Agenda

- It Architecture
- 4+1 Model
- ISO/IEC/IEEE 42010
- Design Steps
- Quality Attributes - Non Functions Requirements

Tue Hellstern © 2022

It Architecture

> The architecture of a software-intensive system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

The Problem of Architectural Description

- What are the main functional elements of your architecture?
- How will these elements interact with one another and with the outside world?
- What information will be managed, stored, and presented?
- What physical hardware and software elements will be required to support these functional and information elements?
- What operational features and capabilities will be provided?
- What development, test, support, and training environments will be provided?

> An architectural description (AD) is a set of artifacts that documents an architecture in a way its stakeholders can understand and demonstrates that the architecture has met their concerns.

What Architecture?

SW Architecture

Focus on a program, or sometimes used to refers to the SW part of a system.

System Architecture

Includes SW, HW and human interaction

Enterprise Architecture

Company wide (or Business wide) system Architecture

Architecture Definition

Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems.

Each structure comprises software elements, relations among them, and properties of both elements and relations.

The architecture of a software system is a metaphor, analogous to the architecture of a building.

It functions as a blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams.

What is bad architecture and how to recognize it?

- **Unnecessarily Complex** — It is easy to write complex code, anyone can do it, but it is hard to write simple code.
- **Rigid/Brittle** — Since it is unnecessarily complex, it is not easy to understand and therefore making it non-maintainable, easy to break for even a small code change.
- **Untestable** — Such code will be tightly coupled, will typically not follow the single responsibility principle, will be difficult to test.
- **Unmaintainable** — Brittle code with less test coverage evolves to becomes a maintenance nightmare

What is good architecture and what properties do they exhibit?

- **Simple** — Easy to understand.
- **Modularity/Layering/Clarity** — This is important so that one layer is able to change independently of the others with as minimum coupling between the layers
- **Flexible/Extendable** — Can be easily adapted to new evolving requirements
- **Testable/Maintainable** — Easy to test, add automated tests, and encourage the culture of TDD and therefore maintainable

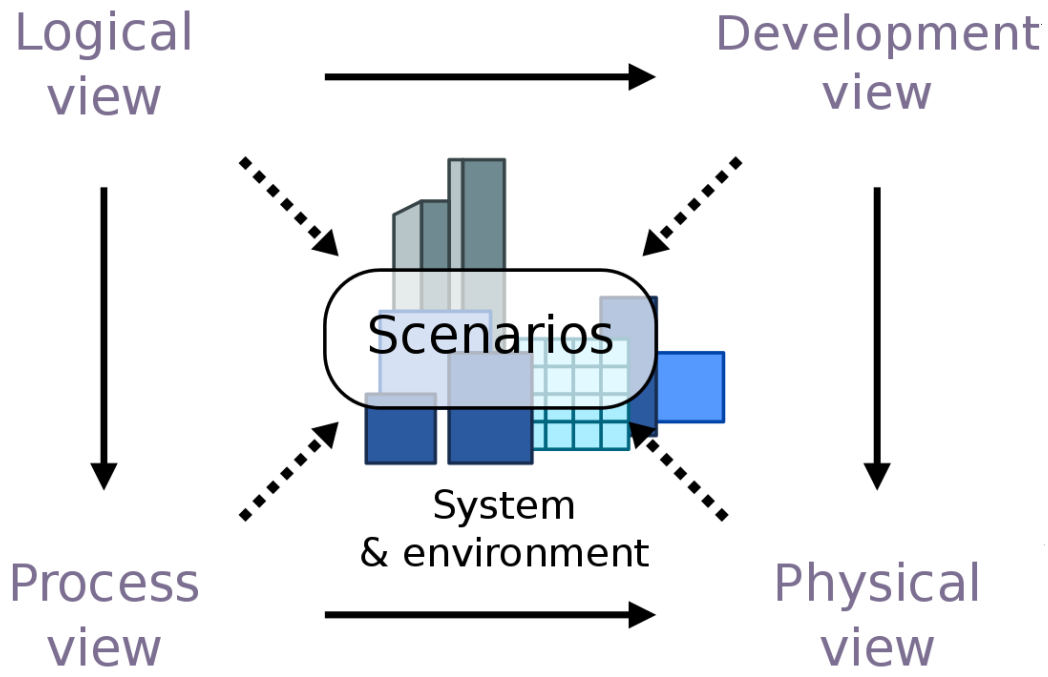
4+1 Model

4+1 Model

Designed by - **Philippe Kruchten**

Describing the architecture of software-intensive systems, based on the use of 4, concurrent views.

- **Logical** (*End user*)
- **Process** (*Integrator*)
- **Development** (*Programmers*)
- **Physical** (*System engineer*)



The “4+1” view model is rather “generic”: other notations and tools can be used, other design methods can be used, especially for the logical and process decompositions, but we have indicated the ones we have used with success.

Philippe Kruchten,
Architectural Blueprints - The "4+1" View Model of Software Architecture

http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Scenarios

The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view.

The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design.

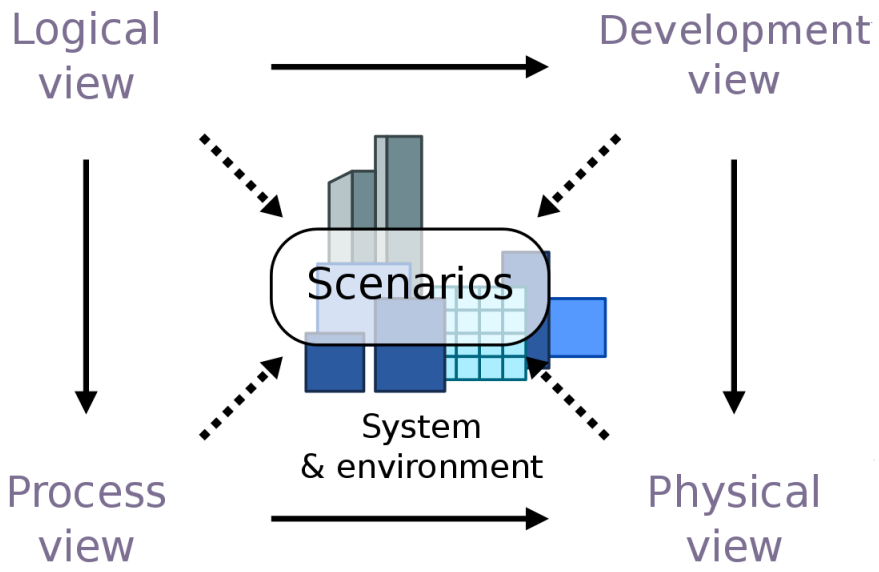
They also serve as a starting point for tests of an architecture prototype.

This view is also known as the **Use Case** view.

Logical View (*End user*)

The logical view is concerned with the functionality that the system provides to end-users.

UML diagrams are used to represent the logical view, and include **Class diagrams**, **Object diagram** and **State diagrams**.

