UNIVERSIDAD
NACIONAL
DE COLOMBIA

Software Architecture

Architectural Views

Jeisson Andrés Vergara Vargas
Departamento de Ingeniería de Sistemas e Industrial
http://colswe.unal.edu.co/~javergarav/
javergarav@unal.edu.co

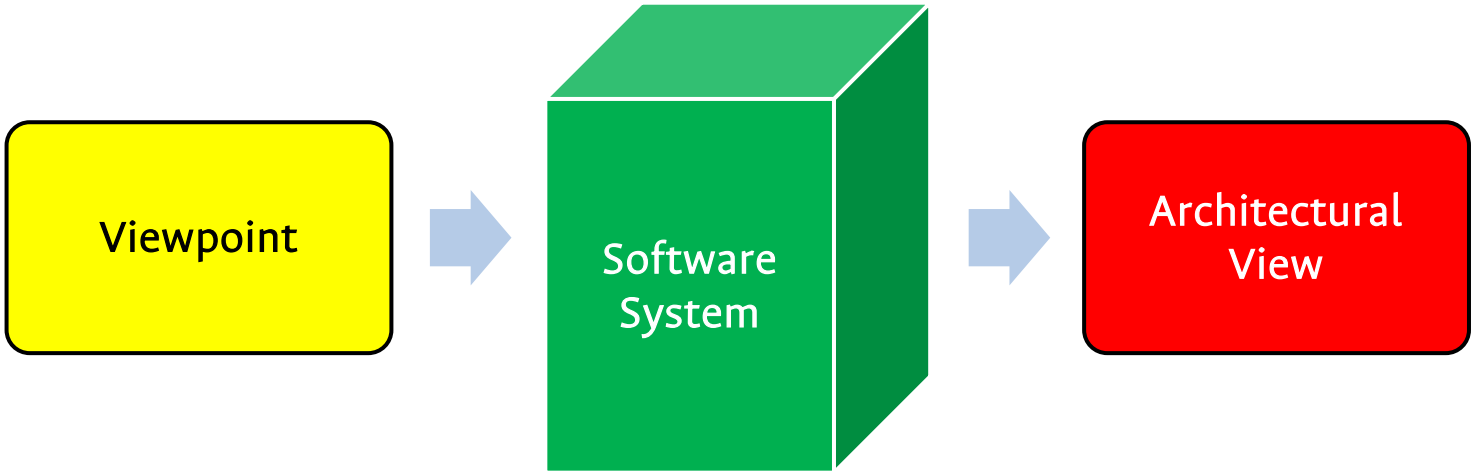
2021-I

©

Architectural Views

Definition

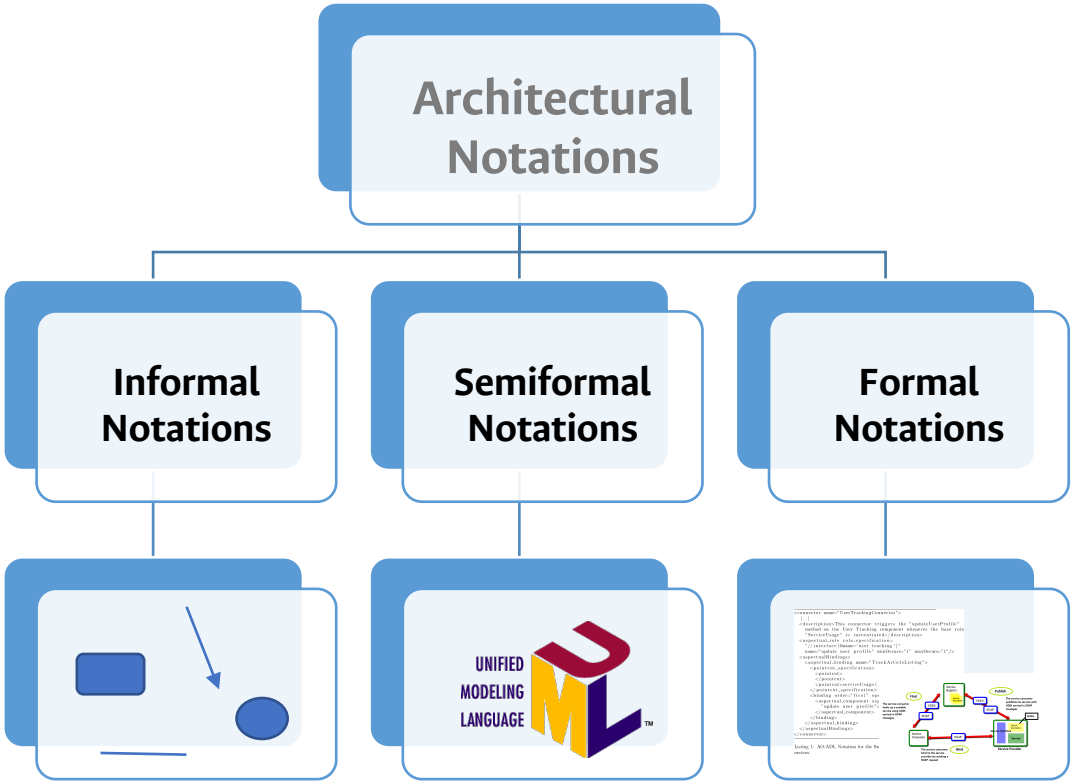
An **architectural view** (software architecture view) is a **representation** of a set of **system elements** and the **relationships** associated with them.



A **viewpoint** is a collection of **patterns**, **templates**, and **conventions** for constructing one type of **view**.

