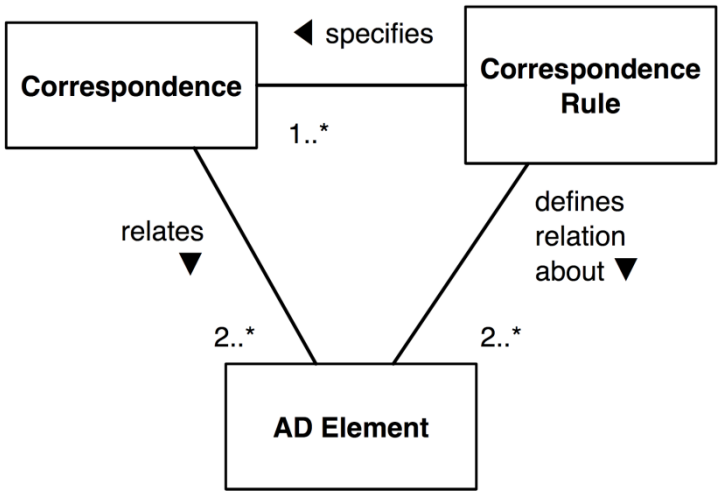

If something changes at the **end** of the arrow, a change will probably be required at the **start** of the arrow.

# Still…

- We need a mechanism to record in the architecture description **relations between different views**
  - What views should be consistent with each other?
  - Does one view refine another? (package vs class?)
  - Should one be traced to another?