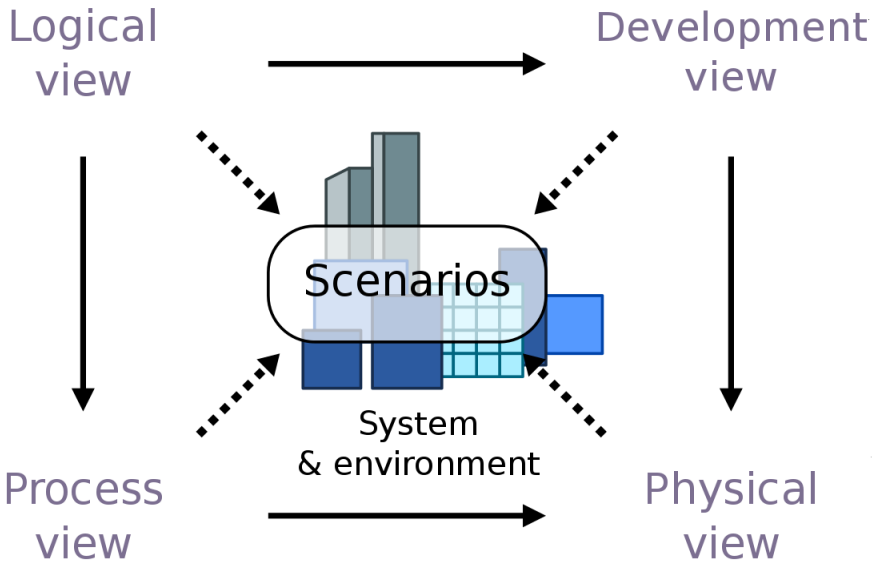


Process View (*Integrator*)

The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system.

The process view addresses concurrency, distribution, integrator, performance, and scalability, etc.

UML diagrams to represent process view include the **Sequence diagram, Communication diagram, Activity diagram**.

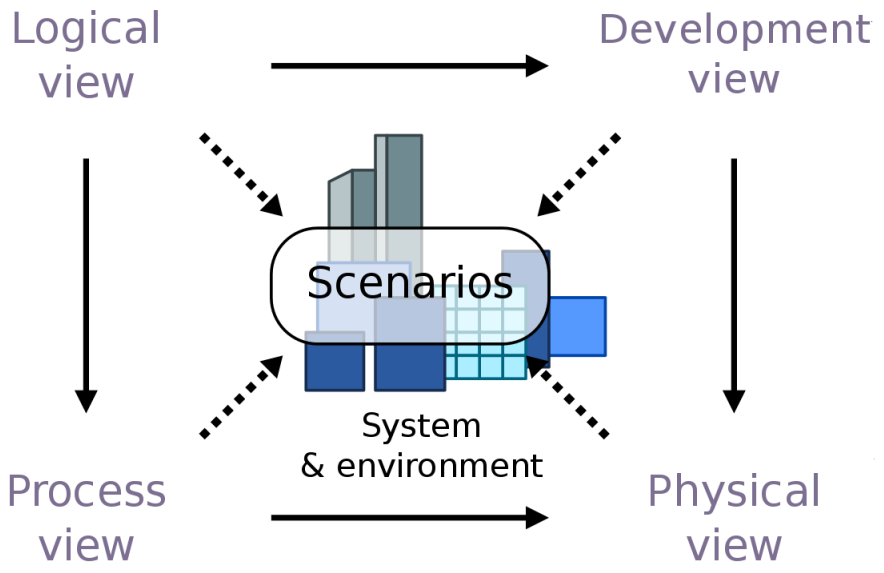


Development View (*Programmers*)

The development view illustrates a system from a programmer's perspective and is concerned with software management.

This view is also known as the implementation view. It uses the UML Component diagram to describe system components.

UML Diagrams used to represent the development view include the **Package diagram** and **Component diagram**.

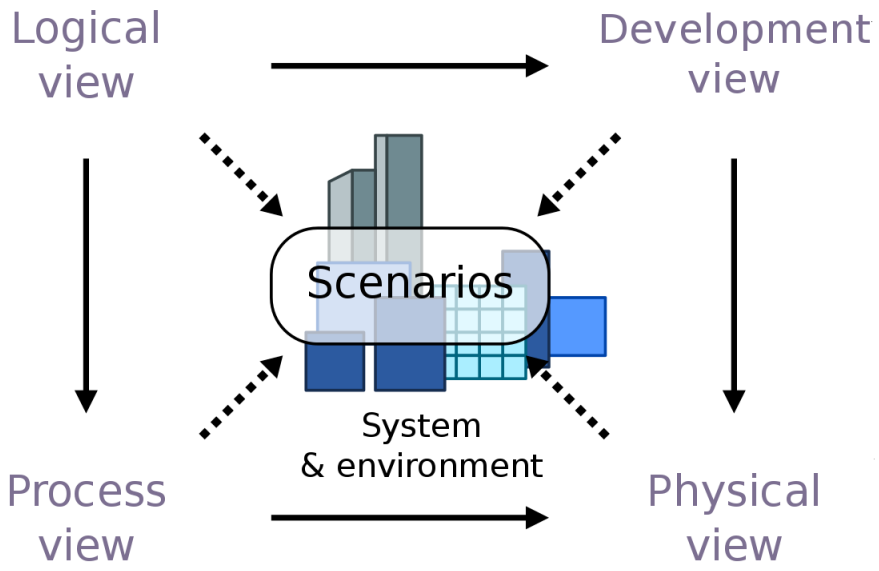


Physical View (*System engineer*)