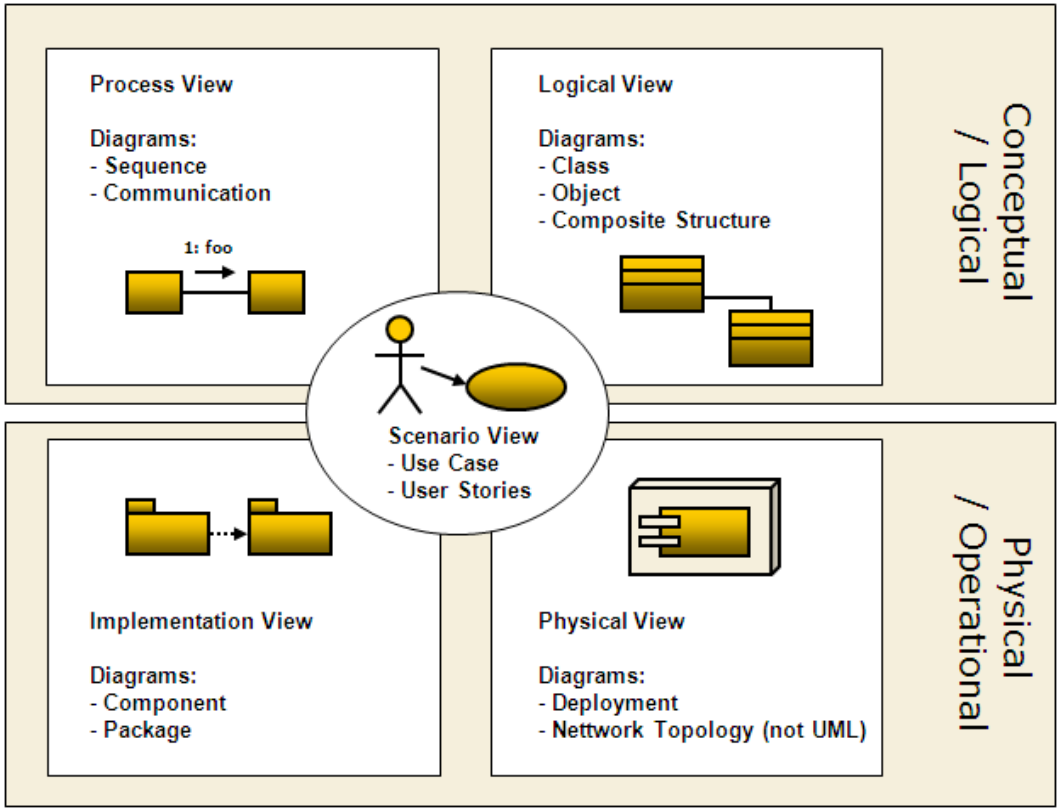
Architectural Views

Notations



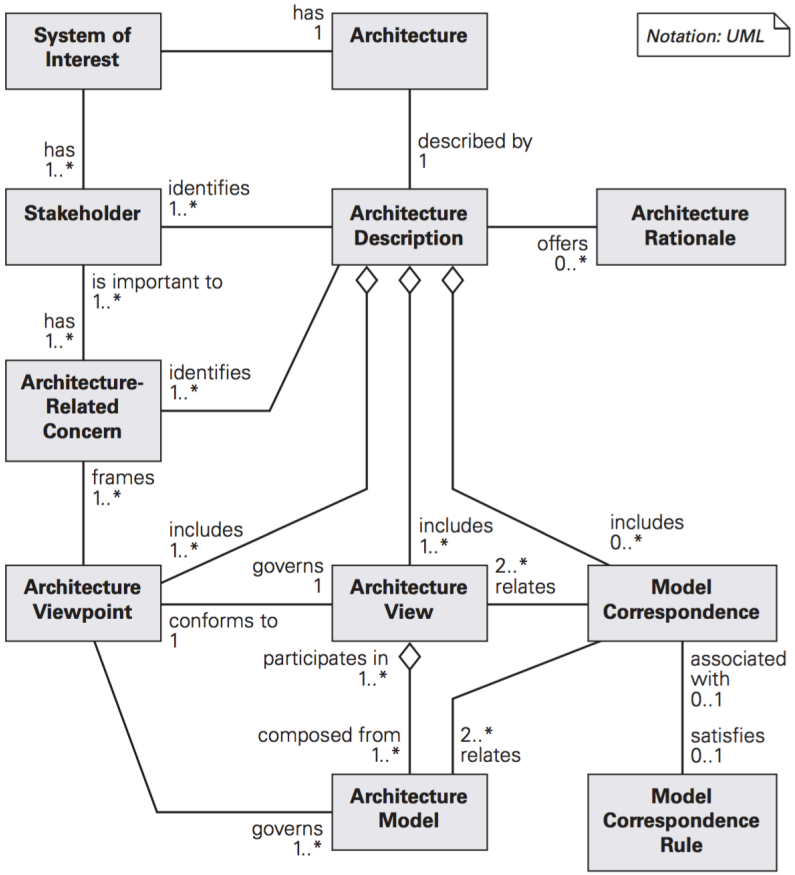
Models

4+1 View Model (Kruchten's 4+1)



