Software Structures & Views

That is the strangest thing about the world: how it looks so different from every point of view.

Lauren Oliver

LECTURE 08
DR USMAN NASIR

Software structures

Software structure

- “The static structures of a software system define its internal design-time elements and their arrangement” and “the dynamic structures of a software system define its runtime elements and their interactions” (Bass et al, 2003)

- Traditionally there are three important categories of architectural structures
  - Module Structures
  - Component and Connector (C&C) structures
  - Allocation structures

- According to Bass *et al*, three software structures correspond to the three broad types of decisions that architectural design involves:
  - System to be structured as a set of code units (modules)
  - System to be structured as a set of elements that have runtime behaviour (components) and interactions (connectors)?
  - System’s relation to non-software structures in its environment? (ie. CPUs, file systems, networks, development teams, etc.)?

Module Structures

- Module structures are unit of implementation
  - Set of code or data units that must be constructed or procured.
  - Depend on programming/implementation language (could be class, function etc..).

- Module structures allow us to answer questions such as these:
  - What is the primary functional responsibility that is assigned to each module?
  - What other software elements a module is allowed to use?
  - What other software does a module it use and depends on?

Component-and-connector (C&C) structures

- Component-and-connector (C&C) structures focus on the way the elements interact with each other at runtime to carry out the system's functions.
  - How the system is structured as a set of elements that have runtime behavior (components) and interactions (connectors).

- C&C structures help answer questions such as the following:
  - What are the major executing components and how do they interact at runtime?
  - What are the major shared data stores?
  - How does data progress through the system?
  - Which parts of the system can run in parallel?

Allocation structures

- Allocation structures describe the mapping from software structures to the system’s non-software structures
  - E.g test, and execution environments.

- Allocation structures answer questions such as the following:
  - Which processor(s) does each software element execute on?
  - In which directories or files is each element stored during development, testing, and system building?
  - What is the assignment of each software element to development teams?

Views

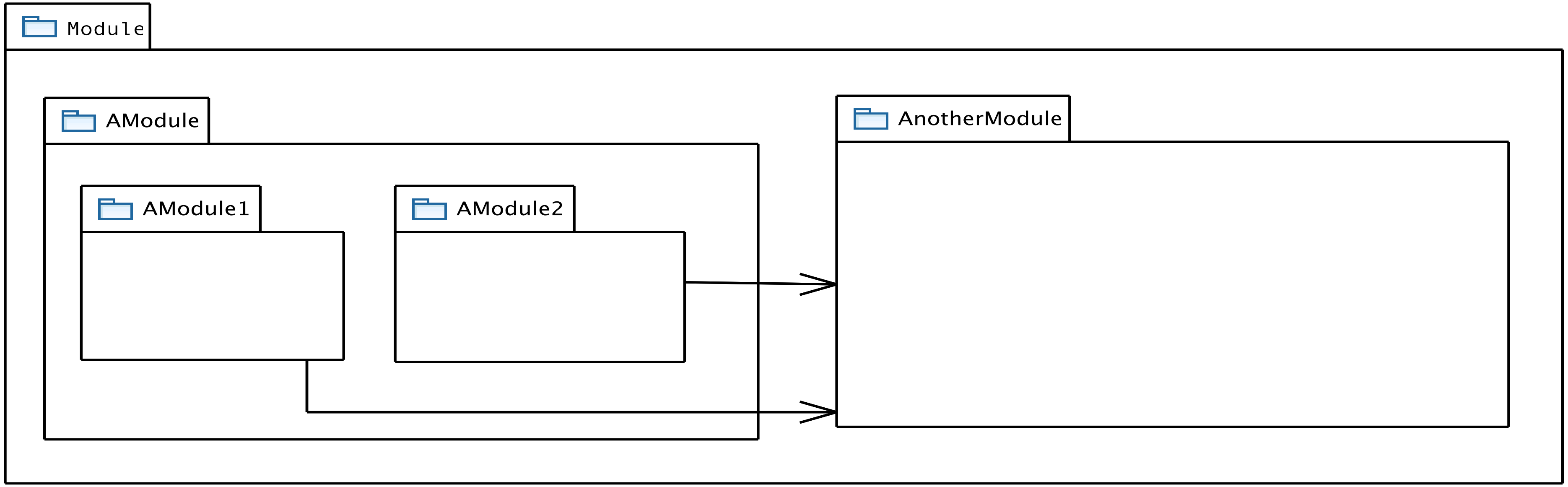
Views

- A view is a representation of a set of architectural elements for system stakeholders.
  - A view consists of a representation of a set of elements and the relations among them.
- A view is a representation of a structure.
  - For example:
    - Module structure is the set of the system's modules and their organization.
    - A module view is the representation of module structure, documented according to a template in a chosen notation, and used by some system stakeholders.

