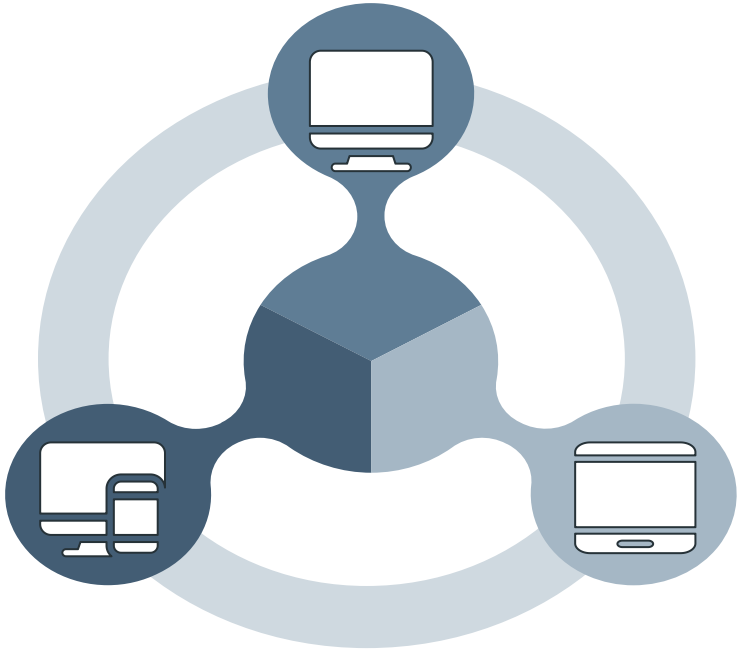
System Architecture

Summary

A **system's architecture** is a *representation of a system* in which there is a *definition of functionality* onto hardware and software components, and *a concern for the human interaction* with these components.

Introduction

- Modern systems are frequently too complex to grasp all at once,
- To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture.



Architectural Structures and Views

Structures

A structure are elements in the software or hardware.

View

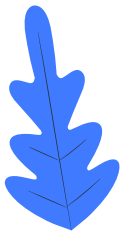
A view is a representation of a coherent set of architectural elements, as written by and read by system stakeholders. It consists of a representation of a set of elements and the relations among them.

