



**BITS Pilani**

Pilani | Dubai | Goa | Hyderabad

# Architectural Structures and Views

Harvinder S Jabbal  
SSZG653 Software Architectures

February 19, 2022

SEZG651/SSZG653 Software  
Architectures



# SEZG651/ SSZG653 Software Architectures Module 4-CS 05/1

February 19, 2022

SEZG651/SSZG653 Software  
Architectures



## Structure & View

# Structure & View



- A **view** is a representation of a coherent set of architectural elements, as written by and read by system stakeholders.
  - It consists of a representation of a set of elements and the relations among them.
- A **structure** is the set of elements itself, as they exist in software or hardware.

# example



- A **module structure** is the set of the system's modules and their organization.
- A **module view** is the representation of that structure, as documented by and used by some system stakeholders.
- These terms are often used interchangeably, but we will adhere to these definitions.



# Architectural structures

# Module structures.

- Here the elements are modules, which are units of implementation.
- Modules represent a code-based way of considering the system.
- They are assigned areas of functional responsibility.
- There is less emphasis on how the resulting software manifests itself at runtime.
- Module structures allow us to answer questions such as
  - What is the primary functional responsibility assigned to each module?
  - What other software elements is a module allowed to use?
  - What other software does it actually use?
  - What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?

# Component-and-connector structures.



- Here the elements are
  - runtime components (which are the principal units of computation) and
  - connectors (which are the communication vehicles among components).
- Component-and-connector structures help answer questions such as
  - What are the major executing components and how do they interact?
  - What are the major shared data stores?
  - Which parts of the system are replicated?
  - How does data progress through the system?
  - What parts of the system can run in parallel?
  - How can the system's structure change as it executes?



# Allocation structures.



- Allocation structures show the relationship between
  - the software elements and
  - the elements in one or more external environments in which the software is created and executed.
- They answer questions such as
  - What processor does each software element execute on?
  - In what files is each element stored during development, testing, and system building?
  - What is the assignment of software elements to development teams?