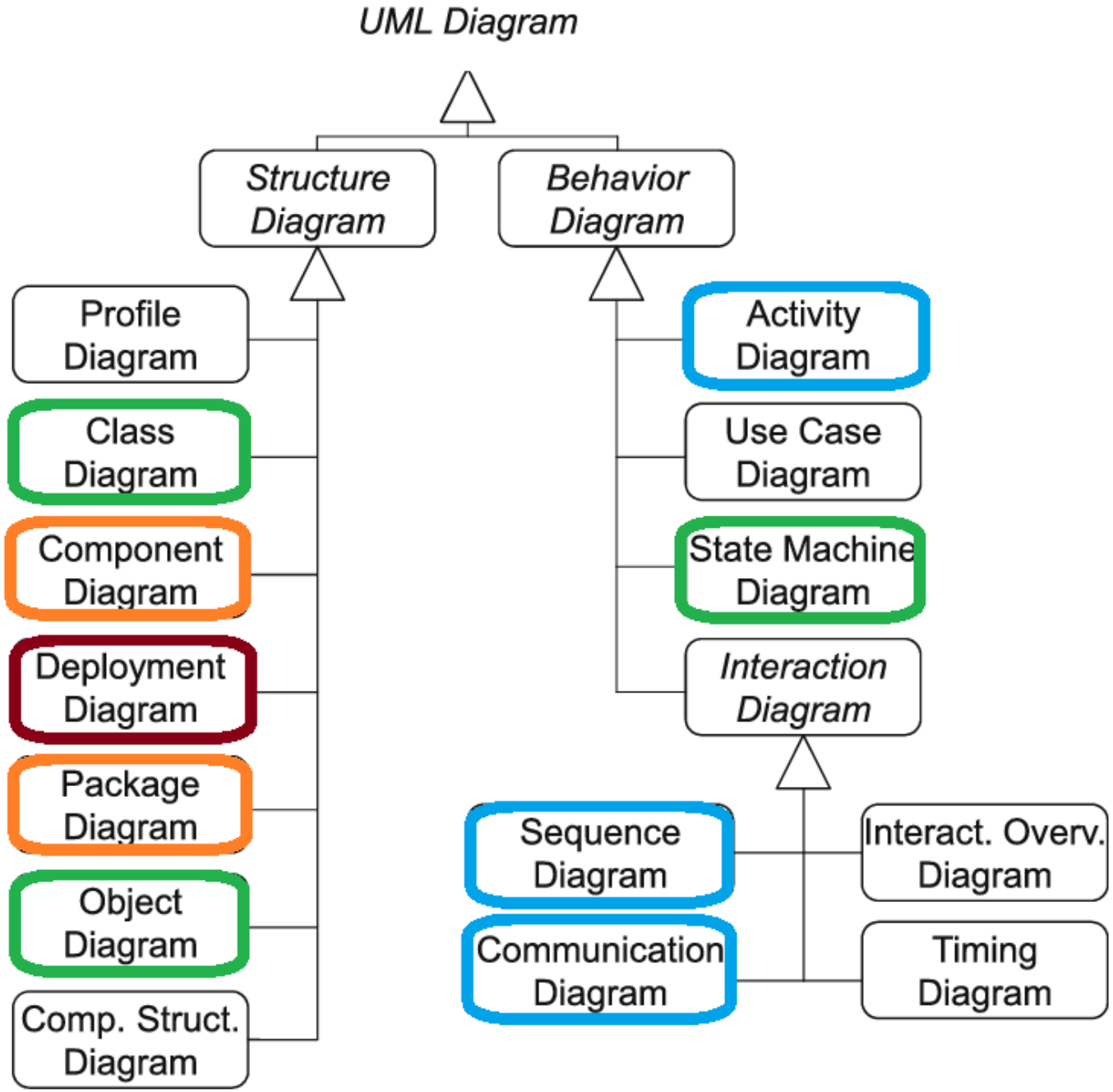
The physical view depicts the system from a system engineer's point of view.

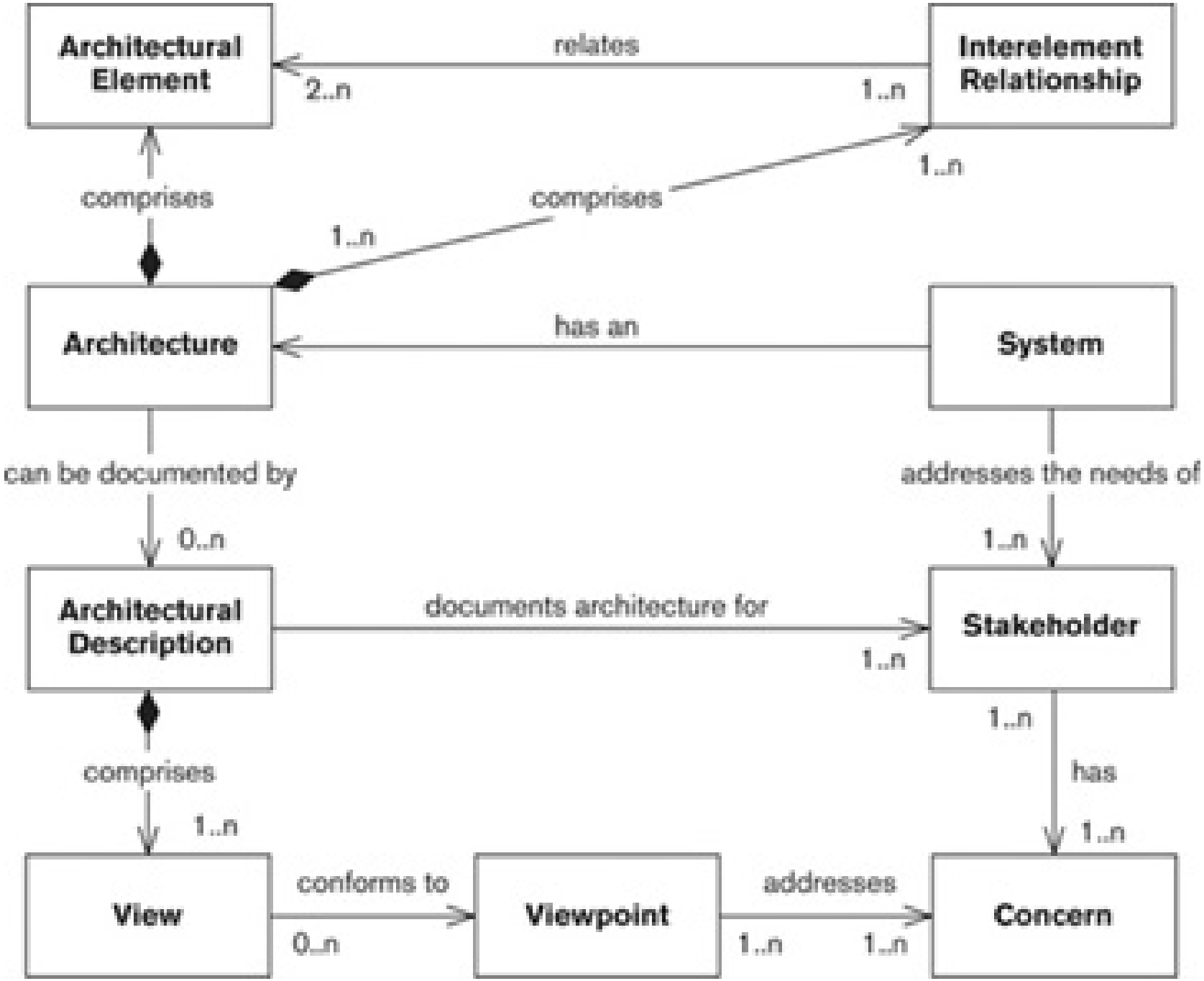
It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. This view is also known as the deployment view.

UML diagrams used to represent the physical view include the **Deployment diagram**.



- Logical - *Green*
- Process - *Blue*
- Development - *Orange*
- Physical - *Brown*




Architectural Element
relates
2..n
1..n
Interelement Relationship
comprises
1..n
comprises
1..n
Architecture
has an
System
can be documented by
0..n
addresses the needs of
1..n
Architectural Description
documents architecture for
1..n
Stakeholder
comprises
1..n
1..n
has
1..n
View
conforms to
0..n
Viewpoint
addresses
1..n
1..n
Concern

Architectural Concepts and Relationships

- A **system** is built to address the needs, concerns, goals and objectives of its **stakeholders**.
- The **architecture** of a **system** is characterized by its static and dynamic structures, and its externally-visible behavior and properties.
- The **architecture** of a **system** is comprised of a number of architectural **elements** and their interrelationships.
- The **architecture** of a **system** can potentially be documented by an **architectural description** (fully, partly or not at all). In fact, there are many potential ADs for a given architecture, some good, some bad.