Three Kind of Structure

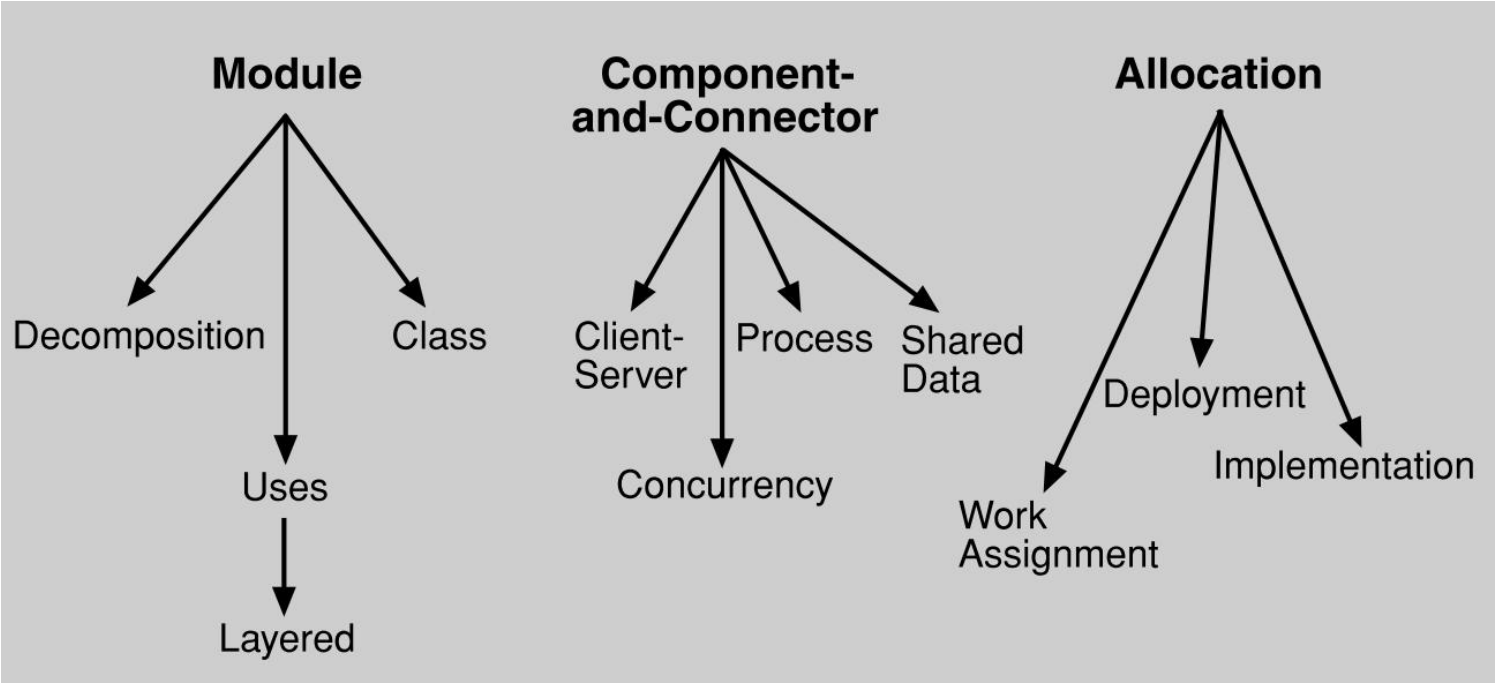
MODULE

COMPONENT AND CONNECTOR

ALLOCATION

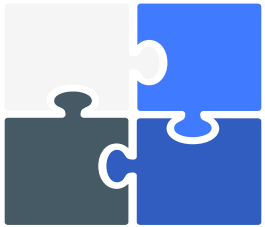


Three Kind of Structure



Module Structure

- The module structure defines how the system will be organized into various code or data units that need to be built or obtained.
- Each module is given a specific function or responsibility, with less focus on how the software will actually run when it's being used.



Module Structure

Module structures allow us to answer questions such as these:

- 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 and depend on?
- What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?

🔗 Component and Connector Structure

- The component and connector structure defines how the system is organized into elements that perform actions during runtime (components) and how these elements interact with each other (connectors).
- Runtime components are the main units that perform computations, and they can be services, peers, clients, servers, filters, or other elements that work during runtime.
- Connectors are the tools that allow components to communicate with each other, such as methods like call-return, process synchronization, or pipes.

🔗 Component and Connector Structure

Component and Connector structures allow us to answer questions such as these:

- What are the major executing components and how do they interact at runtime?
- 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?
- Can the system's structure change as it executes and, if so, how?

Allocation Structure

- The allocation structure defines how the system will interact with non-software elements in its environment, like CPUs, file systems, networks, and development teams.

- These structures show how the software elements connect with parts of one or more external environments where the software is developed and run.

Allocation Structure

Allocation structures allow us to answer questions such as these:

- How are software components mapped to physical hardware?
- In what directories or files is each element stored during development, testing, and system building?
- Which teams are responsible for developing or maintaining different components?