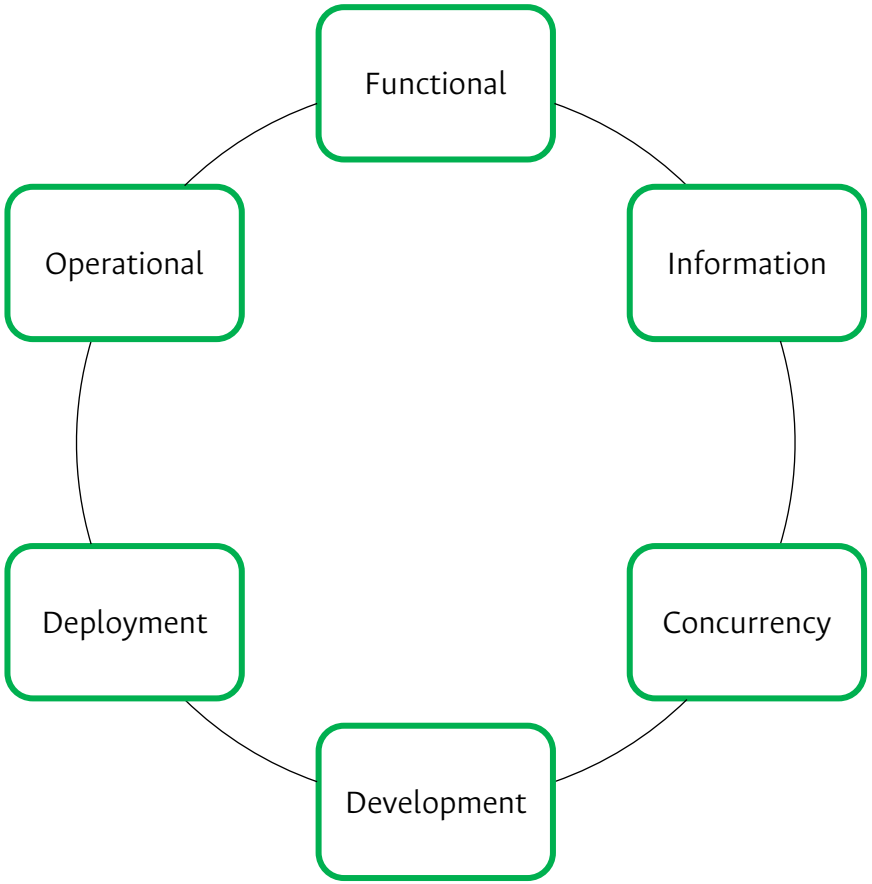
Models

ISO/IEC/IEEE 42010 (Systems and Software Engineering – Architecture Description)



