Unidade Curricular

“Padrões e Desenho de Software”

#02 - Design

António José Ribeiro Neves

an@ua.pt

https://www.ua.pt/pt/uc/12275

Outline

QUIZZ #02

CAMBADA ROBOT'S SOFTWARE

SOFTWARE DESIGN

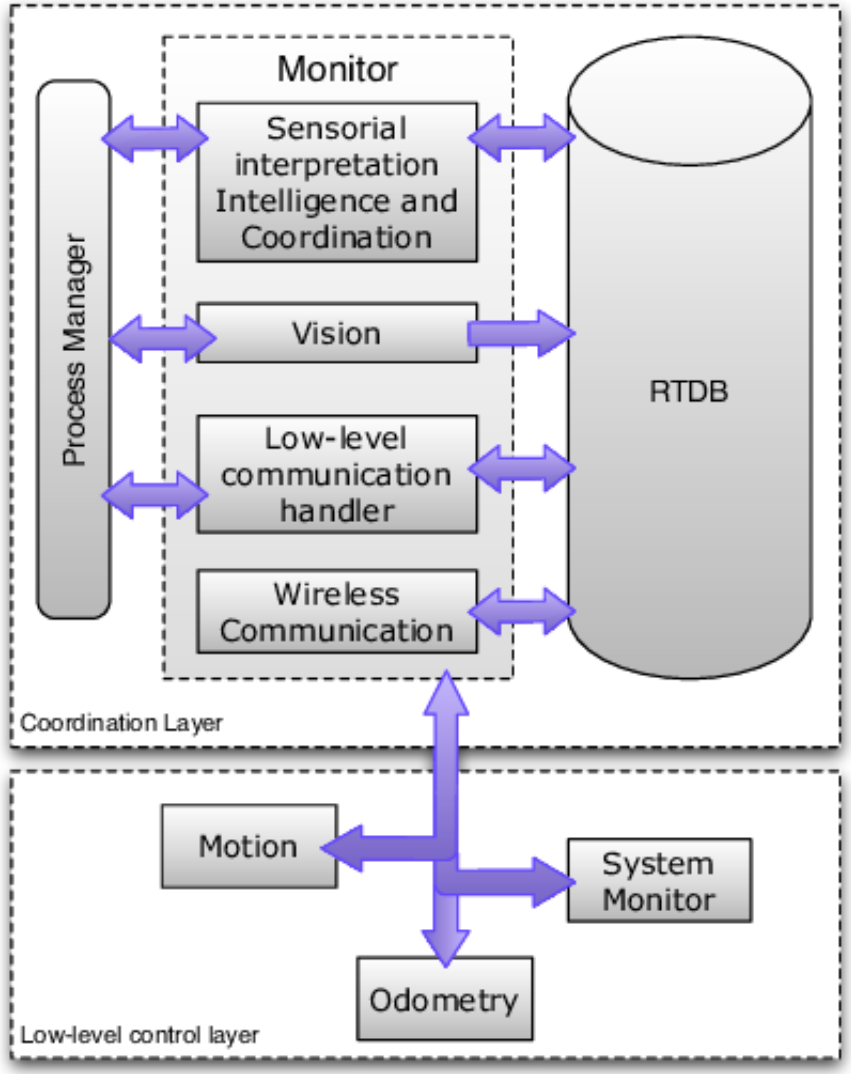