View Representation

- A view representation shows:
  - Elements
  - Relationships among the elements
  - Properties of elements and /or relations
  - Constraints
  - A selection criterion which specifies the elements, relations and properties we consider and the ones we exclude
    - This selection criteria should be clear and unambiguous

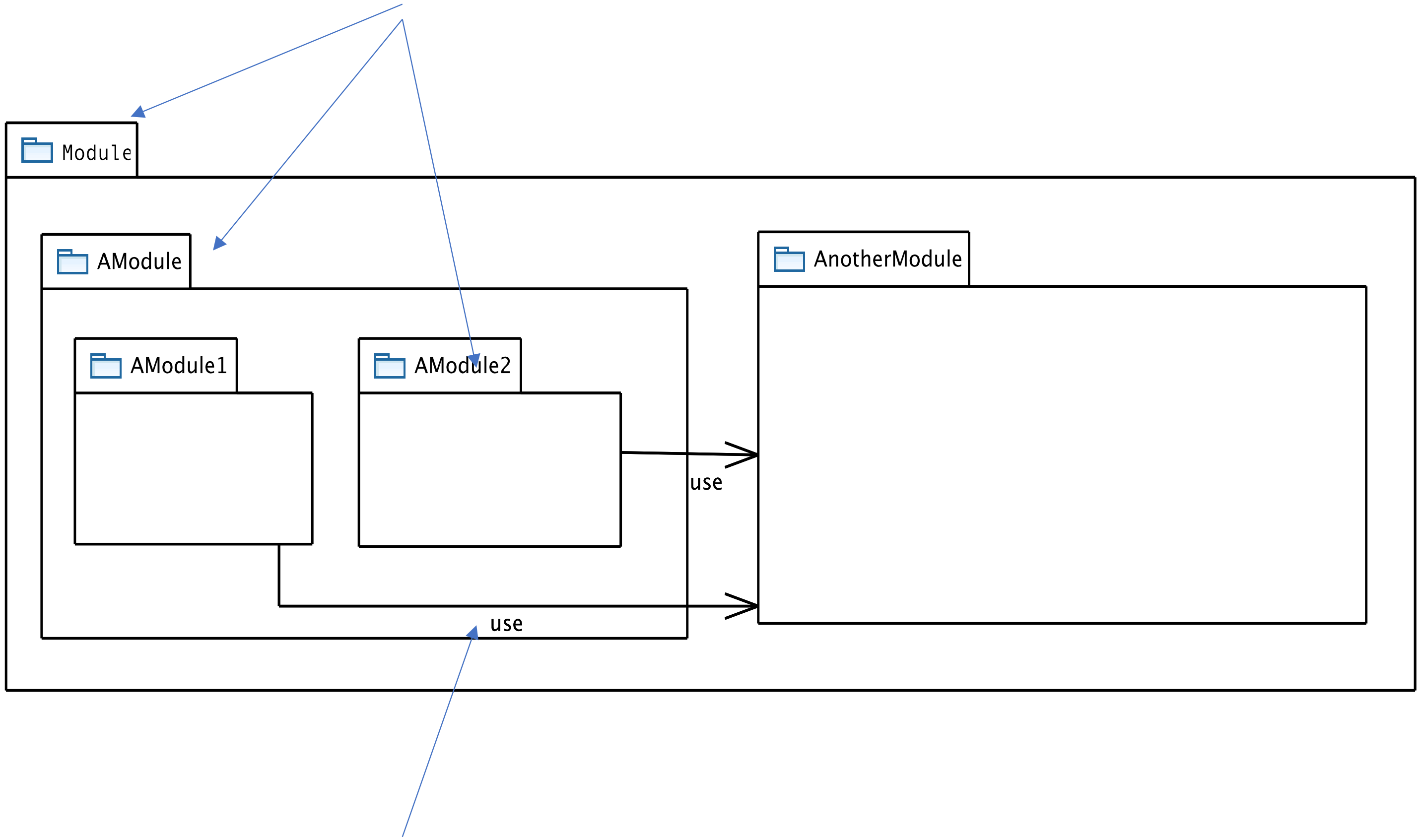
- To be useful, every view in the architecture documentation must include a key / legend that describes the meaning of all the symbols shown
  - In case of pure text views, an explanatory text and example can help as key / legend
  - Adding the legend to the view provides a clear and comprehensible semantics to the diagram

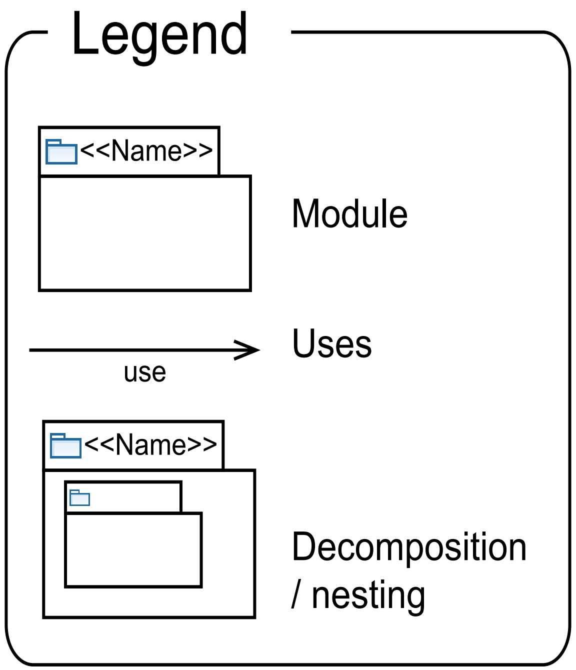


Do you find this diagram useful? What does the arrow mean?