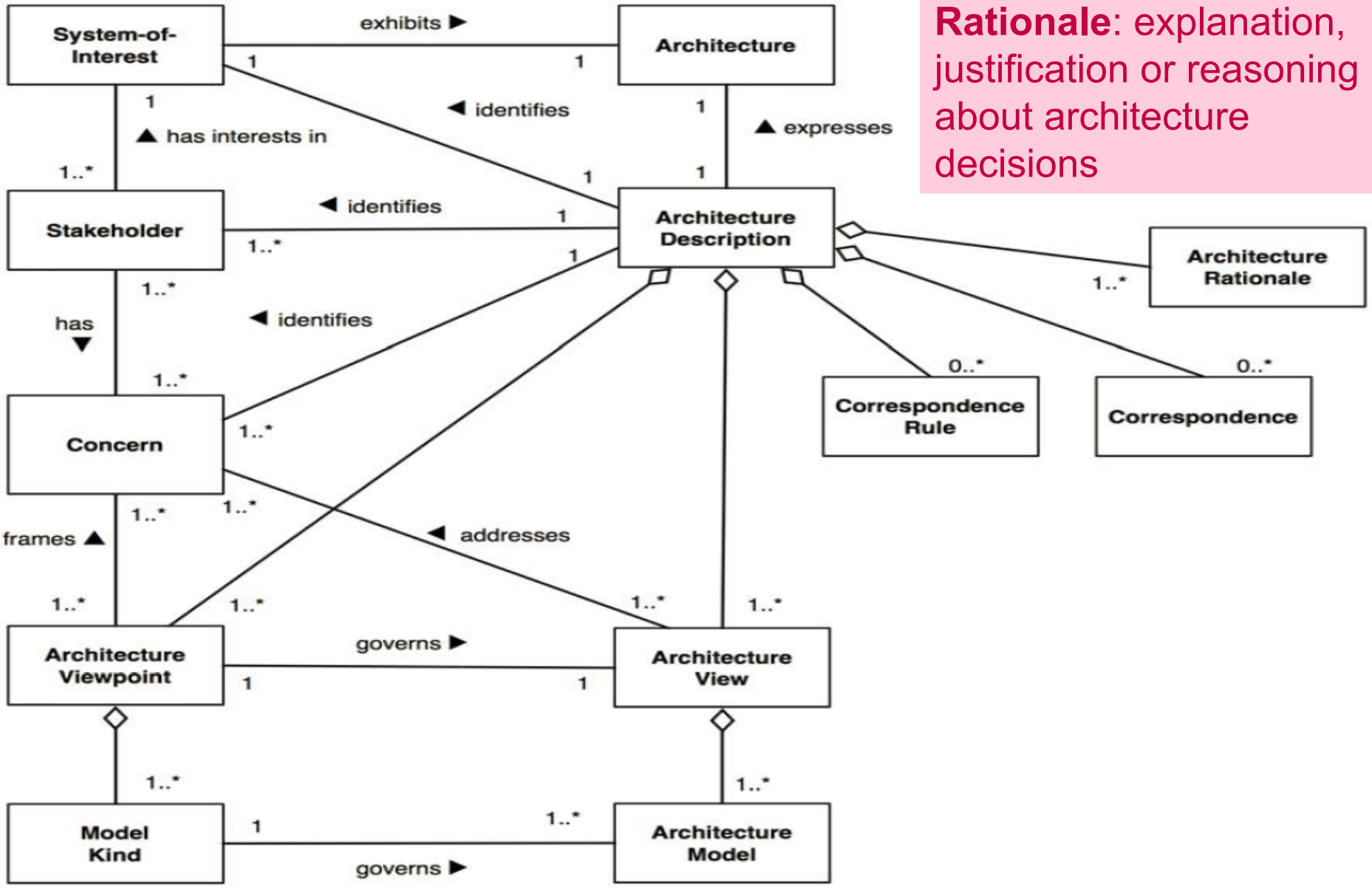
# Still…

- We need a mechanism to record in the architecture description **relations between different views**
  - What views should be consistent with each other?
  - Does one view refine another? (package vs class?)
  - Should one be traced to another?