DESIGN PATTERNS

Propose the software for the CAMBADA robots

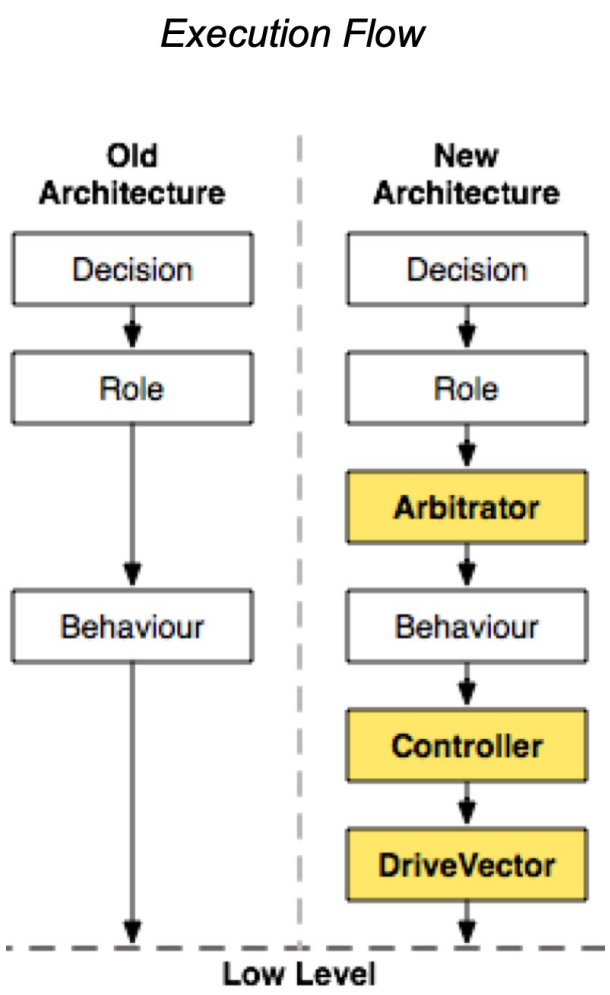
15 minutes

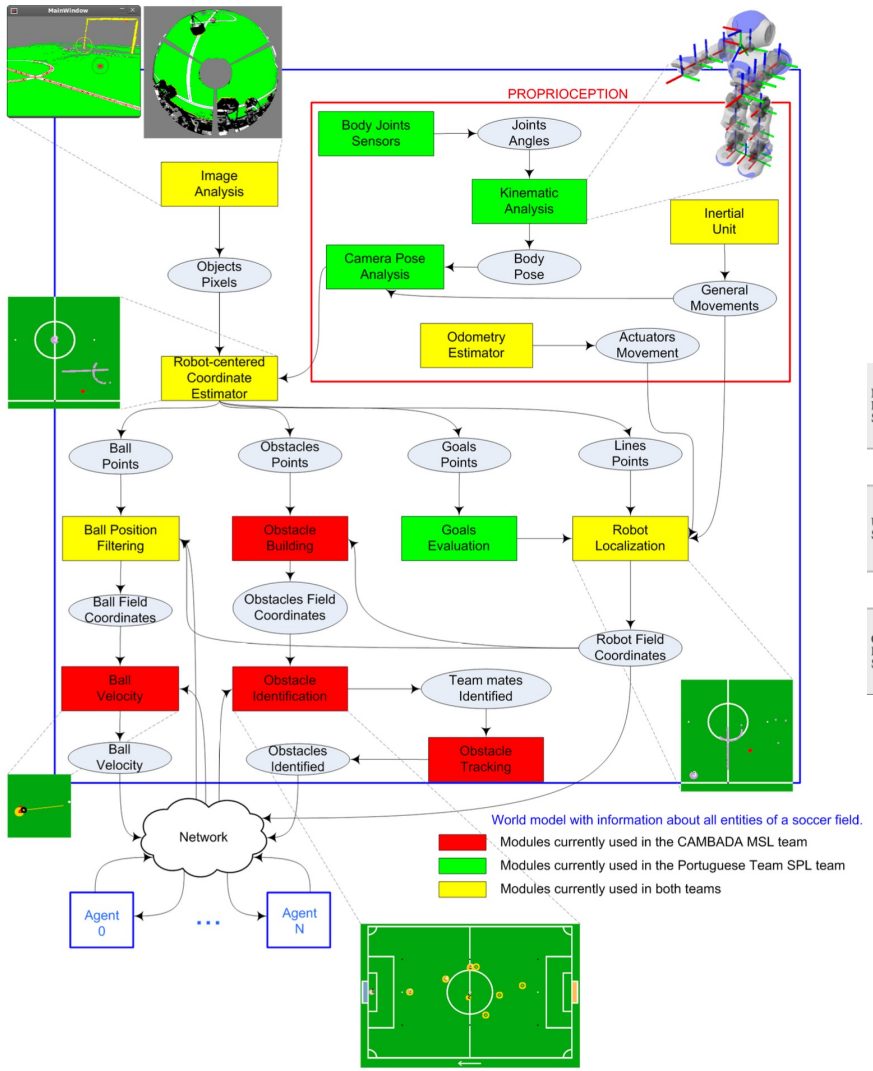
Propose the software for the CAMBADA robots