- An **architectural description** documents an architecture for its **stakeholders**, and demonstrates to them that it has met their needs.
- A **viewpoint** defines the aims, intended audience, and content of a class of **views** and defines the concerns that views of this class will address.
- A **view** conforms to a **viewpoint** and so communicates the resolution of a number of concerns (and a resolution of a concern may be communicated in a number of views).
- An **architectural description** comprises a number of **views**.

ISO/IEC/IEEE 42010

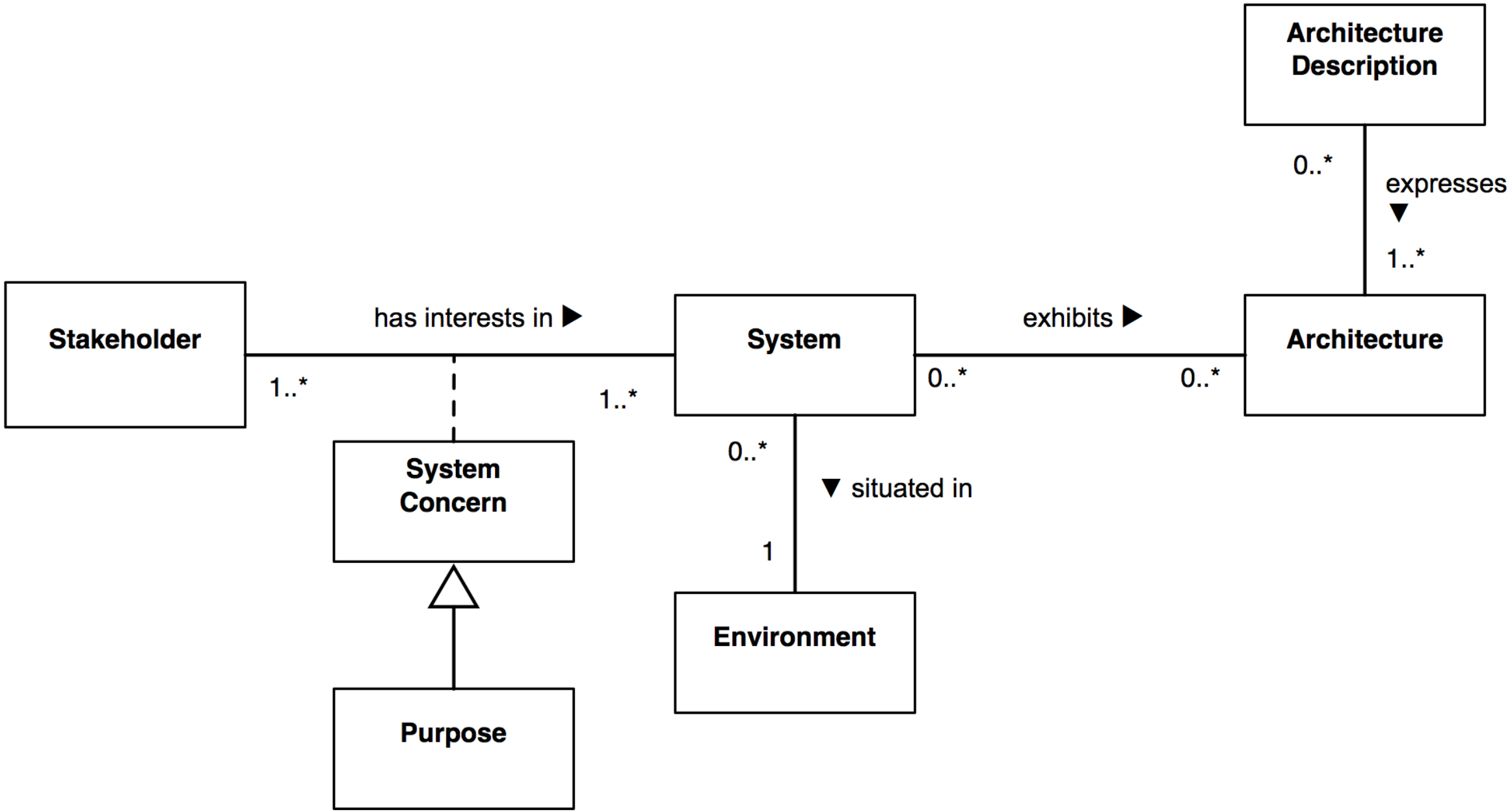
Conceptual Model of Architecture Description

A Conceptual Model of Architecture Description

ISO/IEC/IEEE 42010 is based upon a conceptual model – or *meta model* – of the terms and concepts pertaining to Architecture Description.

The conceptual model is presented in the Standard using UML class diagrams to represent classes of entities and their relationships.

http://www.iso-architecture.org/ieee-1471/cm/#:~:text=A Conceptual Model of Architecture,of entities and their relationships.