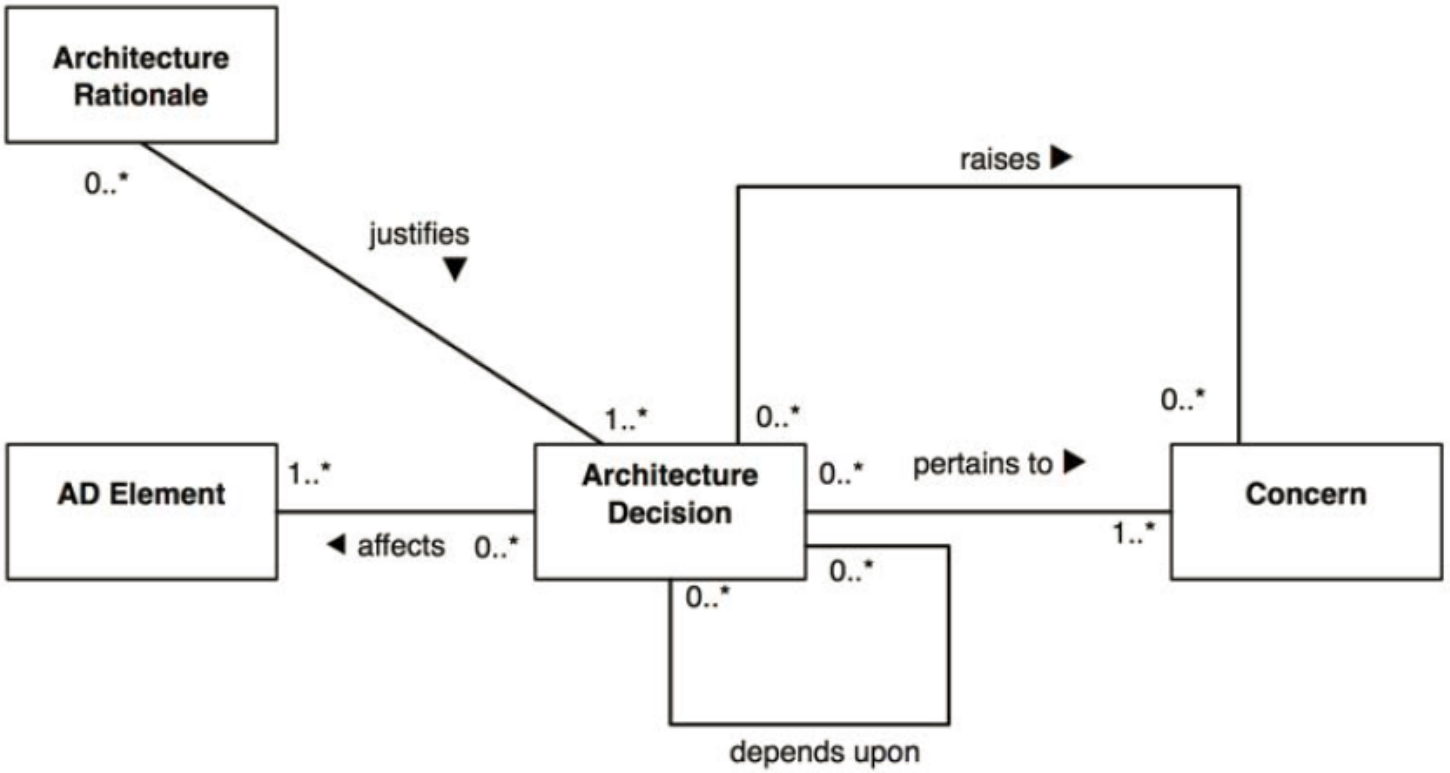

- Architectural description (AD) elements
  - stakeholders, views, viewpoints, models…
- Correspondences vs. Correspondence rules
  - similarly to views vs. viewpoints