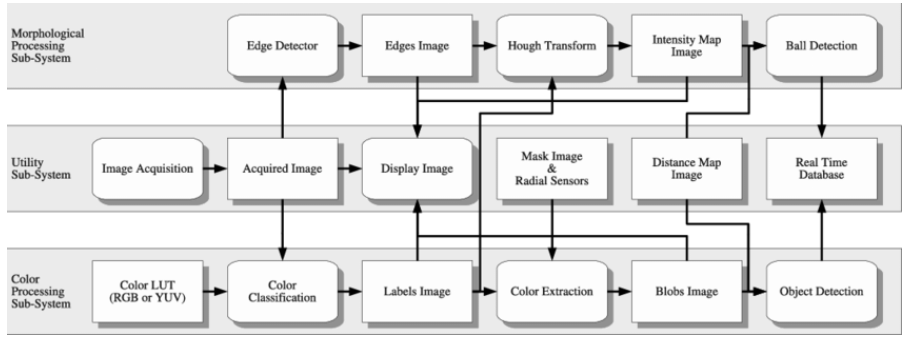
15 more minutes…

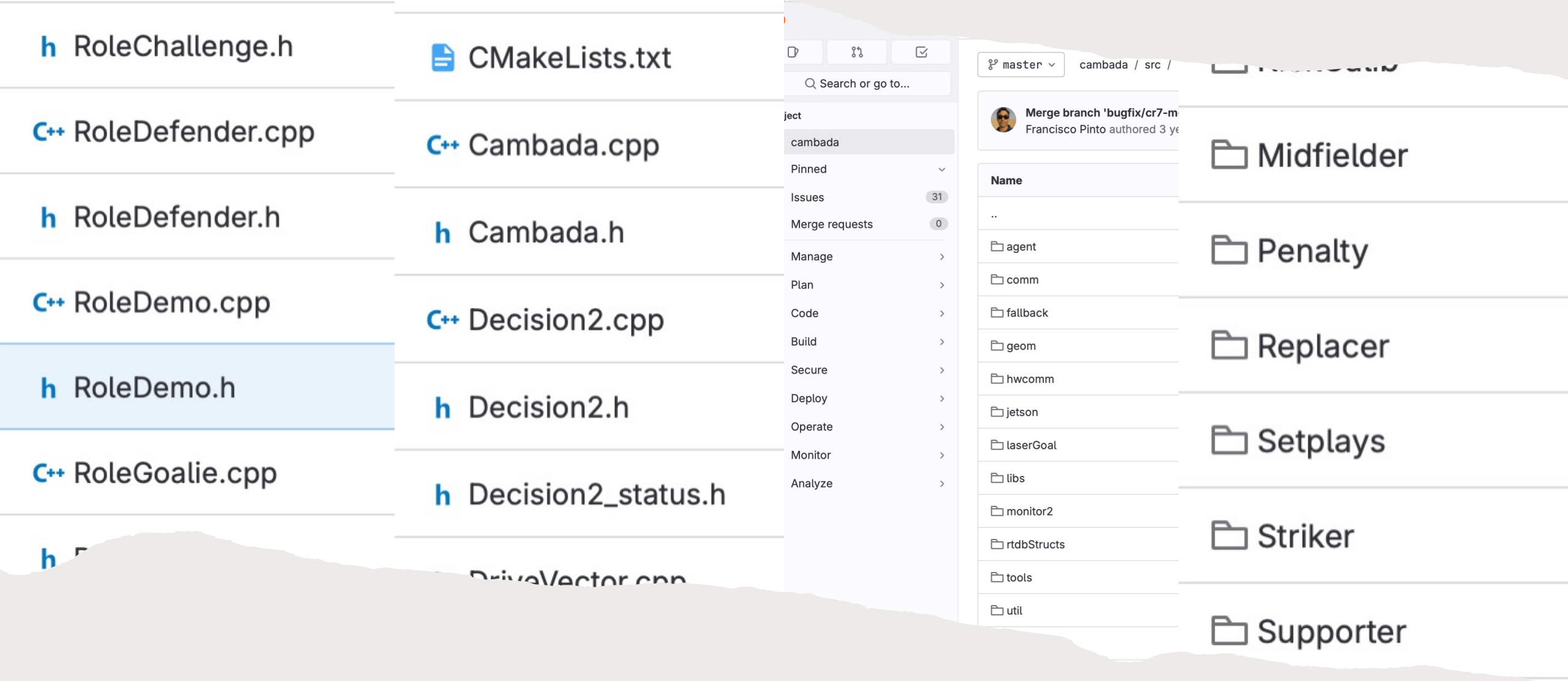


The real software design of CAMBADA









Software implementation

Let’s take a short break
10 Minutes

You are free to go grab
a coffee, water, etc.

But... 10 minutes **is 10 minutes** (600 seconds, **not 601 seconds!)**

10 minutes

Design is a Universal Activity

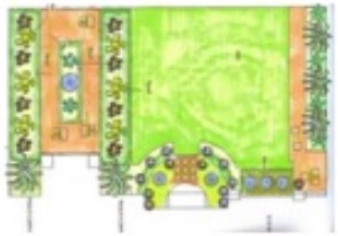
- Any product that is an aggregate of more primitive elements, can benefit from the activity of design