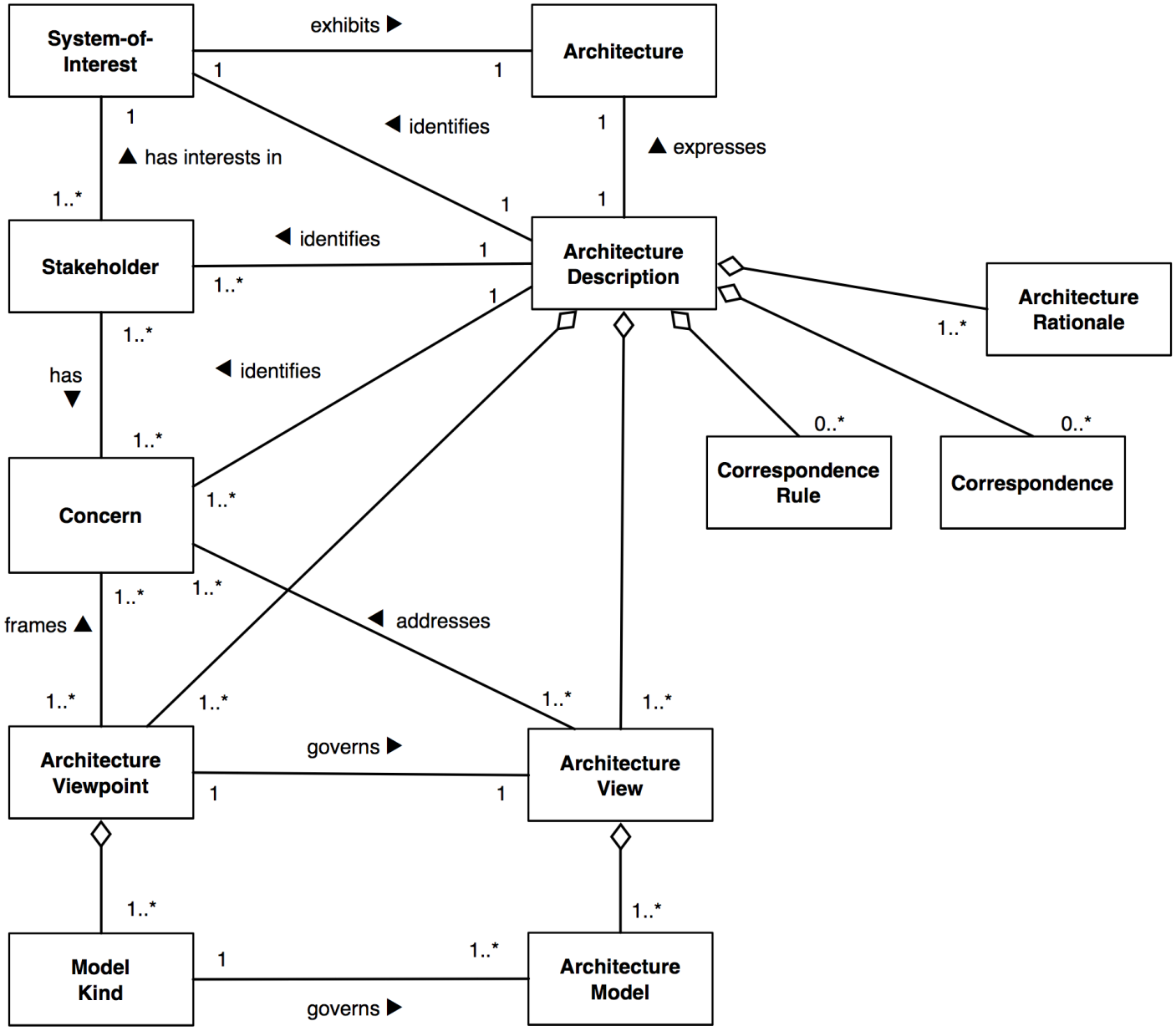

Architecture Description
0..*
expresses
1..*
Stakeholder
has interests in
System
exhibits
Architecture
1..*
1..*
0..*
0..*
System Concern
0..*
situated in
1
Environment
Purpose

- **Systems** is situated in its Environment. That environment could include other Systems
- **Stakeholders** have interests in a System; those interests are called Concerns
- An **Architecture Description** is used to express an Architecture of a System
- **System** is used as a placeholder – e.g., it could refer to an enterprise, a system of systems, a product line, a service, a subsystem, or software
- Every System inhabits its **Environment**
- Systems have architectures
- An **Architecture Description** (*AD*) is an artifact that expresses an Architecture

Core of AD

The Standard is organized around the terms and concepts of this diagram

It depicts the contents of an AD and the relations between those content items when applying the Standard to produce an Architecture Description to express an Architecture for some System of Interest.

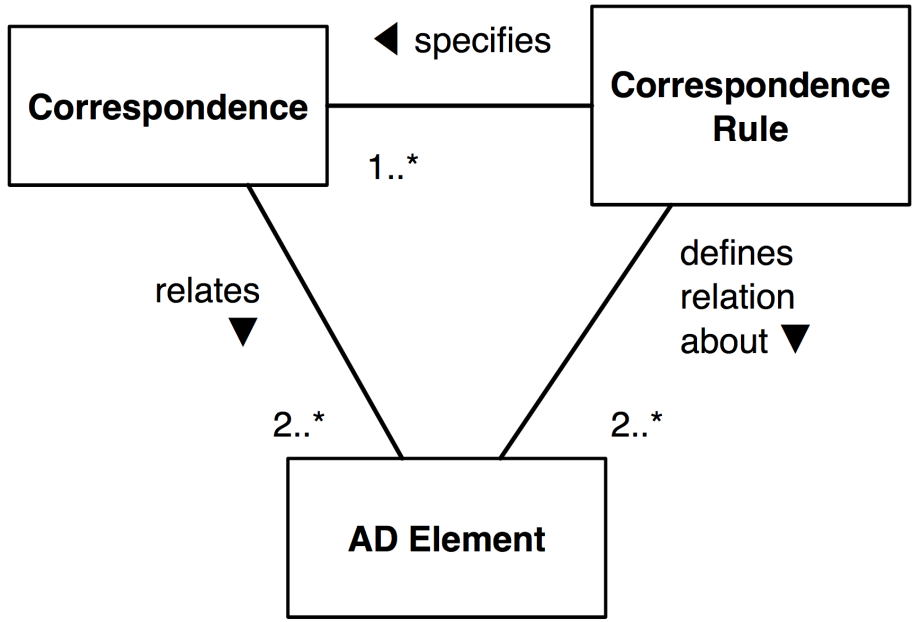


AD Elements and Correspondences

Architecture Descriptions are comprised of AD Elements.

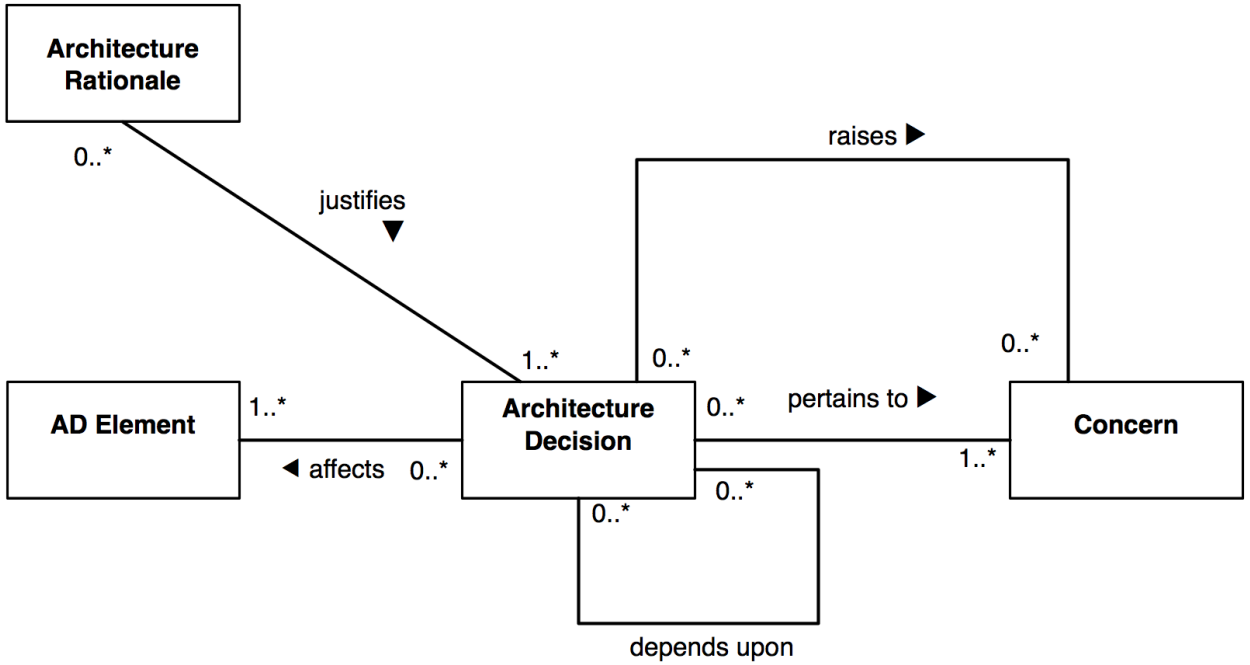
Correspondences capture relationships between AD Elements.

Correspondences and Correspondence Rules are used to express and enforce architecture relations such as composition, refinement, consistency, traceability, dependency, constraint and obligation within or between ADs.