# Putting it all together: architecture

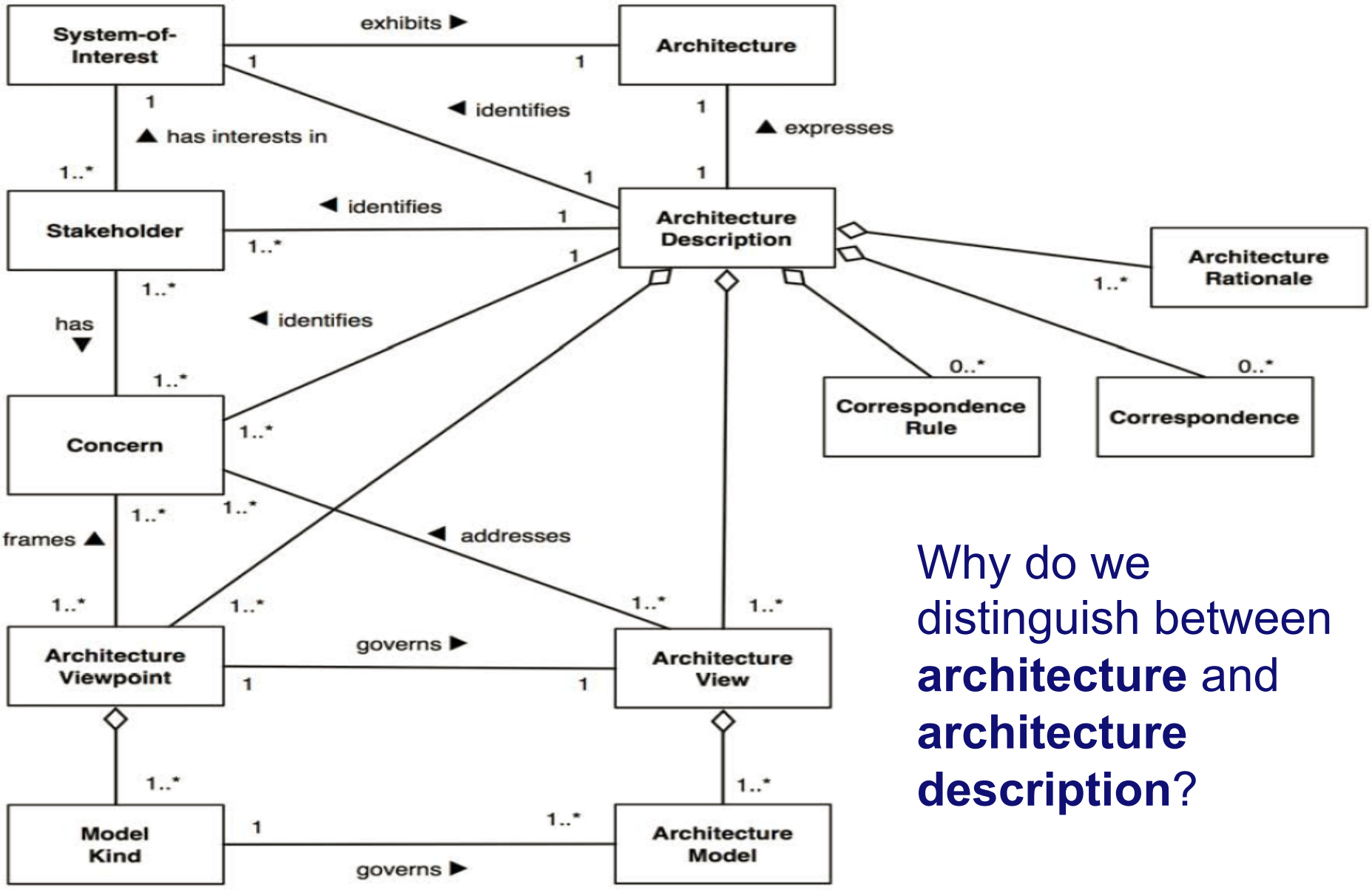



**Rationale**: explanation, justification or reasoning about architecture decisions

# Rationale: closer look

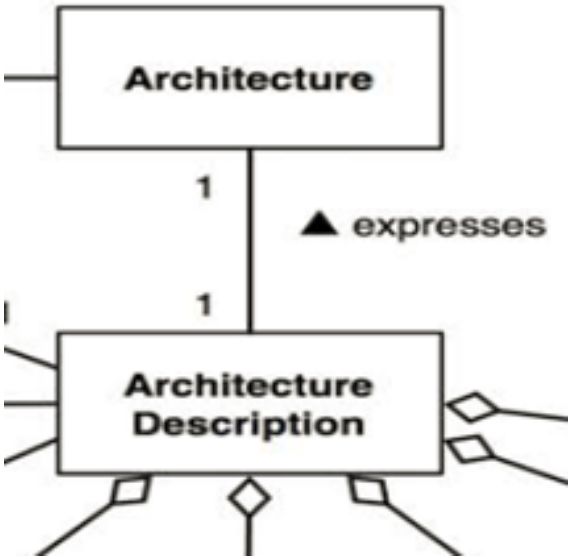


# Putting it all together: architecture




Why do we distinguish between **architecture** and **architecture description**?

# Different flavors of architecture

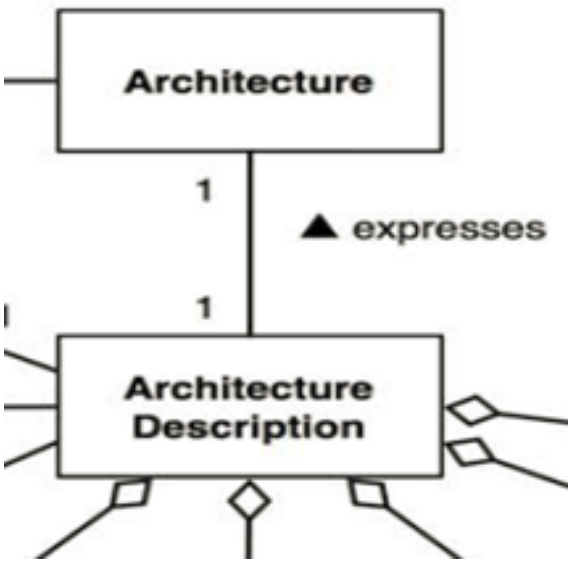



Architecture **as intended**

Architecture **as described**

What other kind of architecture flavor would you expect?