Building Design

Doors, windows, plumbing fixtures, …

Wood, steel, concrete, glass, …

Landscape Design



Trees, flowers, grass, rocks, mulch, …

User Interface Design

Tree view, table view, File chooser, …

Buttons, labels, text boxes, …

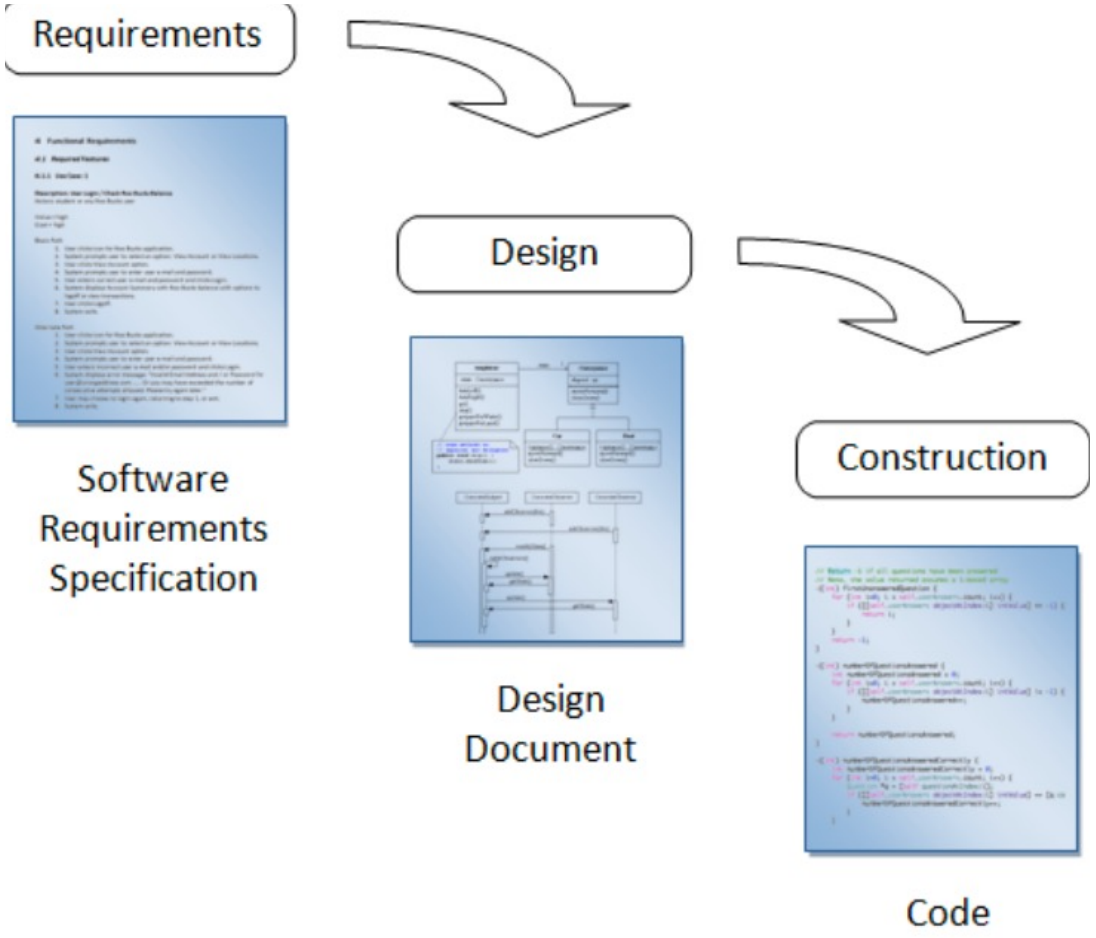
Software Design

Classes, procedures, functions, …

Data declaration, expressions, control flow statements, …

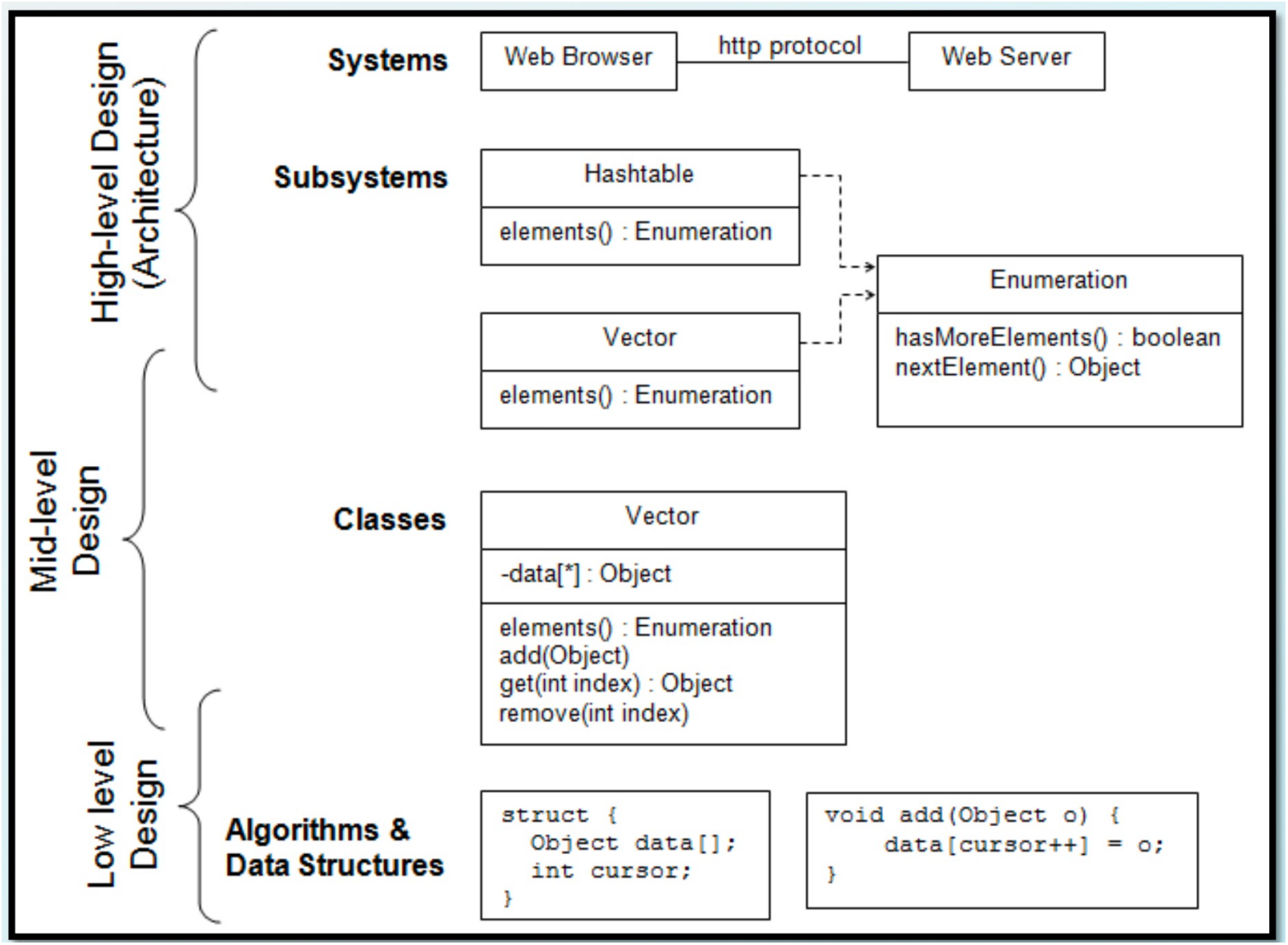
What is software design

- Design bridges that gap between knowing what is needed (software requirements specification) to enter the code that makes it work (the construction phase)