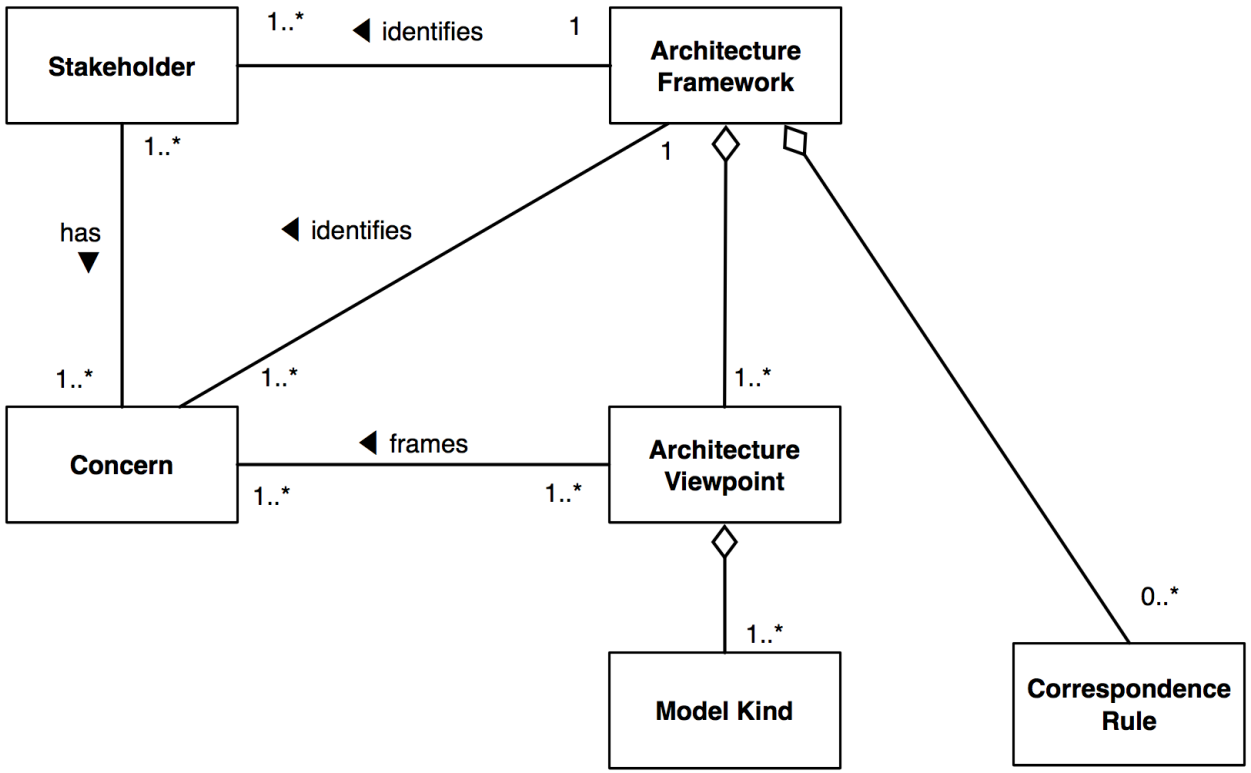
Architecture Decisions and Rationale

Architecture Decisions and Rationale
Creating an Architecture involves making Architecture Decisions.



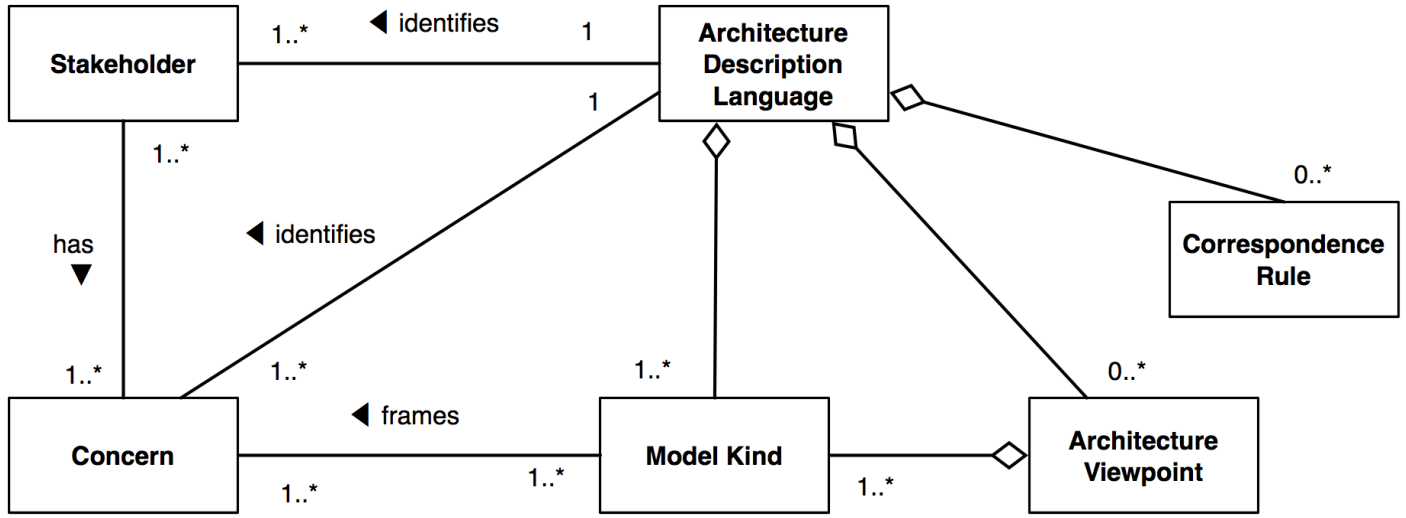
Architecture Frameworks and Architecture Description Languages

Architecture frameworks and architecture description languages (*ADL*) are two mechanisms widely used in architecting. Instances of each can be specified by building on the concepts of Architecture Description.



Architecture Framework

An architecture framework establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community.



4210 Template

- Architecture Viewpoint (VP) - 42010-vp-template.doc
- Architecture Description (AD) - 42010-ad-template.doc