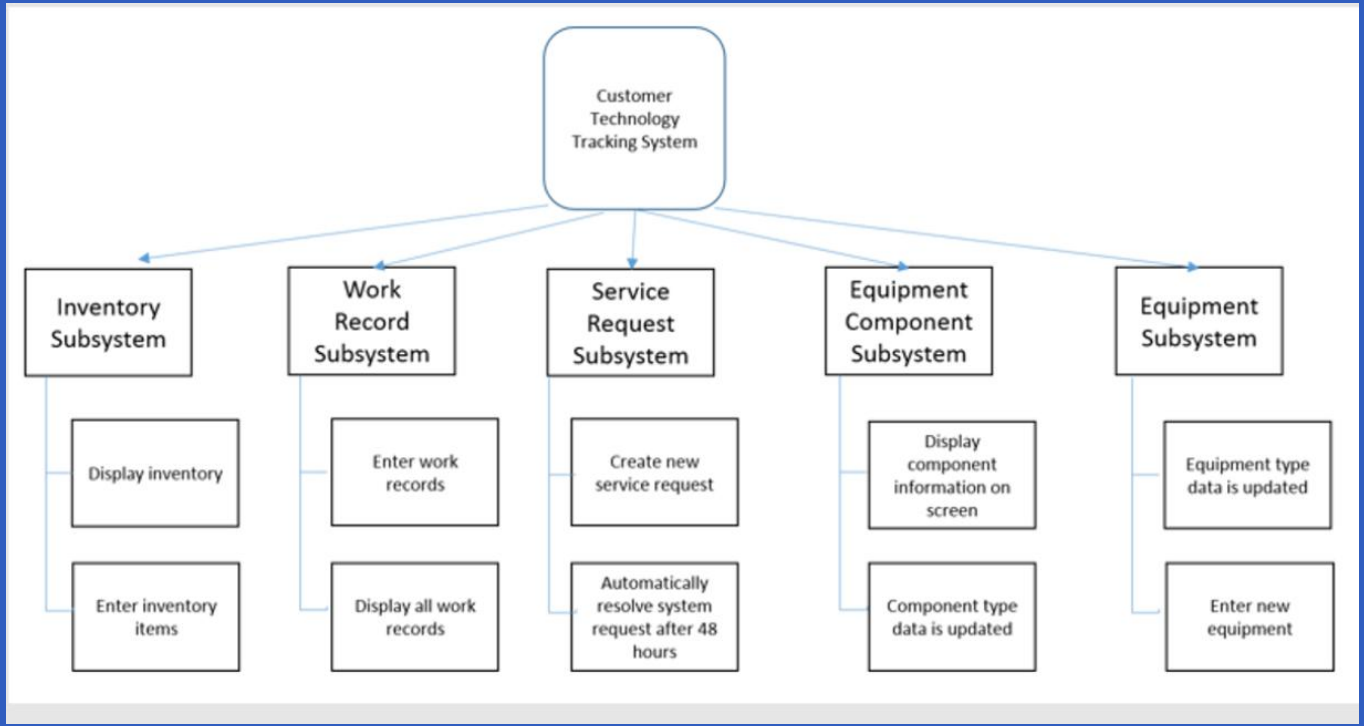
Structures under Module

- DECOMPOSITION
- USES
- LAYER
- CLASS
- DATA MODEL

DECOMPOSITION

- Modules are decomposed into smaller modules. Making it easier to understand.
- This structure is often used as the basis for the development project's organization, including the structure of the documentation, and the project's integration and test plans.

DECOMPOSITION



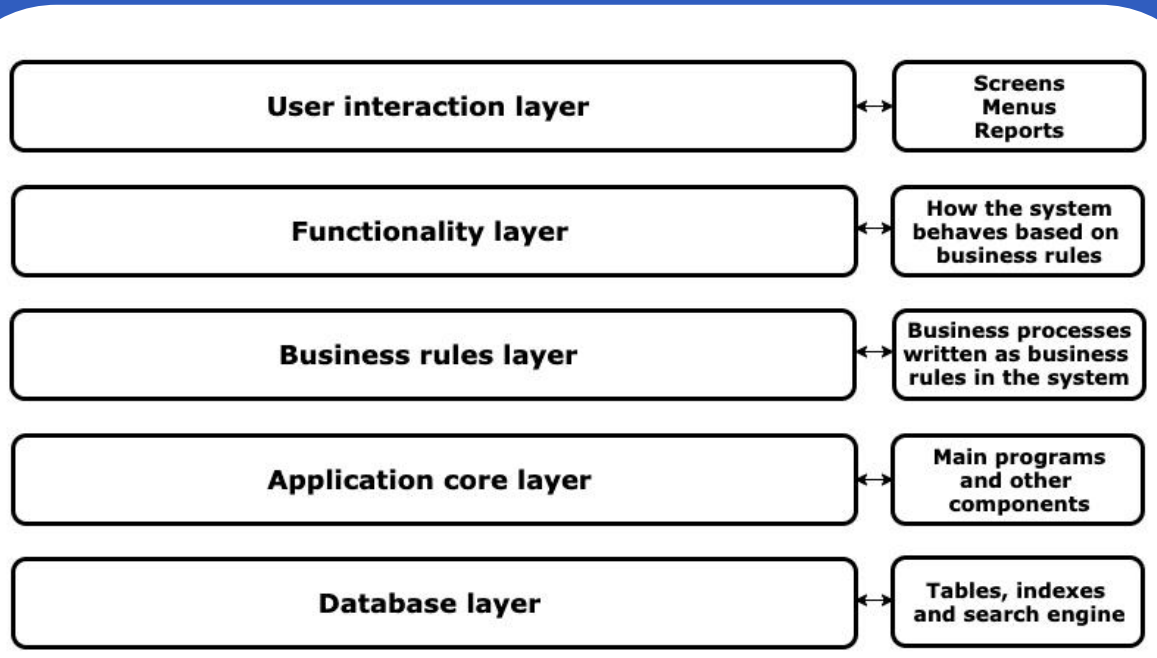
USES

- Illustrates the dependencies or "uses" relationships between modules. It shows which modules rely on the services or functionalities of other modules.
- Shows the "uses" relationships between modules, indicating which modules require the services or functionalities provided by other modules.