- During the design phase, software engineers apply their knowledge of the problem domain and implementation technologies in order to translate system specifications into plans for the technical implementation of the software
- The resulting design expresses the overall structure and organization of the planned implementation. It captures the essence of the solution independent of any implementation language



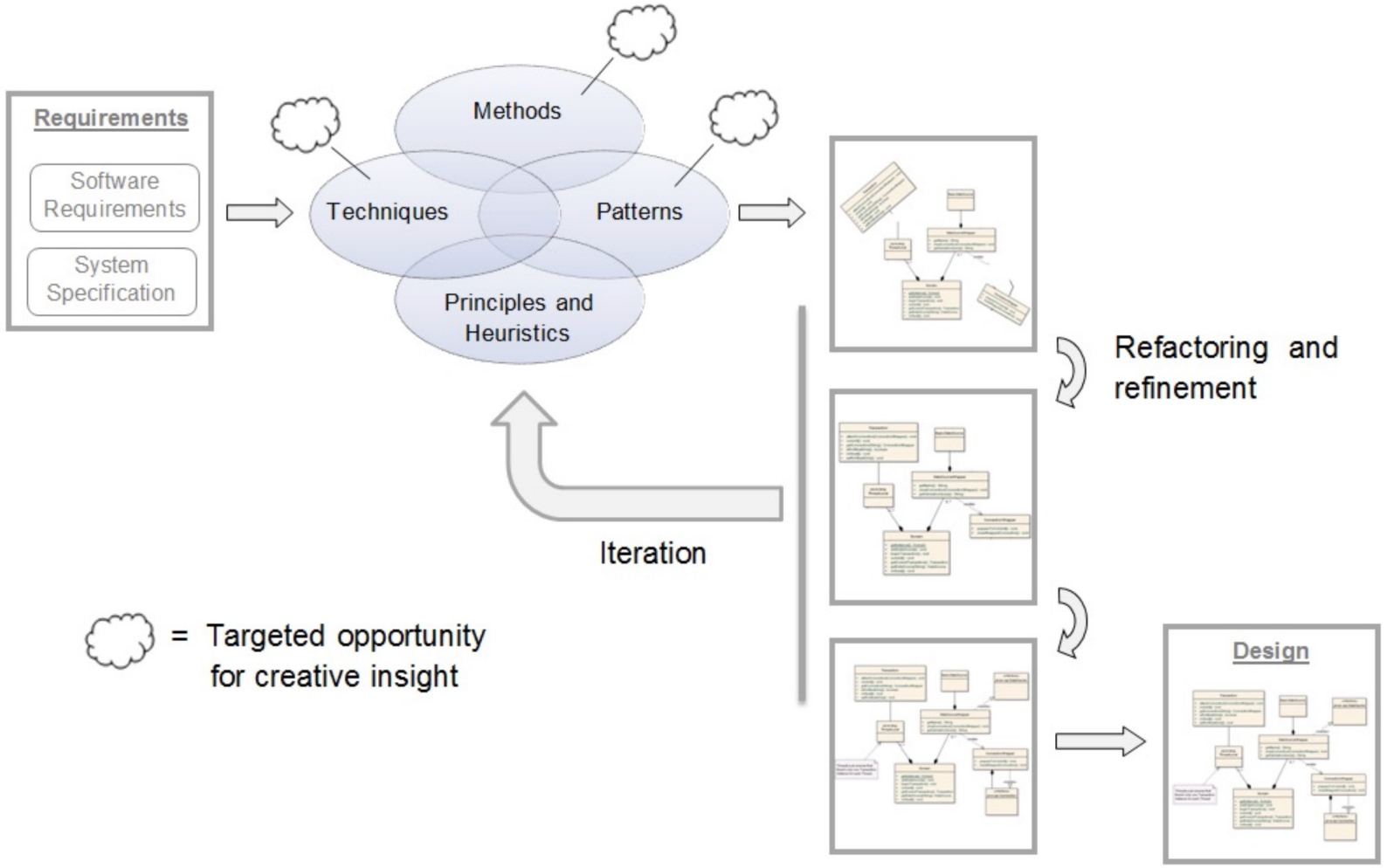
What is software design

- There is a process for design but not an algorithm
- Software design is a heuristic rather than a deterministic process
- Design is needed at several different levels of detail in a system