Exercise – Inventory example

The following happens

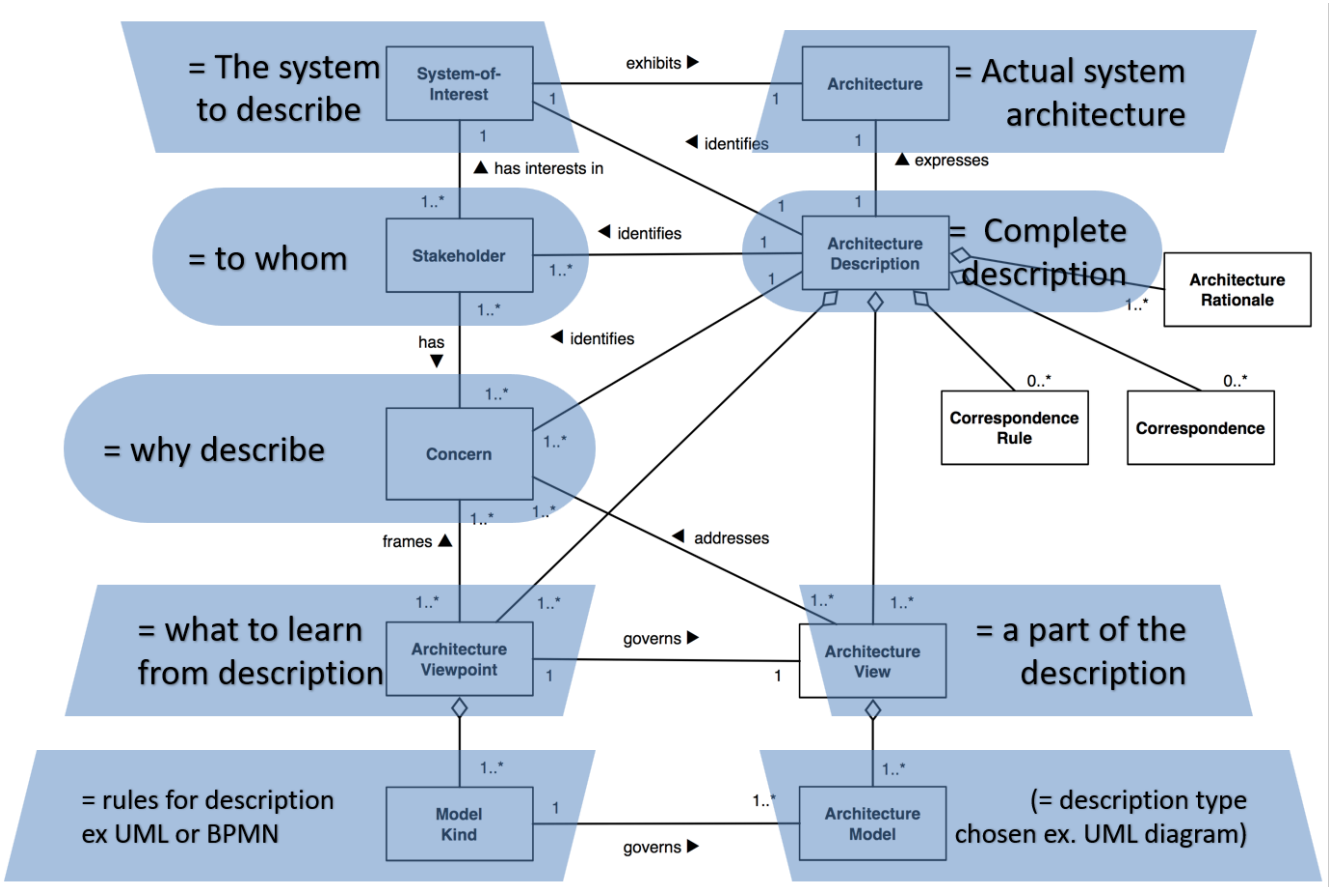
- A webstore displays an item showing 1 element in stock.
- The shopper puts it in the basket
- The shopper goes to exit/checkout and pays the order.
- A message is sent to the warehouse: ”pack and send item”
- An employee picks the last item from the stock
- Parcel is sent
- Stock-count is updated
- Invoice is sent

Exercise

Inventory example

As Architects You must

- Identify what IT components are actively participating in the scenario above.
- Decide: **Where in the system do you decide to store the stock-count?**



Quality Attributes - Non Functions Requirements

Architecture quality

Quality is how well an architecture satisfies **Functional** and **NON-Functional** requirements

NON-functional (*NFR*) requirements define the criteria that are used to evaluate the whole system, but not for specific behavior, and are also called quality attributes and described in detail in architectural specifications.

NFRs can be divided into two main categories:

- Attributes that affect system behavior, design, and user interface during work.
- Attributes that affect the development and support of the system.

Functional requirements

- Things that can be captured in a use-case.
- Things that can be analyzed in diagrams
- Most likely translate to code somewhere in a program

NON-functional requirements

- Development constraints like:
  - Development cost, time, operational cost, performance
- Many dynamic qualities like:
  - maintainability, testability, usability, etc.
  - Is seldom to be found in a single part of a program
  - Also known as Quality attributes

ISO / IEC FCD 25010 diagram

12 software architecture quality attributes

- **Performance** – shows the response of the system to performing certain actions for a certain period of time.
- **Interoperability** is an attribute of the system or part of the system that is responsible for its operation and the transmission of data and its exchange with other external systems.
- **Usability** is one of the most important attributes, because, unlike in cases with other attributes, users can see directly how well this attribute of the system is worked out.
- **Reliability** is an attribute of the system responsible for the ability to continue to operate under predefined conditions.

- **Availability** is part of reliability and is expressed as the ratio of the available system time to the total working time.
- **Security** is responsible for the ability of the system to reduce the likelihood of malicious or accidental actions as well as the possibility of theft or loss of information.
- **Maintainability** is the ability of the system to support changes.
- **Modifiability** determines how many common changes need to be made to the system to make changes to each individual item.

- **Testability** shows how well the system allows performing tests, according to predefined criteria.
- **Scalability** is the ability of the system to handle load increases without decreasing performance, or the possibility to rapidly increase the load.
- **Reusability** is a chance of using a component or system in other components/systems with small or no change.
- **Supportability** is the ability of the system to provide useful information for identifying and solving problems.

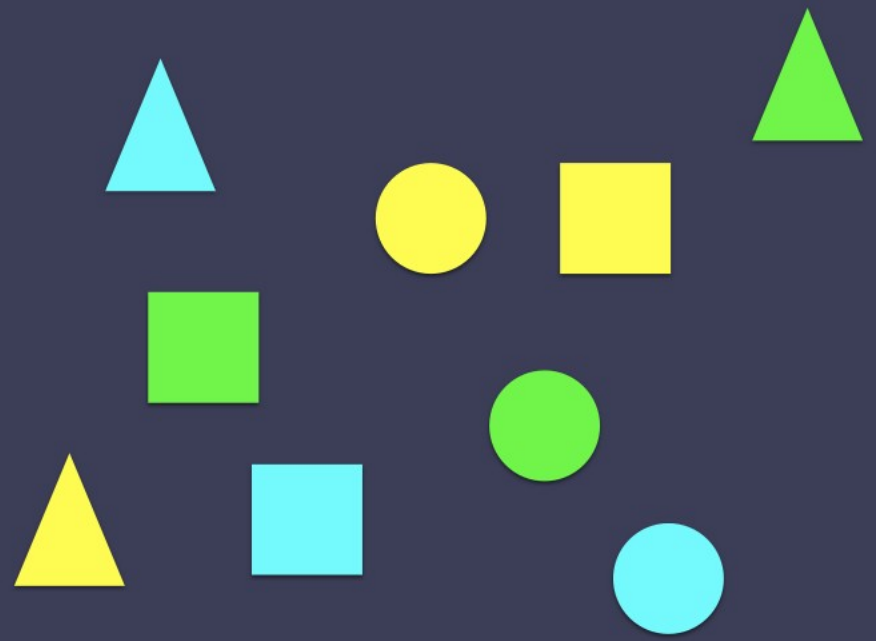
Domain vs Subdomain vs Bounded Context

An area of interest or an area over which a person has control

This definition of a domain is very fuzzy. What is an area of interest? It can be anything. A domain is effectively an arbitrary boundary around some subset of concepts in the universe.