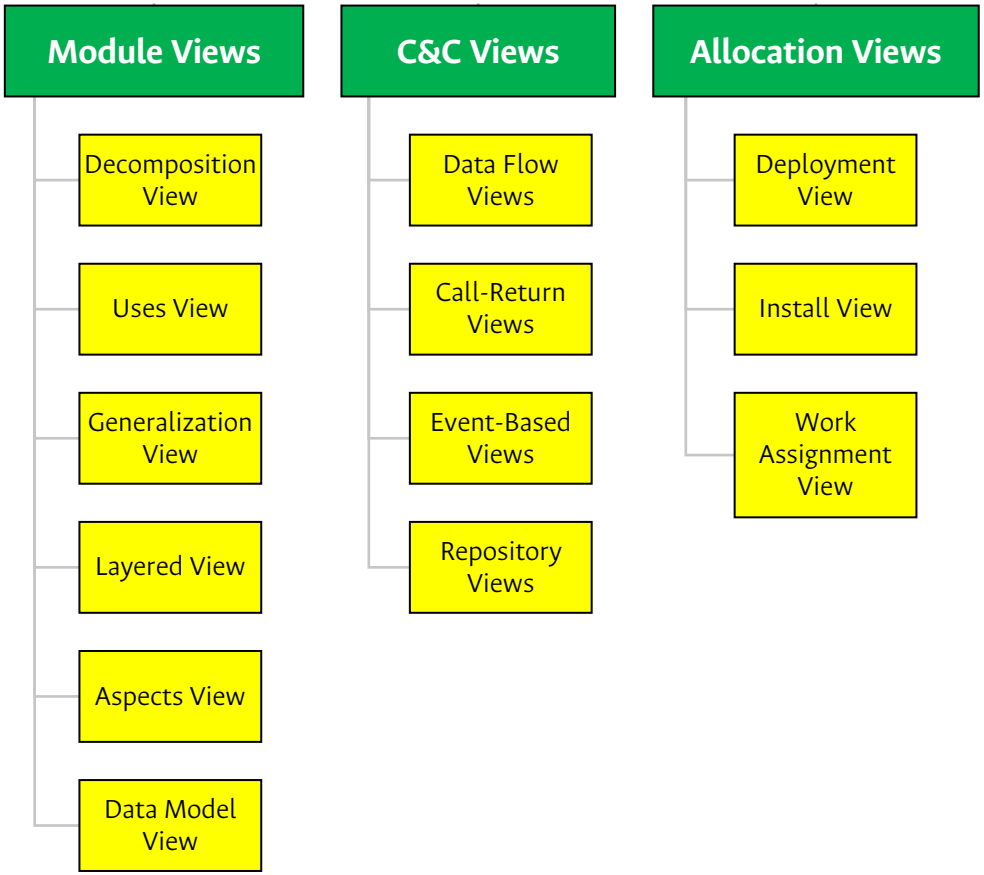
Models

The Viewpoint Catalog



Models

The Views & Beyond Catalog



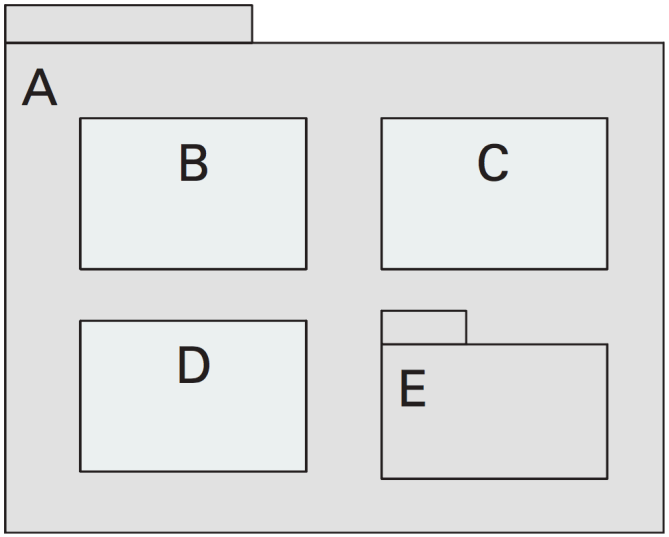
Decomposition View

The **decomposition view** is used for decomposing a system into **units** of **implementation**. This view describes the organization of the **code** as modules and submodules and shows how system responsibilities are partitioned across them.

Elements	**Modules**, **submodules** and **functionalities**.
Relations	**Is part of**: defines a part/whole relationship between the submodule (the **part**) and the aggregate module (the **whole**).
Constraints	• No loops are allowed in the decomposition graph. • A submodule can have only one parent (module or submodule).

Decomposition View

Uses	• To represent and describe the **functionalities** of a software system. • To reason about and communicate to newcomers the **structure** of software in digestible chunks. • To provide input for **work assignment**. • To reason about localization of **changes**.



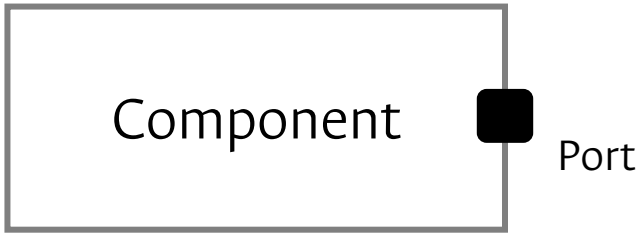
Component-and-Connector (C&C) View

The **component-and-connector (C&C) view** is used for describing **elements** that have some **runtime** presence, such as:

- Clients
- Servers
- Data Stores
- Processes
- Objects

Elements	**Components** and **connectors**.
Relations	**Attachment** and **interface delegation**.
Constraints	• **Components** can be **attached** only to connectors, not other components. • **Connectors** can be **attached** only to components, not other connectors. • **Attachments** can be made only between compatible ports and roles. • **Interface delegation** can be defined only between two compatible ports (or two compatible roles). • **Connectors** cannot appear in **isolation**; a connector must be attached to a component.