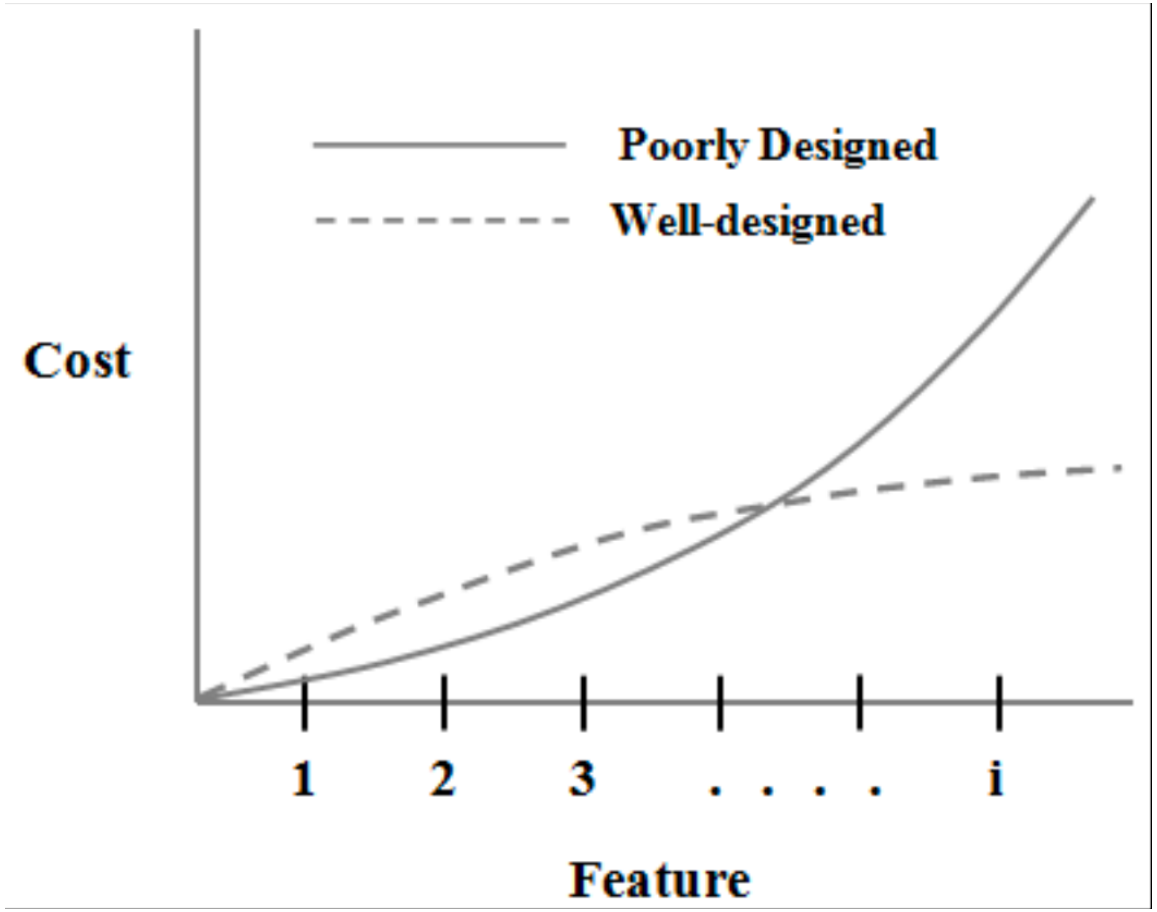
Importance of Software Design

- The design process can be made more systematic and predictable through the application of methods, techniques and patterns, all applied according to principles and heuristics



Importance of Managing Complexity

- Poorly designed programs are difficult to understand and modify
- The larger the program, the more pronounced are the consequences of poor design
- Two major types of complexity in software:
  - Essential complexities– complexities that are inherent in the problem.
  - Accidental/incidental complexities– complexities that are artifacts of the solution
- Design is the primary tool for managing essential and accidental complexities in software



Goal: manage essential complexity while avoiding the introduction of additional accidental complexities

Dealing with Software Complexity

Modularity – subdivide the solution into smaller easier to manage components. (divide and conquer)

Abstraction – use abstractions to suppress details in places where they are unnecessary.

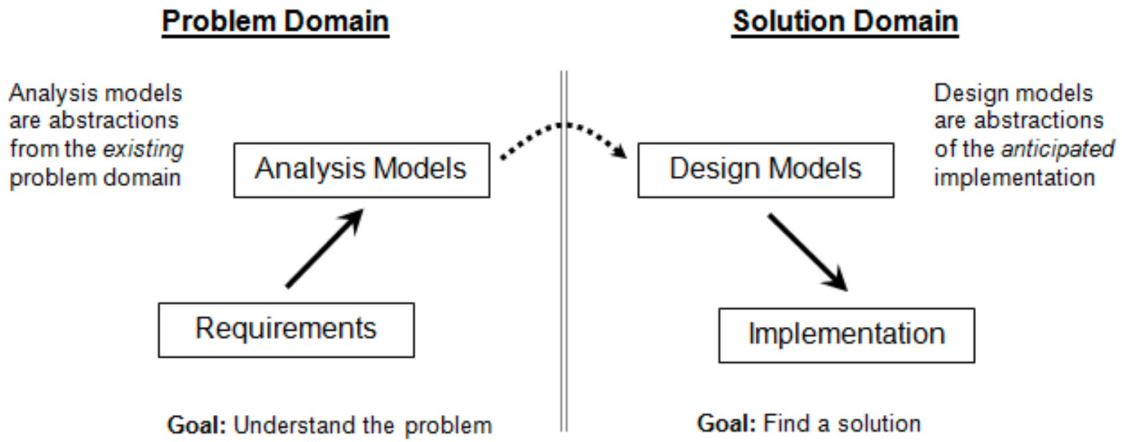
Information Hiding – hide details and complexity behind simple interfaces

Inheritance – general components may be reused to define more specific elements.