- Example of a view diagram
    - Elements: **Modules**
    - Relationships: Uses, **Decomposition**
    - Properties: **Module Name**
    - Selection Criteria: **All modules on the 3 first levels**

Elements: Modules

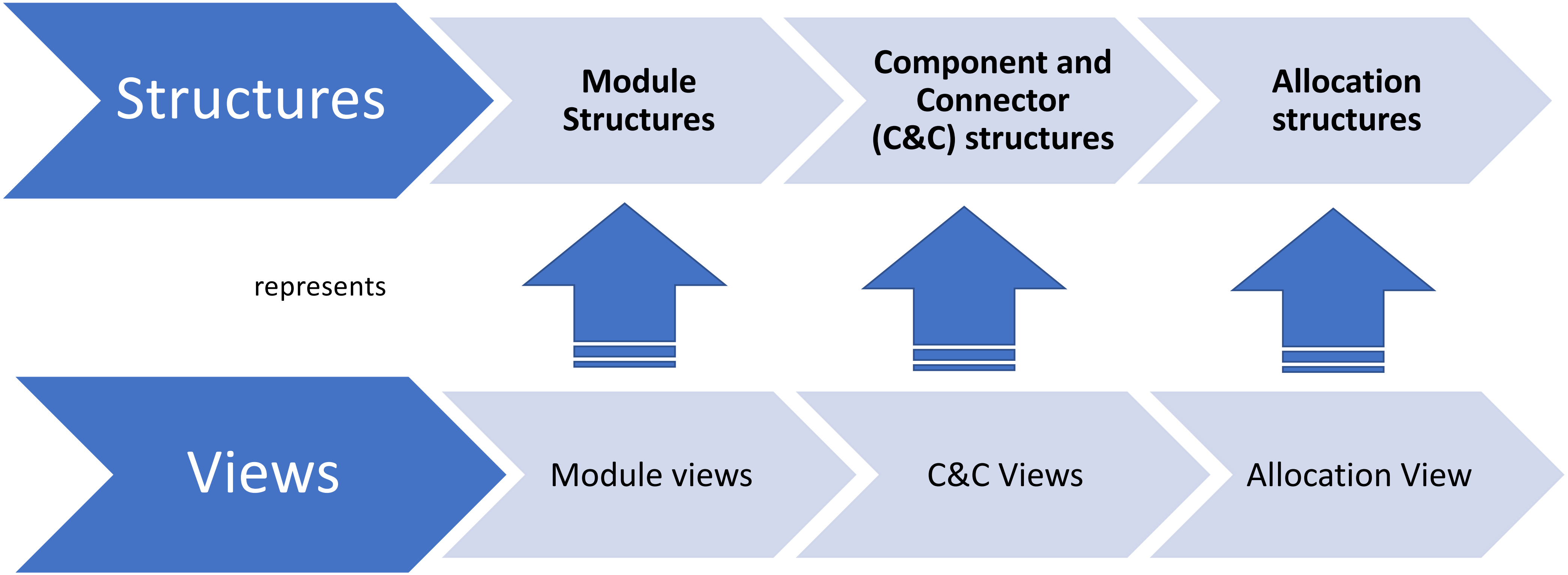




Relationships: Uses, Decomposition

- There are three structure-based views
  - Module views
  - Component and Connector (C&C) views.
  - Allocation views
- Quality views.

Three structure-based views



Module Views

- A module is an implementation unit (e.g., Classes + Interfaces)
  - Provides a coherent set of responsibilities.