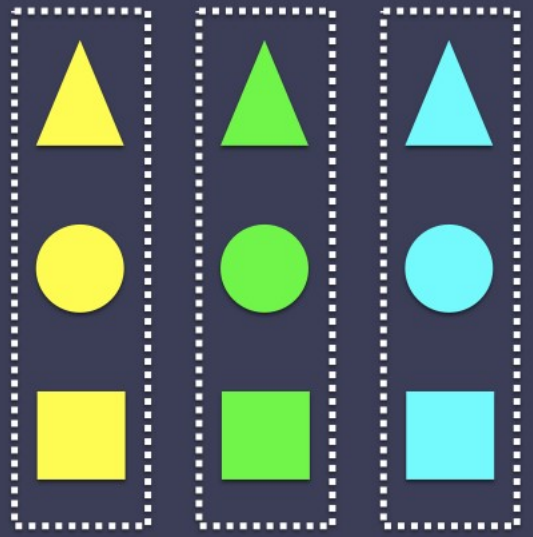
Cambridge Dictionary

HOW TO GROUP CONCEPTS?

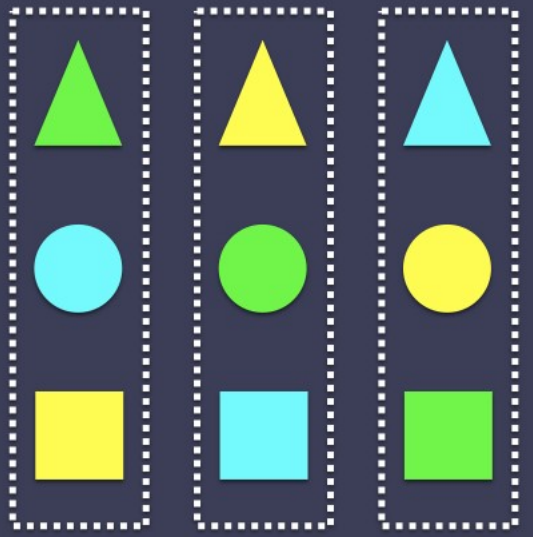


DEFINING DOMAIN BOUNDARIES

Shape Domains



Colour Domains



Other Possible Domains

Sub Domains

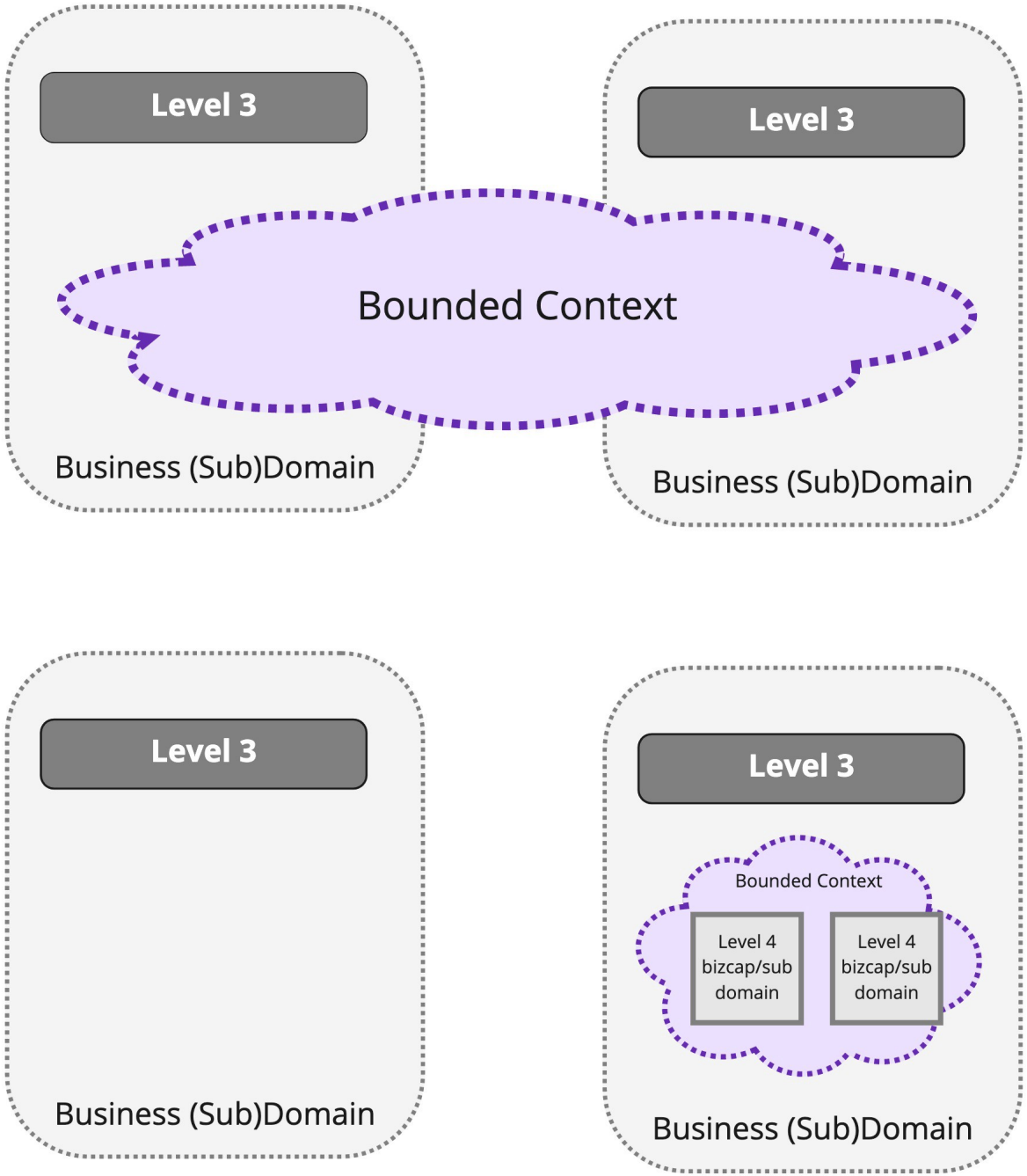
In DDD, a subdomain is a relative term.

Domain and subdomain can be used interchangeably.

When we use the word subdomain, we are emphasizing that the domain we are talking about is a child of another higher-level domain which we have identified.

Every subdomain is, therefore, a domain, and most domains are a subdomain.

Subdomain vs Bounded Context


Level 3
Level 3
Bounded Context
Business (Sub)Domain
Business (Sub)Domain
Level 3
Level 3
Bounded Context
Level 4 bizcap/sub domain
Level 4 bizcap/sub domain
Business (Sub)Domain
Business (Sub)Domain

Aggregates

Aggregate is a pattern in Domain-Driven Design.

A DDD aggregate is a cluster of domain objects that can be treated as a single unit.

An example may be an order and its line-items, these will be separate objects, but it's useful to treat the order (together with its line items) as a single aggregate.