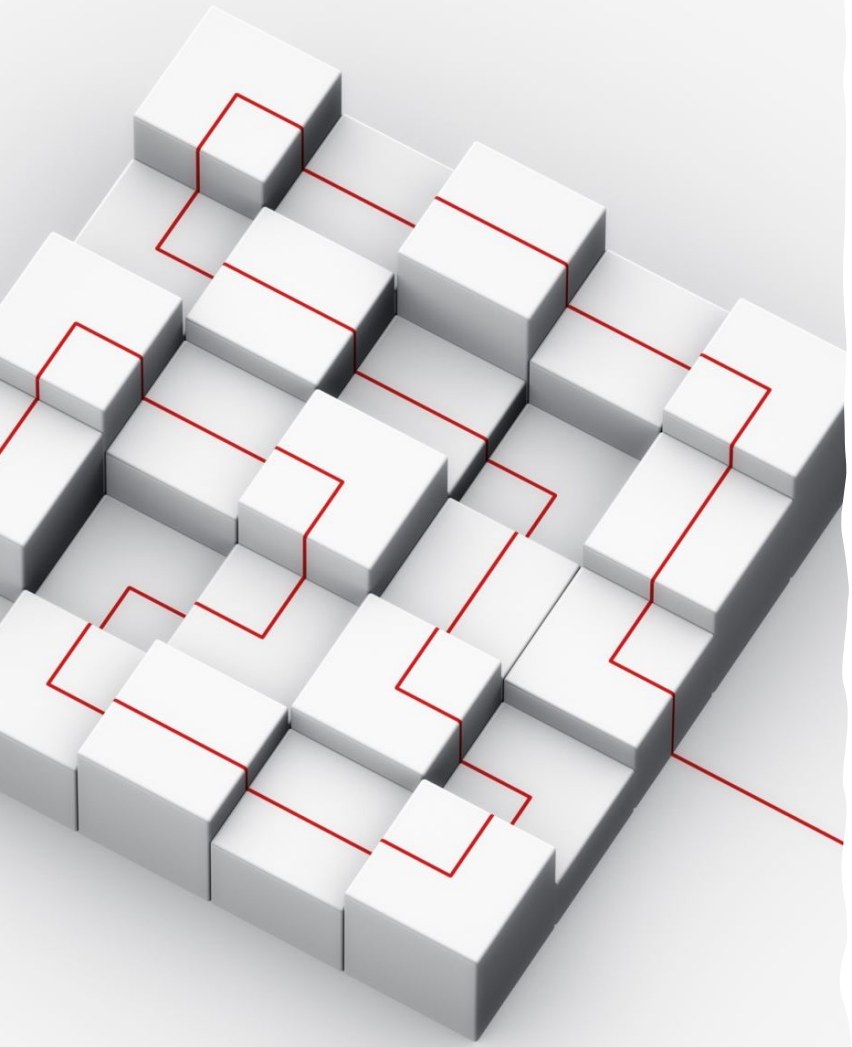
Composition – reuse of other components to build a new solution

Why design is hard?

→ Understand the problem



- Design is difficult because design is an abstraction of the solution which has yet to be created
- Non-deterministic - no two designers or design processes are likely to produce the same output
- Heuristic - Design techniques tend to rely on heuristics and rules-of-thumb rather than repeatable processes
- Emergent - The final design evolves from experience and feedback - it is an iterative and incremental process where a complex system arises out of relatively simple interactions
- Design is a wicked problem - it is one that can only be clearly defined by solving it



A Generic Design Process

- Understand the problem (software requirements)
- Construct a "black-box" model of solution (system specification)
  - System specifications are typically represented with use cases (especially when doing OOD).
- Look for existing solutions (e.g., architecture and design patterns) that cover some or all of the software design problems identified
- Consider building prototypes
- Document and review design
- Iterate over solution (Refactor)
  - Evolve the design until it meets functional requirements and maximizes non-functional requirements

search for solutions

Inputs to the design process

User requirements and system specification

- including any constraints on design and implementation options

Domain knowledge

- for example, if it's a healthcare application the designer will need some knowledge of healthcare terms and concepts

Implementation knowledge

- capabilities and limitations of eventual execution environment

Design Characteristics

- Minimal complexity – Keep it simple. Maybe you don't need high levels of generality.
- Loose coupling – minimize dependencies between modules
- Ease of maintenance – Your code will be read more often then it is written.
- Extensibility – Design for today but with an eye toward the future. Note, this characteristic can be in conflict with "minimize complexity". Engineering is about balancing conflicting objectives.
- Reusability – reuse is a hallmark of a mature engineering discipline
- Portability – works or can easily be made to work in other environments
- High fan-in on a few utility-type modules and low-to-medium fan-out on all modules. High fan-out is typically associated with high complexity.
- Leanness – when in doubt, leave it out. The cost of adding another line of code is much more than the few minutes it takes to type.
- Stratification – Layered. Even if the whole system doesn't follow the layered architecture style, individual components can.
- Standard techniques – sometimes it's good to be a conformist! Boring is good. Production code is not the place to try out experimental techniques.