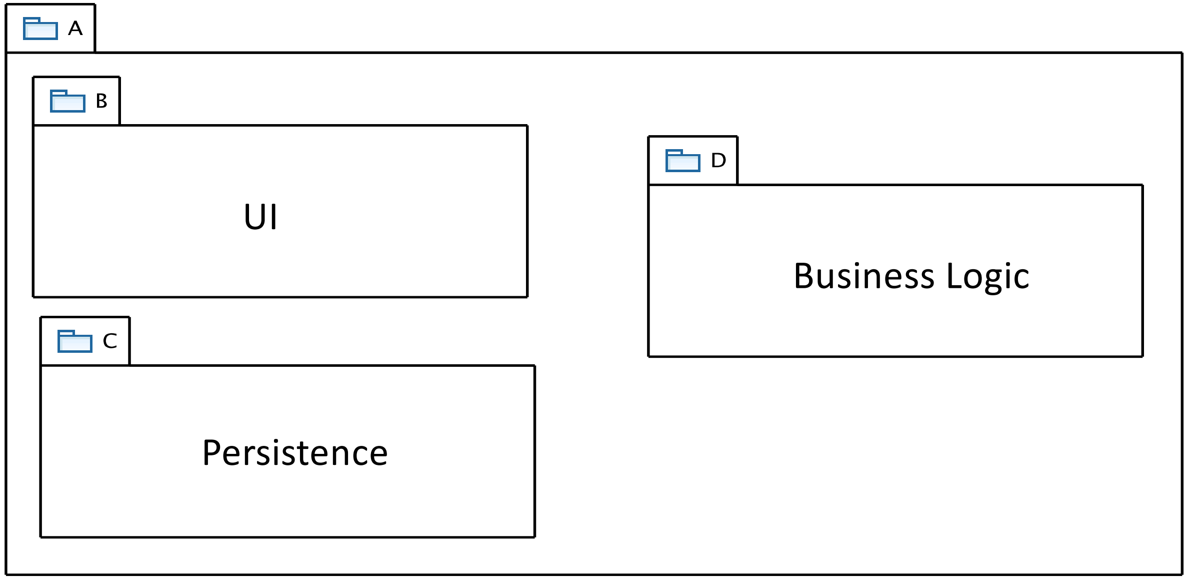
- Elements:
  - Modules: Implementation units (C programs, C++, Java or C# classes) and their grouping (Packages or Namespaces)
- Relationships:
  - Is Part of: Describes the part vs whole relationship between the module and submodule the part
  - Depends on: Dependency relationship between two elements. Uses, Allowed to Use are examples of depend on relationship
  - Is a: Generalization / Specialization

- Some styles of module views:
  - Decomposition view
  - Uses
  - Layers
  - Data model

Decomposition view

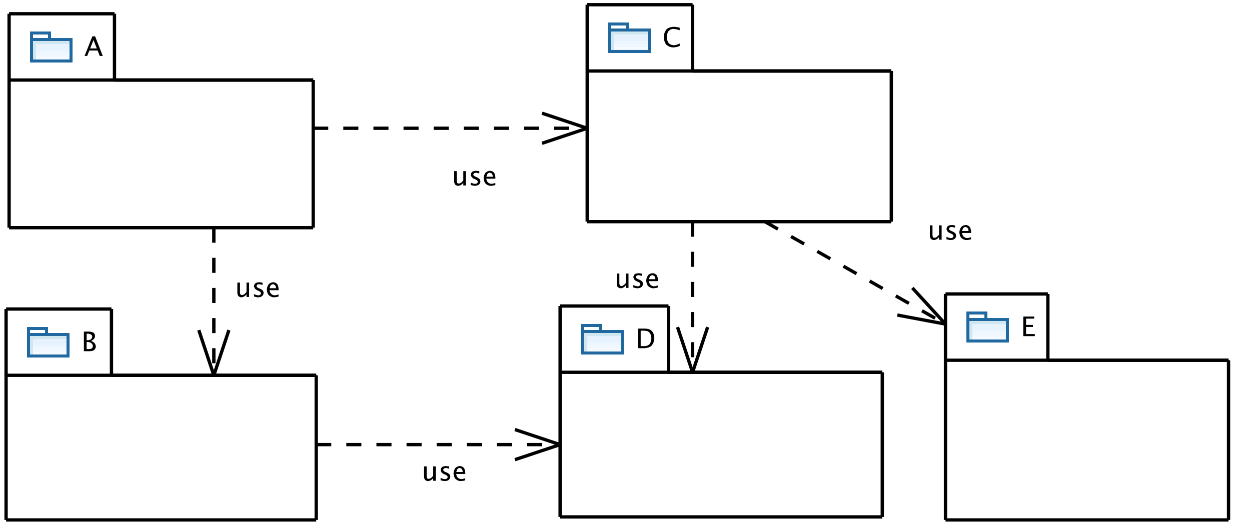
- The units are modules that are related to each other by the is-a submodule-of relation.
  - It shows how modules are decomposed into smaller modules recursively until the modules are small enough to be easily understood.
- The decomposition structure determines, to a large degree, the system's modifiability, by assuring that likely changes are localized.
- Provides good support for the understanding of the system
  - Focuses on the is_part_of relationship

- Example: Decomposition view showing how the system is organized into submodules
  - The responsibilities are partitioned across modules and submodules