# Different flavors of architecture

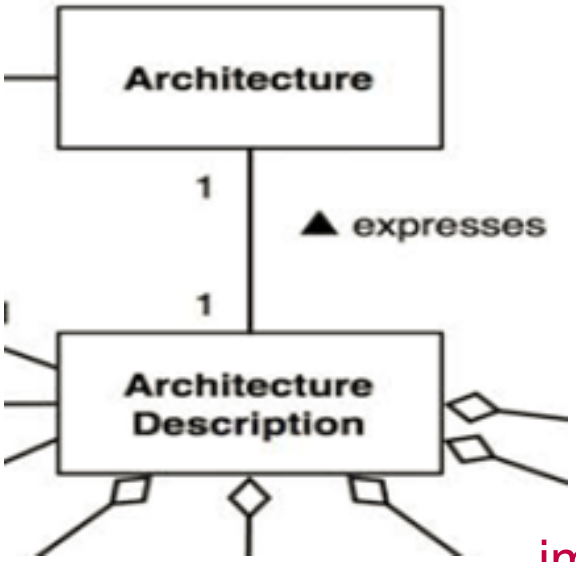



Architecture **as intended**

Architecture **as described**

Architecture **as implemented**

# Different flavors of architecture




Architecture **as intended**

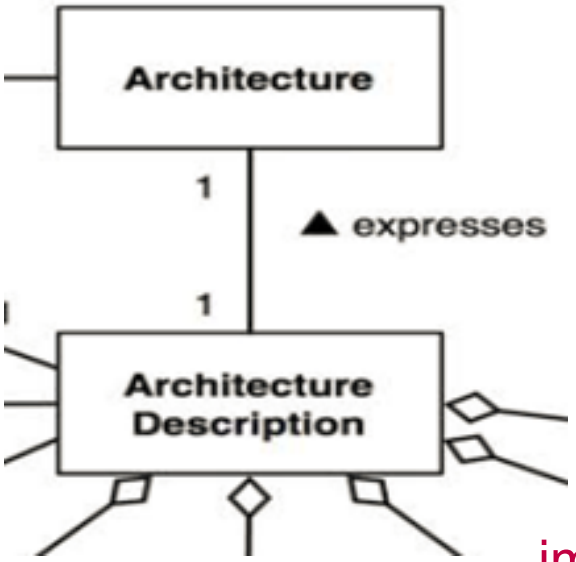
Architecture **as described**

implementation,
forward engineering

architecture
reconstruction,
reverse engineering

Architecture **as implemented**

# Different flavors of architecture




Architecture **as intended**

*Prescriptive architecture*

Architecture **as described**

implementation, forward engineering

architecture reconstruction, reverse engineering

Architecture **as implemented**

*Descriptive architecture*

# Beware: incompleteness

- **Architecture as described** can/should never be complete with respect to
  - the system
    - architecture = “fundamental organization”
      - not everything is fundamental
  - architecture as intended
    - limitation of the architecture description language(s)
  - architecture as implemented
    - freedom of implementation choices

- Architecture as intended should at least be **complete** wrt the stakeholders’ concerns

# Architectural Evolution

- When a system evolves
  - ideally its **prescriptive architecture** is modified first
  - in practice, the system – and thus its **descriptive architecture** – is often directly modified

- This happens because of
  - Developer sloppiness
  - Perception of short deadlines which prevent thinking through and documenting
  - Lack of documented prescriptive architecture
  - Need or desire for code optimizations
  - Inadequate techniques or tool support

# Architectural Degradation

- **Architectural drift** is introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which **do not violate** any of the prescriptive architecture's design decisions

# Architectural Degradation

- **Architectural drift** is introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which **do not violate** any of the prescriptive architecture's design decisions

- **Architectural erosion** is the introduction of architectural design decisions into a system's descriptive architecture that **violate** its prescriptive architecture

# Architectural Degradation

- **Architectural drift** is introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which **do not violate** any of the prescriptive architecture's design decisions

- **Architectural erosion** is the introduction of architectural design decisions into a system's descriptive architecture that **violate** its prescriptive architecture

- If architectural degradation is allowed to occur,
  - one will be forced to *reconstruct* the system's architecture sooner or later

## Software architecture

- Software architecture is the **fundamental organization** of a system embodied in
  - its **elements**,
  - **relationships**,
  - and in the principles of its **design** and **evolution**.