LAYER

- Organizes the system into layers, where each layer depends on the one below it and provides services to the one above it.
- This structure is useful for understanding the separation of concerns and abstraction levels..

LAYER


User interaction layer
Screens
Menus
Reports
Functionality layer
How the system behaves based on business rules
Business rules layer
Business processes written as business rules in the system
Application core layer
Main programs and other components
Database layer
Tables, indexes and search engine

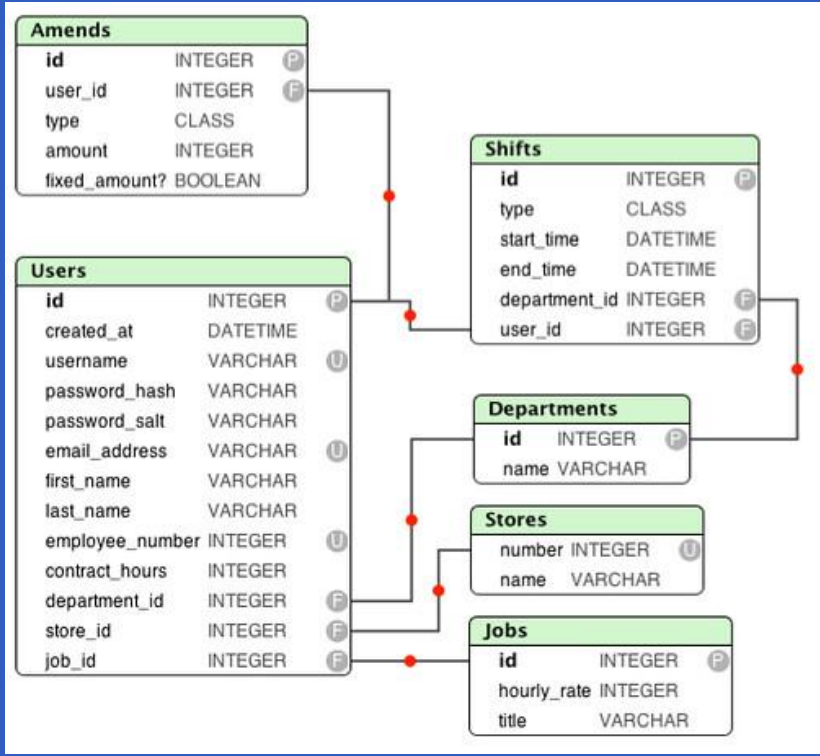
CLASS

- Often used in object-oriented systems, this structure defines the relationships between classes or objects, such as inheritance, associations, and aggregations.
- This view supports reasoning about collections of similar behavior or capability (e.g., the classes that other classes inherit from) and parameterized differences.
- If any documentation exists for a project that has followed an object-oriented analysis and design process, it is typically this structure.

DATA MODEL

- The data model describes the static information structure in terms of data entities and their relationships.
- For example, in a banking system, entities will typically include Account, Customer, and Loan. Account has several attributes, such as account number, type (savings or checking), status, and current balance..

DATA MODEL


Amends
id INTEGER P
user_id INTEGER F
type CLASS
amount INTEGER
fixed_amount? BOOLEAN
Users
id INTEGER P
created_at DATETIME
username VARCHAR U
password_hash VARCHAR
password_salt VARCHAR
email_address VARCHAR U
first_name VARCHAR
last_name VARCHAR
employee_number INTEGER U
contract_hours INTEGER
department_id INTEGER F
store_id INTEGER F
job_id INTEGER F
Shifts
id INTEGER P
type CLASS
start_time DATETIME
end_time DATETIME
department_id INTEGER F
user_id INTEGER F
Departments
id INTEGER P
name VARCHAR
Stores
number INTEGER U
name VARCHAR
Jobs
id INTEGER P
hourly_rate INTEGER
title VARCHAR