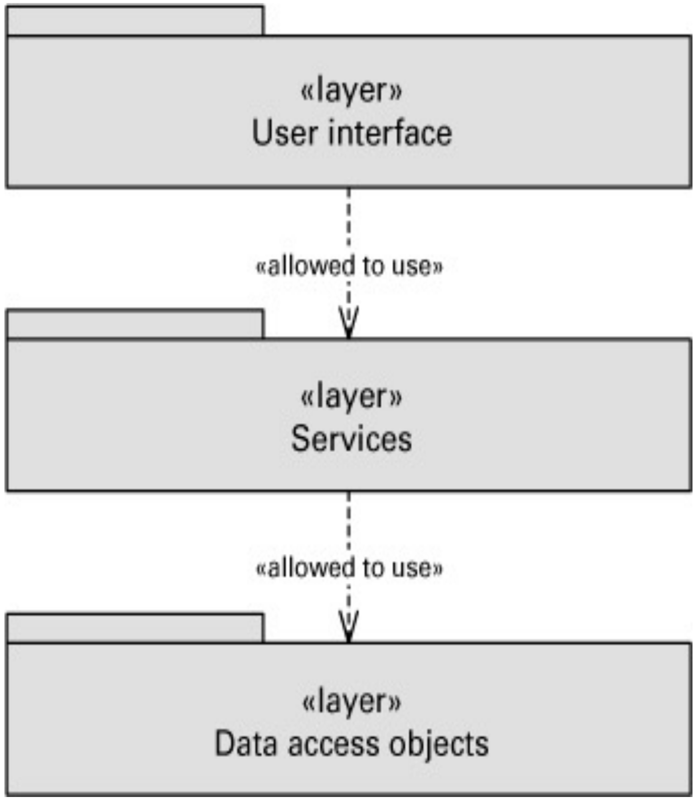
Uses

- This style describes the build dependencies and useful to plan incremental development, system extensions and test cases etc.
  - Focuses on the depends_on relationship



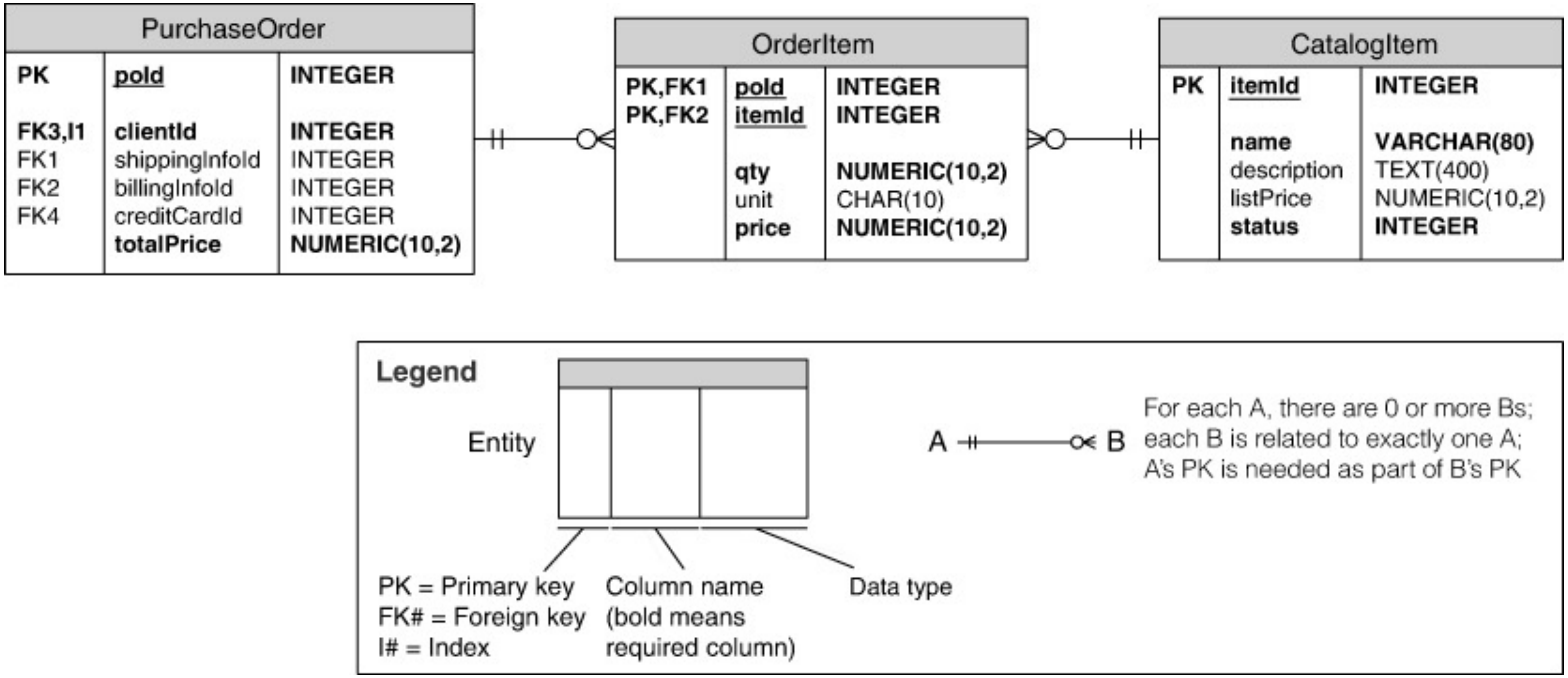
Layers

- A layer is a grouping of modules that together offer a set of services to other layers
- Layers completely partition the system and each partition, through interfaces
- The main relationship is the allowed_to_use
    - Strictly ordered
    - Unidirectional



Data model

- The data model describes the static information structure in terms of data entities and their relationships.