Component-and-Connector (C&C) View

Elements	Components • Components represent the principal **computational elements** and **data stores** that are present at runtime. • Each component in a C&C view has a **name**. • The name should indicate the **intended function** of the component. • A component has a set of **ports** through which it interacts with other components.
	

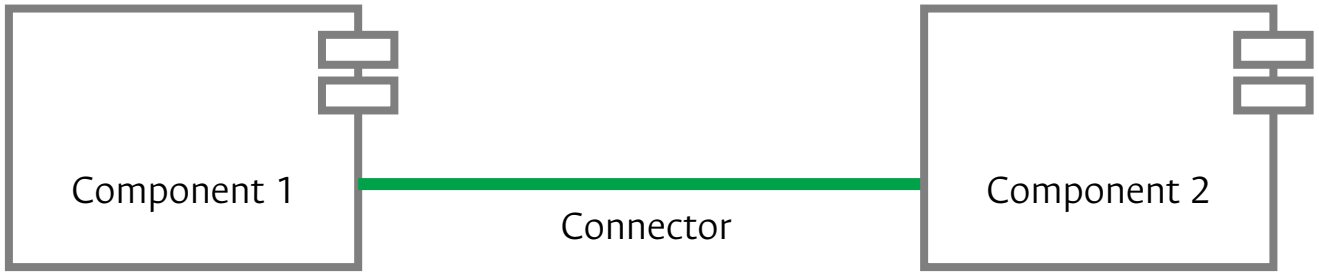
Component-and-Connector (C&C) View

Elements	Connectors • A connector is a **runtime pathway** of interaction between two or more components. • Connectors have **roles**, which are its **interfaces**, defining the ways in which the connector may be used by components to carry out interaction.
	

Component-and-Connector (C&C) View

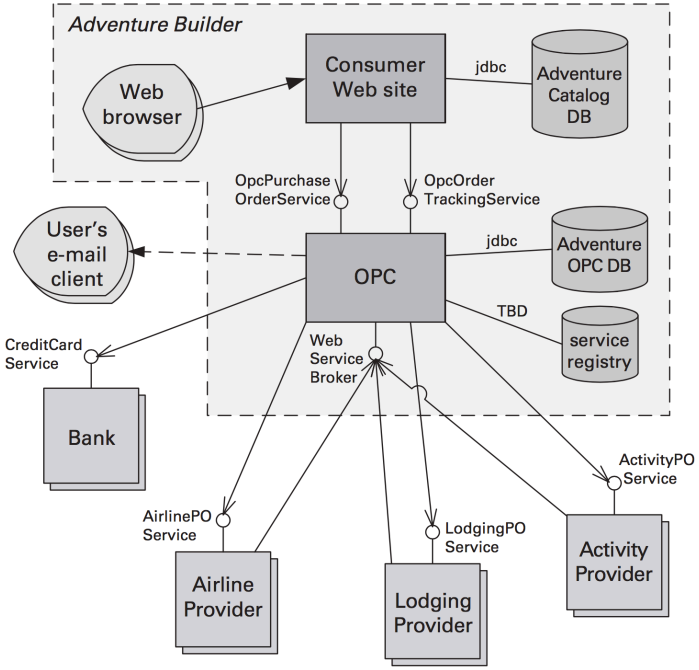
Relations	Attachment
	• Attachments indicate which connectors are **attached** to which components, thereby defining a system as a graph of components (**nodes**) and connectors (**arcs**). • Specifically, an attachment is denoted by **associating** (attaching) a component's **port** to a connector's **role**.
	Interface Delegation
	• When a component or connector has a **sub-architecture**. • To represent the relationship between the **internal structure** and the **external interfaces** of a component or connector.