Structures under Components & Connector

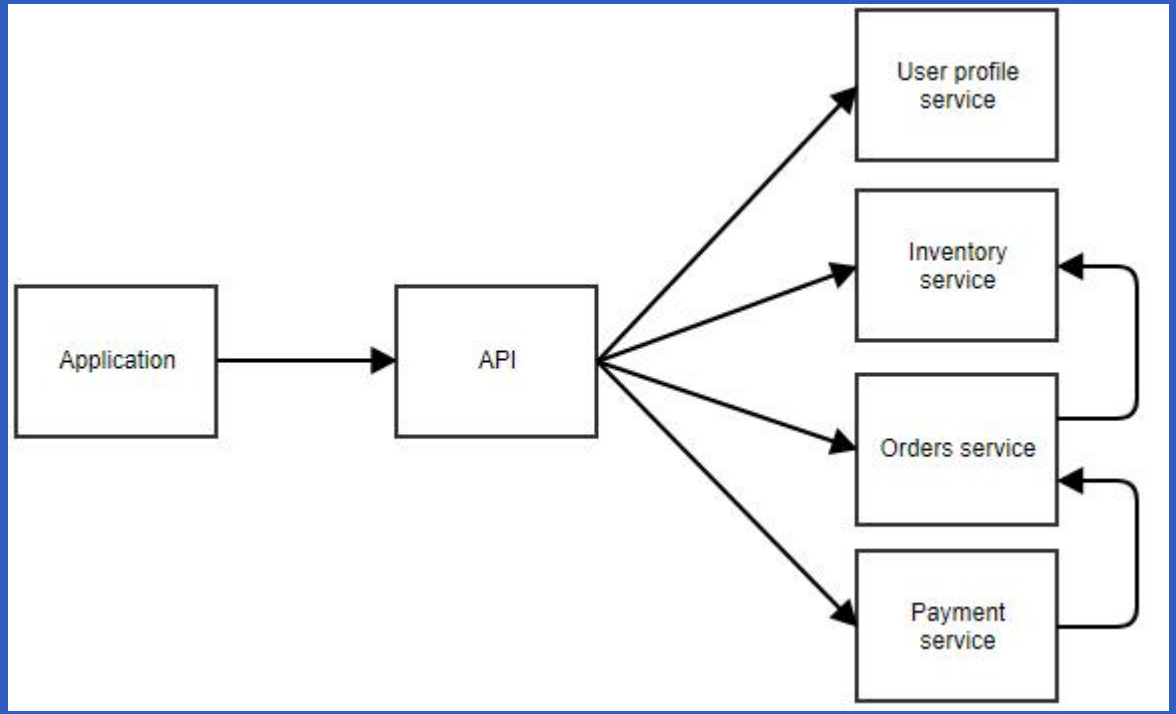
- SERVICE
- CONCURRENCY

Component-and-connector structures show a runtime view of the system.

SERVICE

- The units here are services that interoperate with each other by service coordination mechanisms such as SOAP (Simple Object Access Protocol).
- The service structure is an important structure to help engineer a system composed of components that may have been developed anonymously and independently of each other.

SERVICE


Application
API
User profile service
Inventory service
Orders service
Payment service

CONCURRENCY STRUCTURE

- Illustrates how components are executed concurrently, including threads, processes, and synchronization mechanisms.
- This structure is key for understanding how the system handles parallelism and concurrency.
- The units are components and the connectors are their communication mechanisms.

Structures under Allocation

- DEPLOYMENT
- IMPLEMENTATION
- Work Assignment

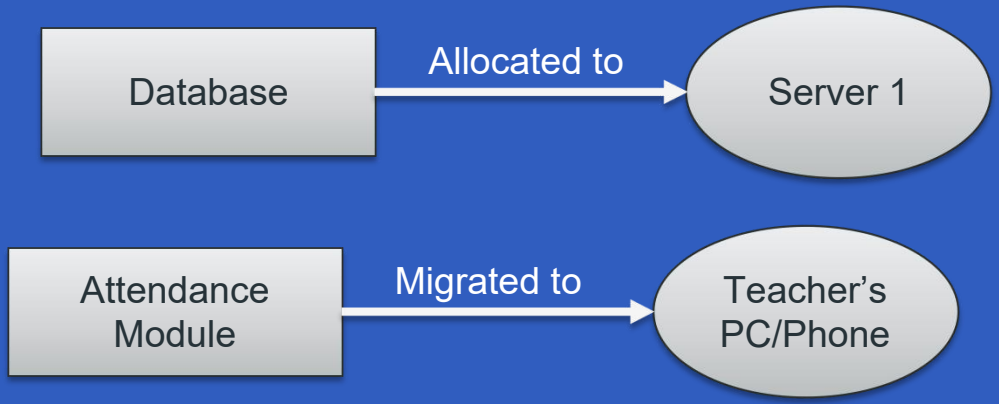
Allocation structures define how the elements from other structures map onto things that are not software: typically hardware, teams, and file systems.