PurchaseOrder

PK	**pold**	**INTEGER**
FK3,I1	**clientId**	**INTEGER**
FK1	shippingInfold	INTEGER
FK2	billingInfold	INTEGER
FK4	creditCardId	INTEGER
	totalPrice	**NUMERIC(10,2)**

OrderItem

PK,FK1	**pold**	**INTEGER**
PK,FK2	**itemId**	**INTEGER**
	qty	**NUMERIC(10,2)**
	unit	CHAR(10)
	price	**NUMERIC(10,2)**

CatalogItem

PK	**itemId**	**INTEGER**
	name	**VARCHAR(80)**
	description	TEXT(400)
	listPrice	NUMERIC(10,2)
	status	**INTEGER**

Legend

Entity

PK = Primary key
FK# = Foreign key
I# = Index

Column name (bold means required column)

Data type

For each A, there are 0 or more Bs; each B is related to exactly one A; A's PK is needed as part of B's PK

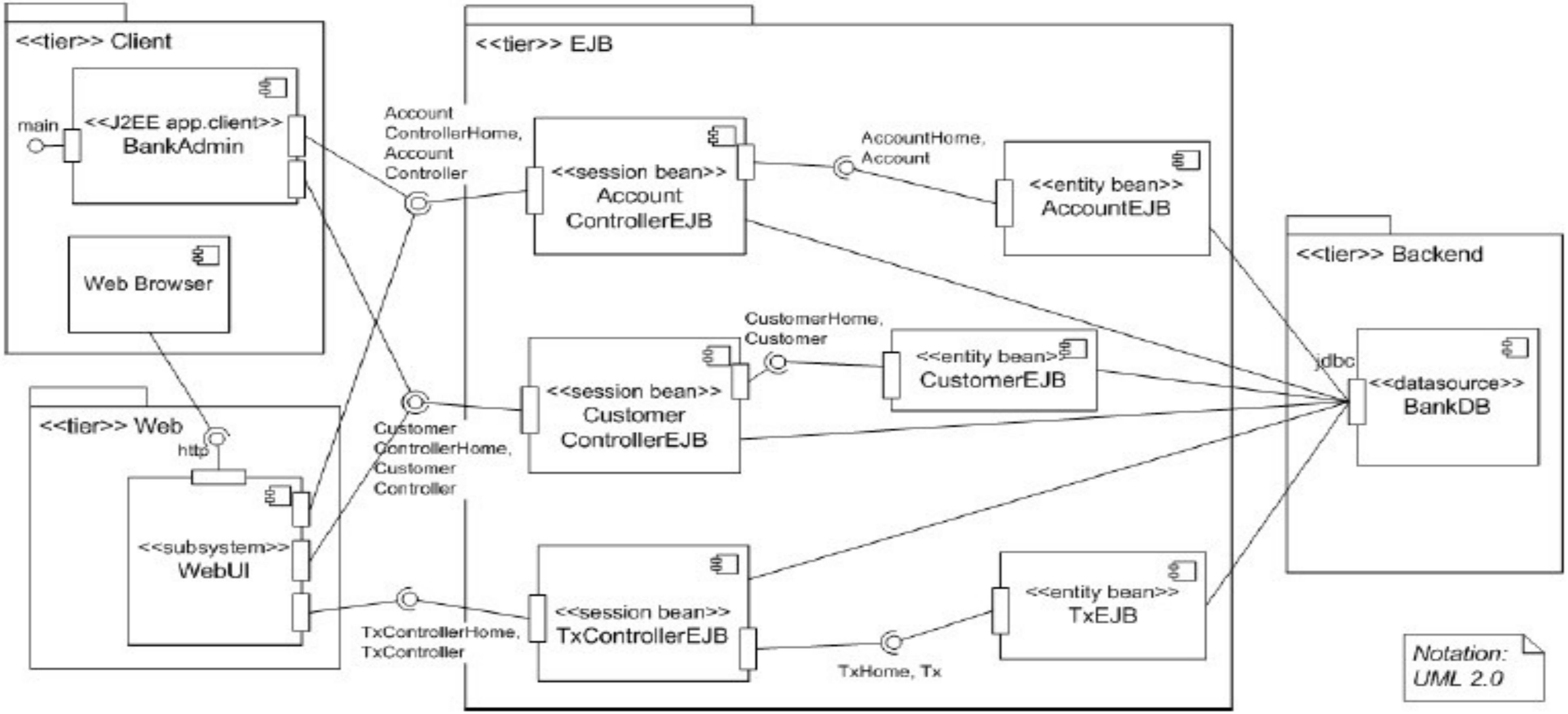
Component and Connector Views

- Describe the system in terms of runtime software elements (typically threads and processes, but also objects and data stores)
  - Component have interfaces called ports that define their interaction with their environment
  - Ports of the same type can be replicated to offer different input or output channels at runtime