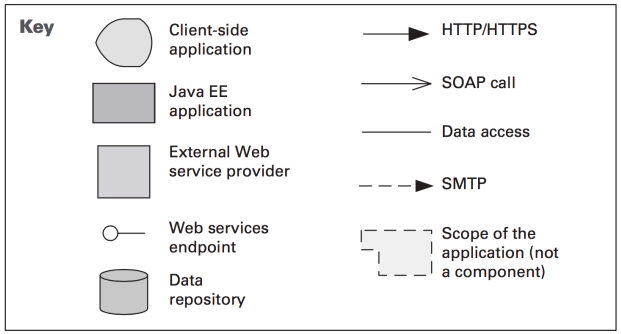
Component-and-Connector (C&C) View



Component-and-Connector (C&C) View

Uses	• To show developers and other stakeholders how the **system works**. • To specify the **structure** and **behavior** of the **runtime elements**. • To allow architects to **predict** overall **system properties**.





Data Model View

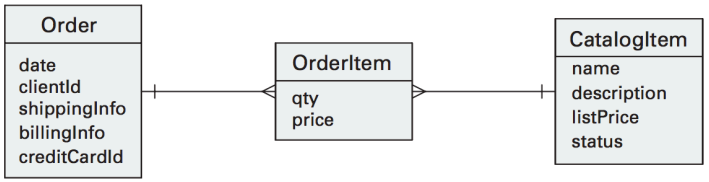
The **data model view** describes the structure of the data elements and their relationships.

Elements	**Data entity**.
Relations	• **One-to-one**, **one-to-many**, **many-to-one** and **many-to-many** relationships, which are logical associations between data entities. • **Generalization/specialization**, which indicate an *is-a* relation between data entities. • **Aggregation**, which turns a relationship into an aggregate data entity. • **Composition**, strong aggregation.
Constraints	• Functional dependencies should be avoided.

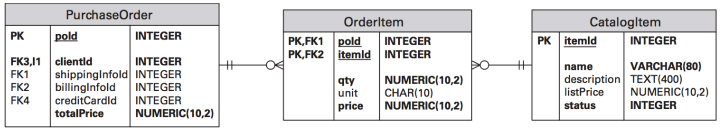
Data Model View

Uses	• Describing the **structure** of the data used in the system. • Performing impact analysis of **changes** to the data model; extensibility analysis. • Enforcing data **quality** by avoiding redundancy and inconsistency. • Guiding **implementation** of modules that access the data.

Conceptual



Logical



Physical

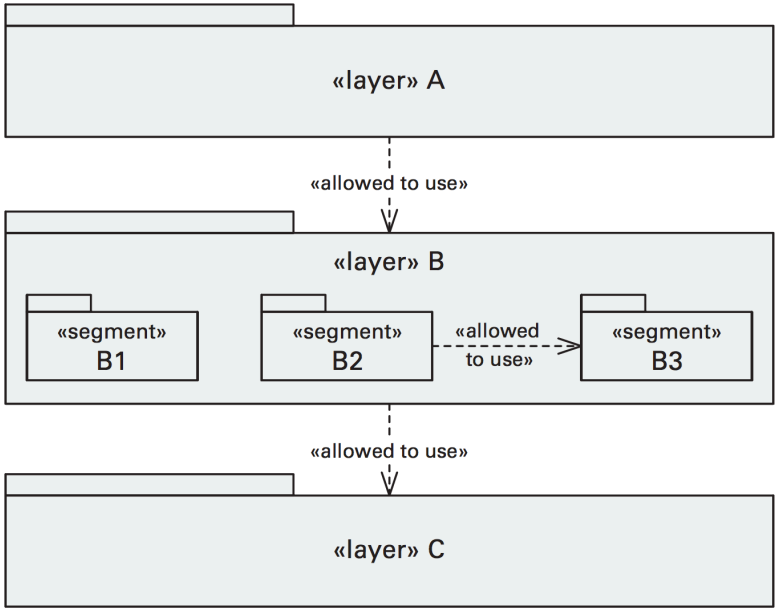
Layered View

The **layered view** puts together layers (**groupings** of **code pieces** that offer a cohesive set of services) in a unidirectional *allowed-to-use* relation.