# architectural design

# 3 structures

## 3 broad decision types

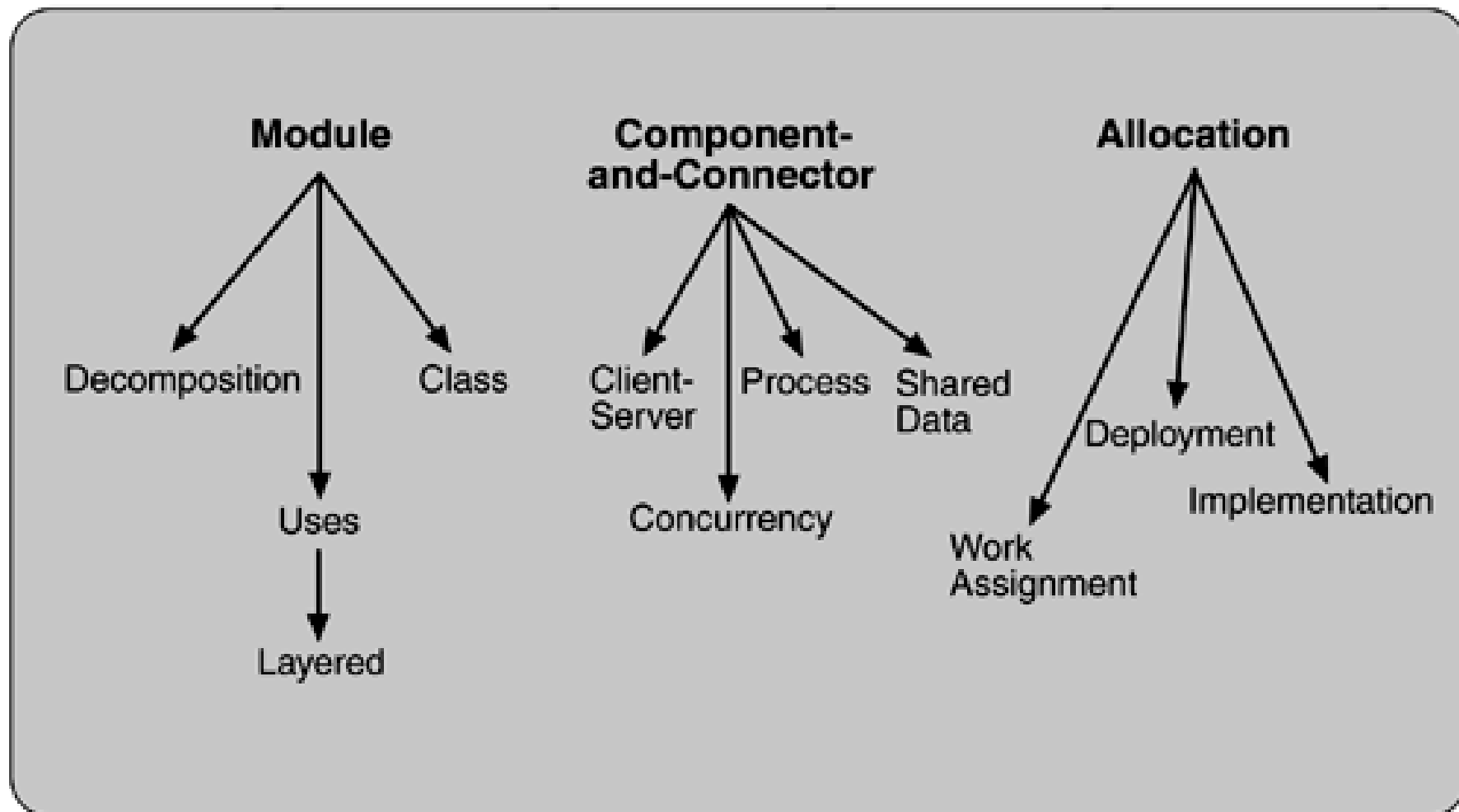


- How is the system to be structured as a set of code units (modules)?
- How is the system to be structured as a set of elements that have
  - runtime behavior (components) and
  - interactions (connectors)?
- How is the system to relate to nonsoftware structures in its environment
  - (i.e., CPUs, file systems, networks, development teams, etc.)?



# SOFTWARE STRUCTURES

# SOFTWARE STRUCTURES





# Module

# Decomposition.



- The units are modules related to each other by the "is a submodule of " relation, showing how larger modules are decomposed into smaller ones recursively until they are small enough to be easily understood.
- Modules in this structure represent a common starting point for design, as the architect enumerates what the units of software will have to do and assigns each item to a module for subsequent (more detailed) design and eventual implementation.
- Modules often have associated products (i.e., interface specifications, code, test plans, etc.).
- The decomposition structure provides a large part of the system's modifiability, by ensuring that likely changes fall within the purview of at most a few small modules.
- It is often used as the basis for the development project's organization, including the structure of the documentation, and its integration and test plans. The units in this structure often have organization-specific names.
- Certain U.S. Department of Defense standards, for instance, define Computer Software Configuration Items (CSCIs) and Computer Software Components (CSCs), which are units of modular decomposition.

# Uses.



- The units of this important but overlooked structure are also modules, or (in circumstances where a finer grain is warranted) procedures or resources on the interfaces of modules.
- The units are related by the uses relation.
- One unit uses another if the correctness of the first requires the presence of a correct version (as opposed to a stub) of the second.
- The uses structure is used to engineer systems that can be easily extended to add functionality or from which useful functional subsets can be easily extracted.
- The ability to easily subset a working system allows for incremental development.



# Layered.



- When the uses relations in this structure are carefully controlled in a particular way, a system of layers emerges, in which a layer is a coherent set of related functionality.
- In a strictly layered structure, layer  $n$  may only use the services of layer  $n - 1$ .
- Many variations of this (and a lessening of this structural restriction) occur in practice, however.
- Layers are often designed as abstractions (virtual machines) that hide implementation specifics below from the layers above, engendering portability.

# Class, or generalization.



- The module units in this structure are called classes.
- The relation is "inherits-from" or "is-an-instance-of."
- This view supports reasoning about collections of similar behavior or capability (i.e., the classes that other classes inherit from) and parameterized differences which are captured by subclassing.
- The class structure allows us to reason about re-use and the incremental addition of functionality.