- Elements:
  - Components: Runtime elements as Processes and / or Threads.
  - Connectors: Are defined as the forms of interaction between components (synchronous, asynchronous, complex transactions)
- Relations:
  - The relations are pathways for interaction and communication.
  - Component ports are associated with connector roles to yield a graph of components and connectors.

- Usage
  - Show how the system works.
  - Guide development by specifying structure and behavior of runtime elements.
  - Help reason about runtime system quality attributes, such as performance and availability.

- Runtime View



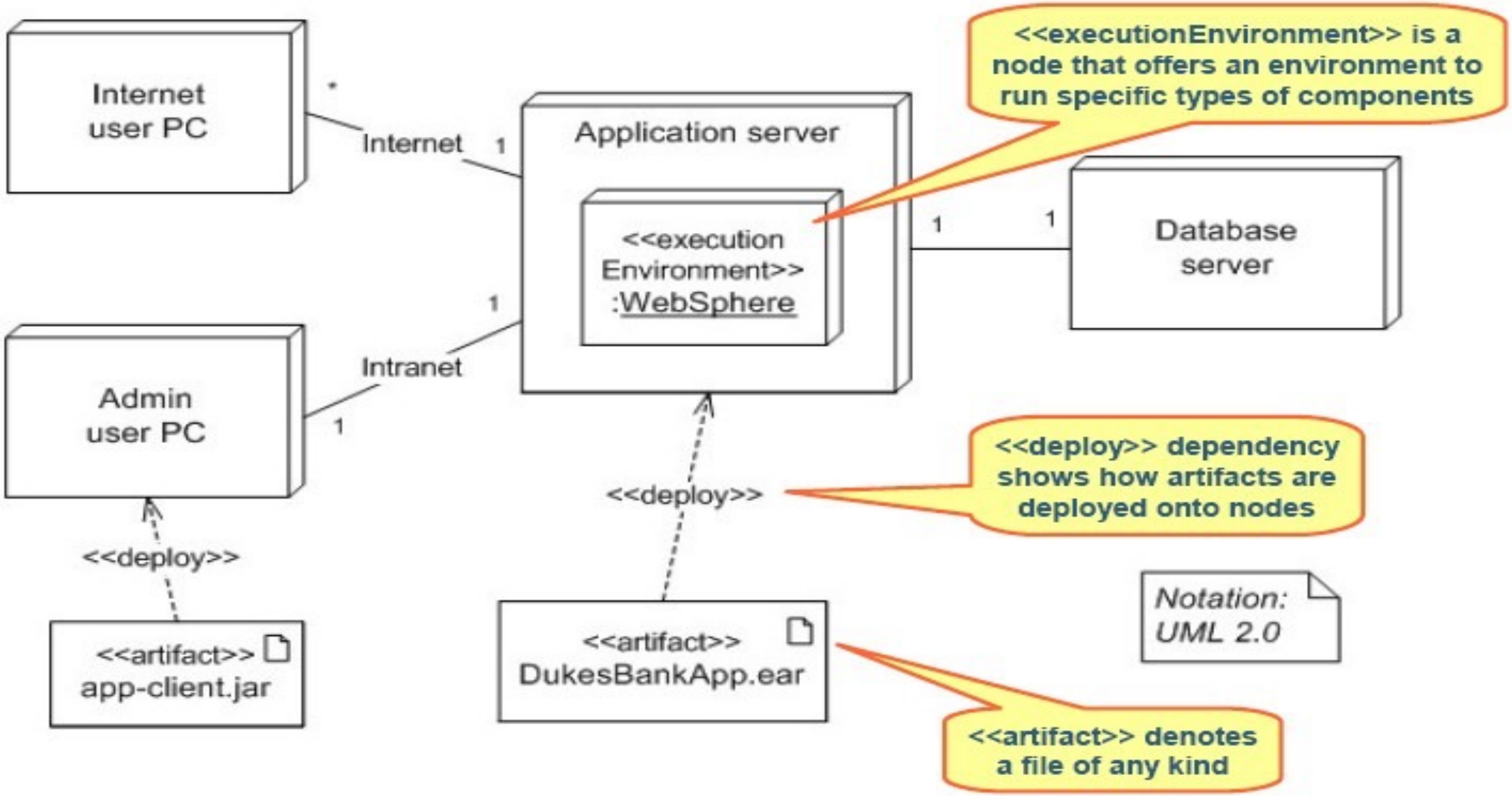
Notation: UML 2.0

Allocation Views

- Describe the mapping of software elements into non-software (typically hardware) elements
  - The most typical allocation view is the deployment view
- Elements:
  - Software Elements: elements from the C&C View
  - Environmental Elements: Hardware of the computing platforms
- Relationships:
  - Allocated-to: Describes the physical units where the software elements will be executed
  - Migrates-to, Copy-migrates-to or Execution-migrates-to to express migration tactics in case of failure / need of backup

- Usage
  - Reasoning about performance, availability, security, and safety.
  - Reasoning about distributed development and allocation of work to teams.
  - Reasoning about the form and mechanisms of system installation.