DEPLOYMENT

- The deployment structure shows how software is assigned to hardware processing and communication elements.
- Relations are *allocated-to*, showing on which physical units the software elements reside, and *migrates-to* if the allocation is dynamic.
- This structure can be used to reason about performance, data integrity, security, and availability. It is of particular interest in distributed and parallel systems.

DEPLOYMENT

EXAMPLE:



IMPLEMENTATION

- The Implementation Structure focuses on how the software's logical elements (like modules, components, or layers) are mapped to the physical artifacts of the software, such as source code files, directories, and other development assets.
- 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 teams who will carry it out.
- The architect will know the expertise required on each team
- This structure will also determine the major communication pathways among the teams: regular teleconferences, wikis, email lists, and so forth.

	Software Structure	Element Types	Relations	Useful For	Quality Attributes Affected
Module Structures	Decomposition	Module	Is a submodule of	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control	Modifiability
	Uses	Module	Uses (i.e., requires the correct presence of)	Engineering subsets, engineering extensions	"Subsetability," extensibility
	Layers	Layer	Requires the correct presence of, uses the services of, provides abstraction to	Incremental development, implementing systems on top of "virtual machines"	Portability
	Class	Class, object	Is an instance of, shares access methods of	In object-oriented design 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
C&C Structures	Service	Service, ESB, registry, others	Runs concurrently with, may run concurrently with, excludes, precedes, etc.	Scheduling analysis, performance analysis	Interoperability, modifiability
	Concurrency	Processes, threads	Can run in parallel	Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed	Performance, availability
Allocation Structures	Deployment	Components, hardware elements	Allocated to, migrates to	Performance, availability, security analysis	Performance, security, availability
	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

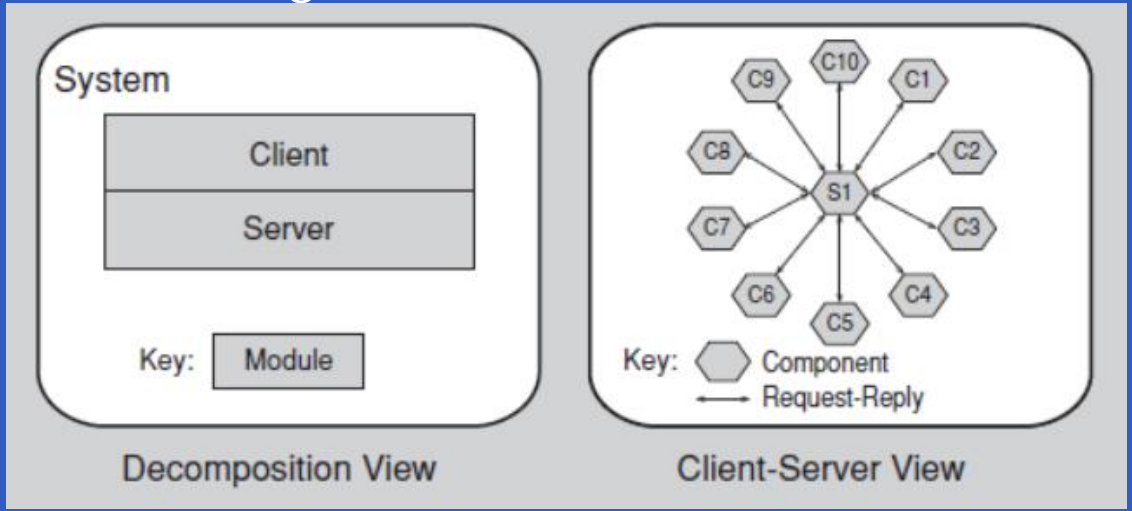
Relating Structures to each other

- Each of these structures provides a different perspective and design handle on a system, and each is valid and useful in its own right.

- Structures are not independent.
- For example, a module in a decomposition structure may be manifested as one, part of one, or several components in one of the component-and-connector structures, reflecting its runtime alter ego.

Relating Structures to each other

- The Figure below shows a very simple example of how two structures might relate to each other.



END of Presentation