Characteristics of a bad design

- RIGIDITY- It is hard to change because every change affects too many other parts of the system
- FRAGILITY- When you make a change, unexpected parts of the system break
- IMMOBILITY- It is hard to reuse in another application because it cannot be disentangled from the current application
- VISCOSITY: The law of least resistance when faced with a choice
  - Design viscosity: Hacks are easier/faster than preserving the design
  - Environment viscosity: Slow cycle time -> fastest choice

Design methods

Patterns play an important role in the design methods of today

A design pattern is a reusable solution to a commonly occurring design problem

A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations

Design patterns are adapted to the unique characteristics of the particular problem

Architecture Styles/Patterns

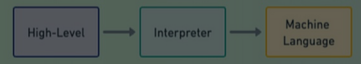
Design Patterns

Programming Idioms

Principles (SOLID, DRY, …)

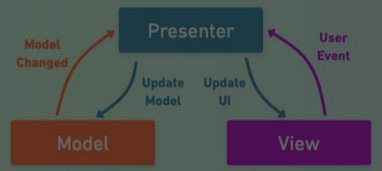
Orchestration Architecture

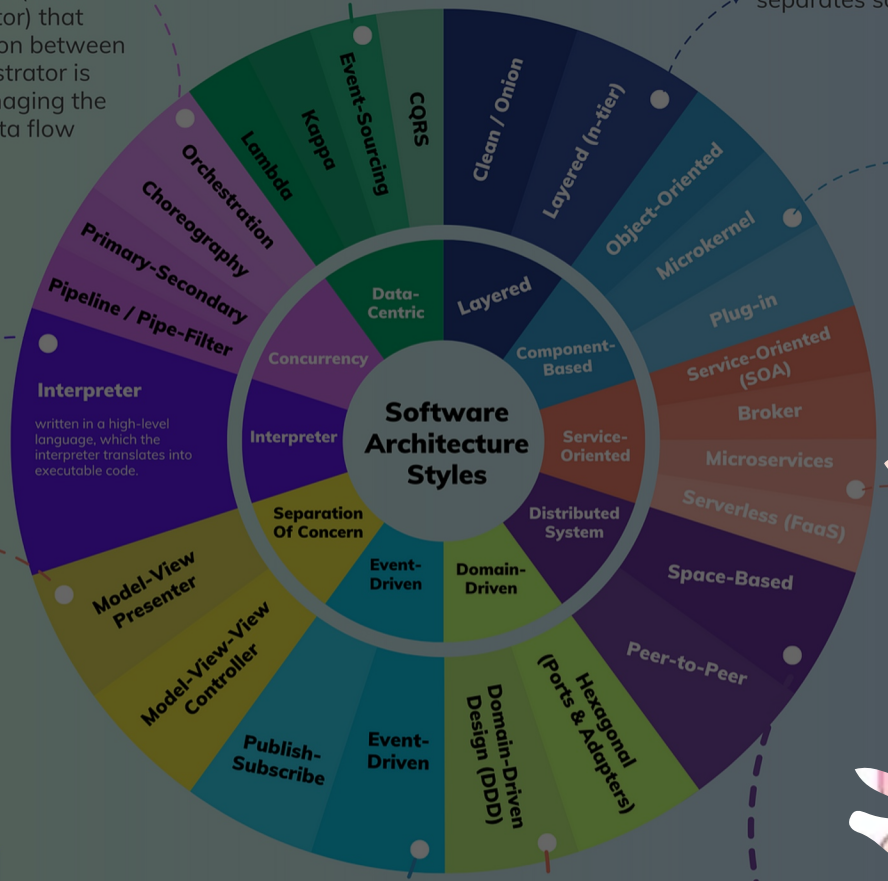
a central coordinator (often called an orchestrator) that directs the interaction between services. The orchestrator is responsible for managing the control flow and data flow between services.



MVP Architecture

derivative of the Model-View-Controller (MVC) pattern, which aims to separate the concerns of data management, user interface, and control flow.

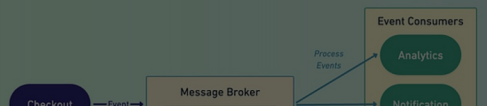




Next...

Event-Driven Architecture

promotes the production, detection, consumption of, and reaction to events.



DDD Architecture

focuses on the domain logic and complexity rather than the technology used.

Space-Based Architecture

This resolves the issues of consistency, reliability, performance, and scalability in large-scale distributed systems.

Layered (n-tier)

separates software in

Microkernel Architecture

separates a minimal core from extended and customer-specific