Deployment View



<<executionEnvironment>> is a node that offers an environment to run specific types of components

<<deploy>> dependency shows how artifacts are deployed onto nodes

<<artifact>> denotes a file of any kind

Choosing the views

- At a minimum, expect to have at least one module view, at least one C&C view, and for larger systems, at least one allocation view in your architecture document.
- Different views support different goals and uses.
  - The views you should document depend on the uses you expect to make of the documentation.
- Each view has a cost and a benefit
  - We should ensure that the benefits of maintaining a view outweigh its costs.

Which are the relevant views?

- What are the most relevant views?
- How many views are “good enough” to have?

- This totally depends:
  - On the nature of System
  - The goal you pursue when documenting the architecture
- Different views expose different quality attributes to different extent
  - How many? Occam´s razor – only the necessary ones
  - Less and updated is better than many obsolete and that nobody cares to read

Architectural Structures & Quality Concerns

Architectural Structures & Quality Concerns

Module structures

Software Structure	Element Types	Relations	Useful for	Quality Concerns Affected
Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; encapsulation	Modifiability
Uses	Module	Uses (i.e., requires the correct presence of)	Designing subsets and extensions	extensibility
Layers	Layer	Allowed to use the services of; provides abstraction to	Incremental development; implementing systems on top of “virtual machines”	Portability, modifiability
Class	Class, object	Is an instance of; is a generalization of	In object-oriented systems, factoring out commonality; planning extensions of functionality	Modifiability, extensibility
Data model	Data entity	{one, many}-to-{one, many}; generalizes; specializes	Engineering global data structures for consistency and performance	Modifiability, performance

Architectural Structures & Quality Concerns

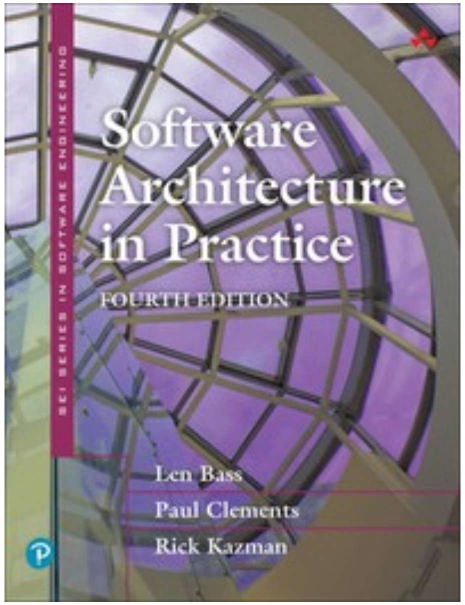
C & C structures				
Software Structure	**Element Types**	**Relations**	**Useful for**	**Quality Concerns Affected**
Service	Service, service registry	Attachment (via message-passing)	Scheduling analysis; performance analysis; robustness analysis	Interoperability, availability, modifiability
Concurrency	Processes, threads	Attachment (via communication and synchronization mechanisms)	Identifying locations where resource contention exists, opportunities for parallelism	Performance

Architectual Structures & Quality Concerns

Allocation structures				
Software Structure	**Element Types**	**Relations**	**Useful for**	**Quality Concerns Affected**
Deployment	Components, hardware elements	Allocated to; migrates to	Mapping software elements to system elements	Performance, security, energy, availability, deployability
Implementation	Modules, file structure	Stored in	Configuration control, integration, test activities	Development efficiency
Work assignment	Modules, organizational units	Assigned to	Project management, best use of expertise and available resources, management of commonality	Development efficiency

Reading (Must)

- Chapter 1 of T1: Software Architecture in Practice, 4th Edition, 2021.


SEI SERIES IN SOFTWARE ENGINEERING
Software Architecture in Practice
FOURTH EDITION
Len Bass
Paul Clements
Rick Kazman

https://learning.oreilly.com/library/view/software-architecture-in/9780136885979/

Acknowledgment

- Material (diagrams, text etc.) in this lecture is borrowed from the following sources:
  - Bass, Len, Paul Clements, and Rick Kazman. Software architecture in practice. Addison-Wesley Professional, 2021.

- Learning material and course ideation is provided by Dr. Javier Gonzalez Huerta, Dr. Muhammad Usman & Mr. Ehsan Zabardast

BLEKINGE TEKNISKA HÖGSKOLA • BTH •