Elements	**Layers** and **layer segments**.
Relations	**Allowed-to-use** and **allowed-to-use-below**: specializations of the generic relation **depends-on** relation.
Constraints	• Every piece of software is **allocated** to exactly **one layer**. • There are at least two layers (typically three or more). • The allowed-to-use relations should not be circular (that is, a lower layer cannot use a layer above).

Layered View

Uses	• Promoting **modifiability** and **portability**. • Managing **complexity** and facilitating the **communication** of the code structure to developers. • Promoting **reuse**. • Achieving **separation** of concerns.



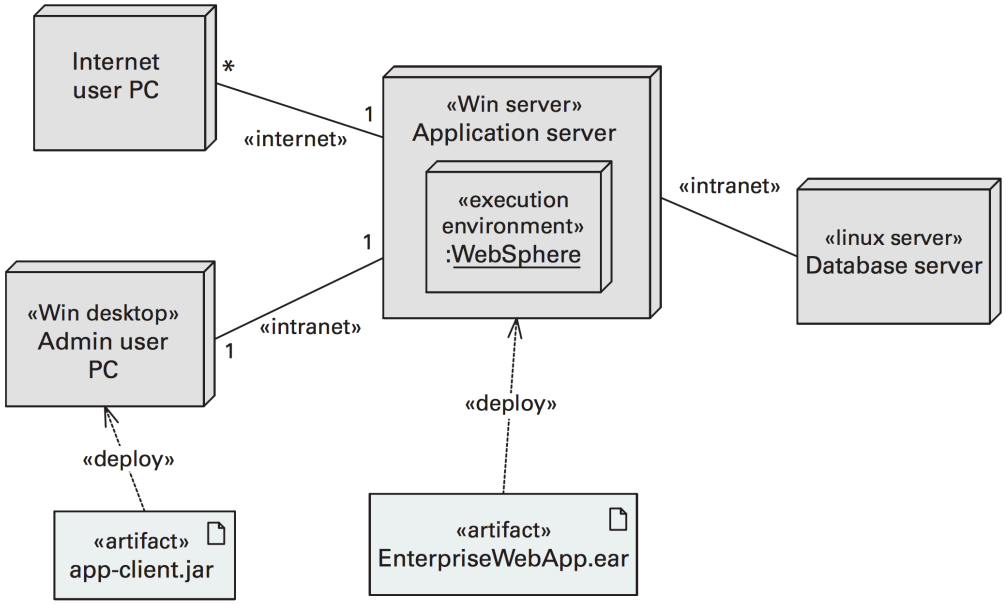
Deployment View

The **deployment view** describes the mapping between the **software's components** and **connectors** and the **hardware** of the computing platform on which the software executes.

Elements	**Software elements** and **environmental elements**.
Relations	**Allocated-to** (**deployed-in**): form that shows on which physical units the software elements reside at a given moment in time.
Constraints	The allocation topology is **unrestricted**. However, the required properties of the software must be satisfied by the provided properties of the hardware.

Deployment View

Uses	• To analyze **performance**, **availability**, **reliability** and **security**. • To understand **runtime dependencies**. • To support **cost estimation** when evaluating purchasing options for hardware.



References

- [**CLEMENTS**] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond. 2011.
- [**KRUCHTEN**] P. Kruchten, “Architectural Blueprints - The ”4+1” View Model of Software Architecture,” IEEE Softw., vol. 12, no. November, pp. 42–50, 1995.
- [**IEEE**] IEEE, ISO/IEC/IEEE: Systems and Software Engineering - Architecture Description. 2011.
- [**ROZANSKI**] N. Rozanski and E. Woods, Software Systems Architecture, 2nd ed. 2011.