# Component and Connector

# Process, or communicating processes.



- Like all component-and-connector structures, this one is orthogonal to the module-based structures and deals with the dynamic aspects of a running system.
- The units here are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations.
- The relation in this (and in all component-and-connector structures) is attachment, showing how the components and connectors are hooked together.
- The process structure is important in helping to engineer a system's execution performance and availability.

# Concurrency.



- This component-and-connector structure allows the architect to determine opportunities for parallelism and the locations where resource contention may occur.
- The units are components and the connectors are "logical threads."
- A logical thread is a sequence of computation that can be allocated to a separate physical thread later in the design process.
- The concurrency structure is used early in design to identify the requirements for managing the issues associated with concurrent execution.

# Shared data, or repository.



- This structure comprises components and connectors that create, store, and access persistent data.
- If the system is in fact structured around one or more shared data repositories, this structure is a good one to illuminate.
- It shows how data is produced and consumed by runtime software elements, and it can be used to ensure good performance and data integrity.

# Client-server.



- If the system is built as a group of cooperating clients and servers, this is a good component-and-connector structure to illuminate.
- The components are the clients and servers, and the connectors are protocols and messages they share to carry out the system's work.
- This is useful for separation of concerns (supporting modifiability), for physical distribution, and for load balancing (supporting runtime performance).



# Allocation



# Deployment.



- The deployment structure shows how software is assigned to hardware-processing and communication elements.
- The elements are software (usually a process from a component-and-connector view), hardware entities (processors), and communication pathways.
- Relations are "allocated-to," showing on which physical units the software elements reside, and "migrates-to," if the allocation is dynamic.
- This view allows an engineer to reason about performance, data integrity, availability, and security.
- It is of particular interest in distributed or parallel systems.

# Implementation.



- This structure shows how software elements (usually modules) are mapped to the file structure(s) in the system's development, integration, or configuration control environments.
- This is critical for the management of development activities and build processes.

# Work assignment.



- This structure assigns responsibility for implementing and integrating the modules to the appropriate development teams.
- Having a work assignment structure as part of the architecture makes it clear that the decision about who does the work has architectural as well as management implications.
- The architect will know the expertise required on each team.
- Also, on large multi-sourced distributed development projects, the work assignment structure is the means for calling out units of functional commonality and assigning them to a single team, rather than having them implemented by everyone who needs them.

# elements and relations in each structure



Software Structure	Relations	Useful for
Decomposition	Is a submodule of; shares secret with	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control
Uses	Requires the correct presence of	Engineering subsets; engineering extensions
Layered	Requires the correct presence of; uses the services of; provides abstraction to	Incremental development; implementing systems on top of "virtual machines" portability
Class	Is an instance of; shares access methods of	In object-oriented design systems, producing rapid almost-alike implementations from a common template

# elements and relations in each structure



Software Structure	Relations	Useful for
Client-Server	Communicates with; depends on	Distributed operation; separation of concerns; performance analysis; load balancing
Process	Runs concurrently with; may run concurrently with; excludes; precedes; etc.	Scheduling analysis; performance analysis
Concurrency	Runs on the same logical thread	Identifying locations where resource contention exists, where threads may fork, join, be created or be killed
Shared Data	Produces data; consumes data	Performance; data integrity; modifiability

# elements and relations in each structure



Software Structure	Relations	Useful for
Deployment	Allocated to; migrates to	Performance, availability, security analysis
Implementation	Stored in	Configuration control, integration, test activities
Work Assignment	Assigned to	Project management, best use of expertise, management of commonality



## Kruchten's four views

# WHICH STRUCTURES TO CHOOSE?



## Logical.

The elements are "key abstractions," which are manifested in the object-oriented world as objects or object classes. This is a module view.

## Process.

This view addresses concurrency and distribution of functionality. It is a component-and-connector view.

## Development.

This view shows the organization of software modules, libraries, subsystems, and units of development. It is an allocation view, mapping software to the development environment.

## Physical.

This view maps other elements onto processing and communication nodes and is also an allocation view (which others call the deployment view).



# Thank you



Ref. Text Book