*[IEEE Std. 42010-2011]*

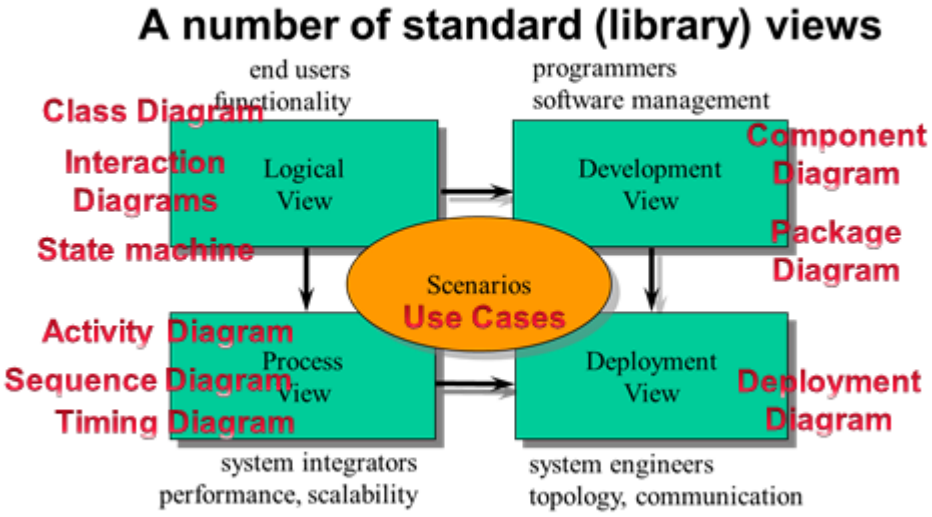


## Stakeholders, goals and concerns

- Different **stakeholders** have different **concerns**
- Different **concerns** imply different **viewpoints**
  - **Viewpoint** a way of looking at a system from the position of a certain *stakeholder* with a particular *concern*
    - defines creation, depiction and analysis of a view
    - language, used models, notation, methods, analysis techniques
  - **View**: whatever you see in a system from a particular viewpoint
    - collection of system models
    - conforming to the viewpoint
  - **Model**:
    - leaves out details irrelevant to concerns
    - preserves the properties of interest with respect to those concerns

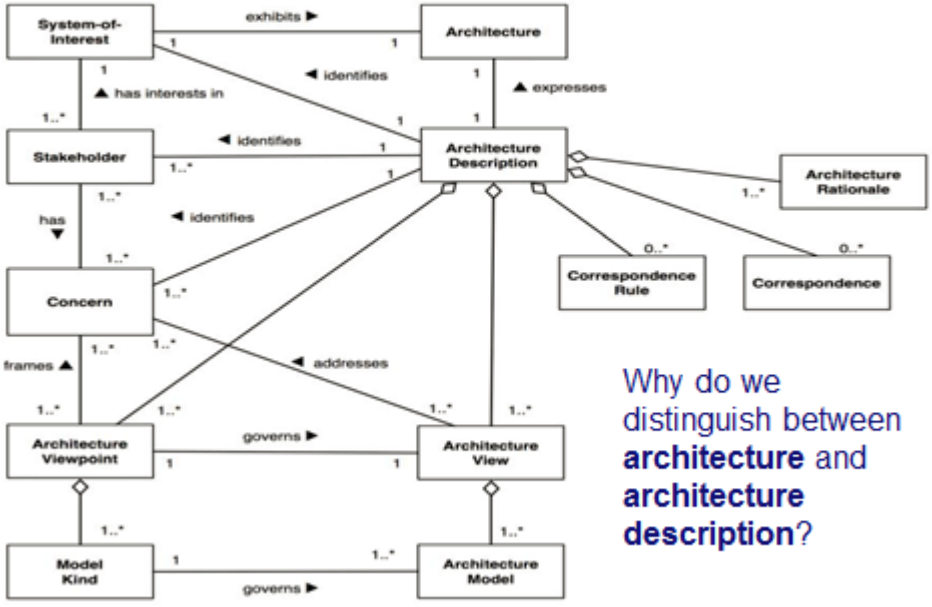


## UML diagrams and Kruchten's view(point)s?






## Putting it all together: architecture




Why do we distinguish between **architecture** and **architecture description**?

# RW: viewpoint library (1 of 2)

- **The functional viewpoint**
  - describes the system's runtime functional elements and their responsibilities, interfaces and primary inter-actions.
  - is relevant to all stakeholders
  - addresses concerns like: functional capabilities, external interfaces, internal structure and design philosophy.

- **The information viewpoint**
  - describes the way the architecture stores, manipulates and distributes information.
  - is primarily relevant for users, acquirers, developers and maintainers
  - addresses concerns about information structure, content and flow; data ownership, volume, validness, lifetime, and accessibility; transaction management; recovery; regulations.

# RW: viewpoint library (2 of 3)

- **The concurrency viewpoint**
  - describes the concurrency units of the system, their functionality, and the required coordination
  - is relevant to developers, testers and some administrators
  - addresses concerns like: task structure, inter process communication, state management, synchronization and reentrance, process creation and destruction

- **The development viewpoint**
  - describes the architecture that supports the development process
  - is relevant to software developers and testers
  - addresses their concerns about. Module organization, common processing, standardizations of design and testing, code organization and instrumentation.

# RW: viewpoint library (3 of 3)

- **The deployment viewpoint**
  - describes the environment into which the system will be deployed, including dependencies the system has on its run-time environment
  - is relevant for system administrators, developers, testers, communicators and assessors and addresses their concerns about
  - hardware (processing elements, storage elements, and network), third-party software, and technology compatibility.

- **The operational viewpoint**
  - describes how the system will be operated, administered, and supported when running in its production environment
  - is relevant to system administrators, developers, testers, communicator, and assessors, and addresses their concerns about
  - installation and upgrade, operational monitoring and control, configuration management, resource management.