Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

What software architecture is about?

Software Architecture

©Juan Carlos Cruellas

Bibliography

[1] Robert C. Martin: "Clean Architecture. A Craftsman’s Guide to Software Structure and Design". Prentice Hall.

Introduction

- The term "architecture" when applied to software systems does not have a formal, unambiguous and universally accepted definition, so it is not strange that there is ambiguity in terms of what the concept covers.

- The term itself is taken from building sector and brings the idea of a structure, where everything is conditioned to obey physical laws (which limits choices range).

Introduction

- However, in software systems this limitation IS NOT PRESENT, and indeed there are bad software systems architecturally speaking , doing their job.

- A different question, though, is HOW EXPENSIVE is to maintain (i.e. fix errors and evolve them for satisfying new requirements from users) such software systems.

Introduction

- Architecture in software systems: the shape given to the system by its builders.

- The form of shape is defined when the architects divide the system into components, arrange them, and define the way they will interact when the system is running.

Introduction

- The aim of that shape is to properly support software system lifecycle, i.e:
  - Development.
  - Deployment.
  - **Maintenance**. Note bold and underline for emphasizing how relevant is architecture in this phase.

Screaming architecture

- Look at blueprints of a single family home generated by an architect. You will immediately notice that hey are the blueprints of a home.

- Look at blueprints of a theatre. You will be able to identify them as the blueprints of a theatre without needing anyone to tell you.

- **That occurs because the architecture blueprints scream what they are for**.

Screaming architecture

- **This should also be the case for complex software systems**.

- Good software systems architectures scream what the systems are for without requiring to go into the details.

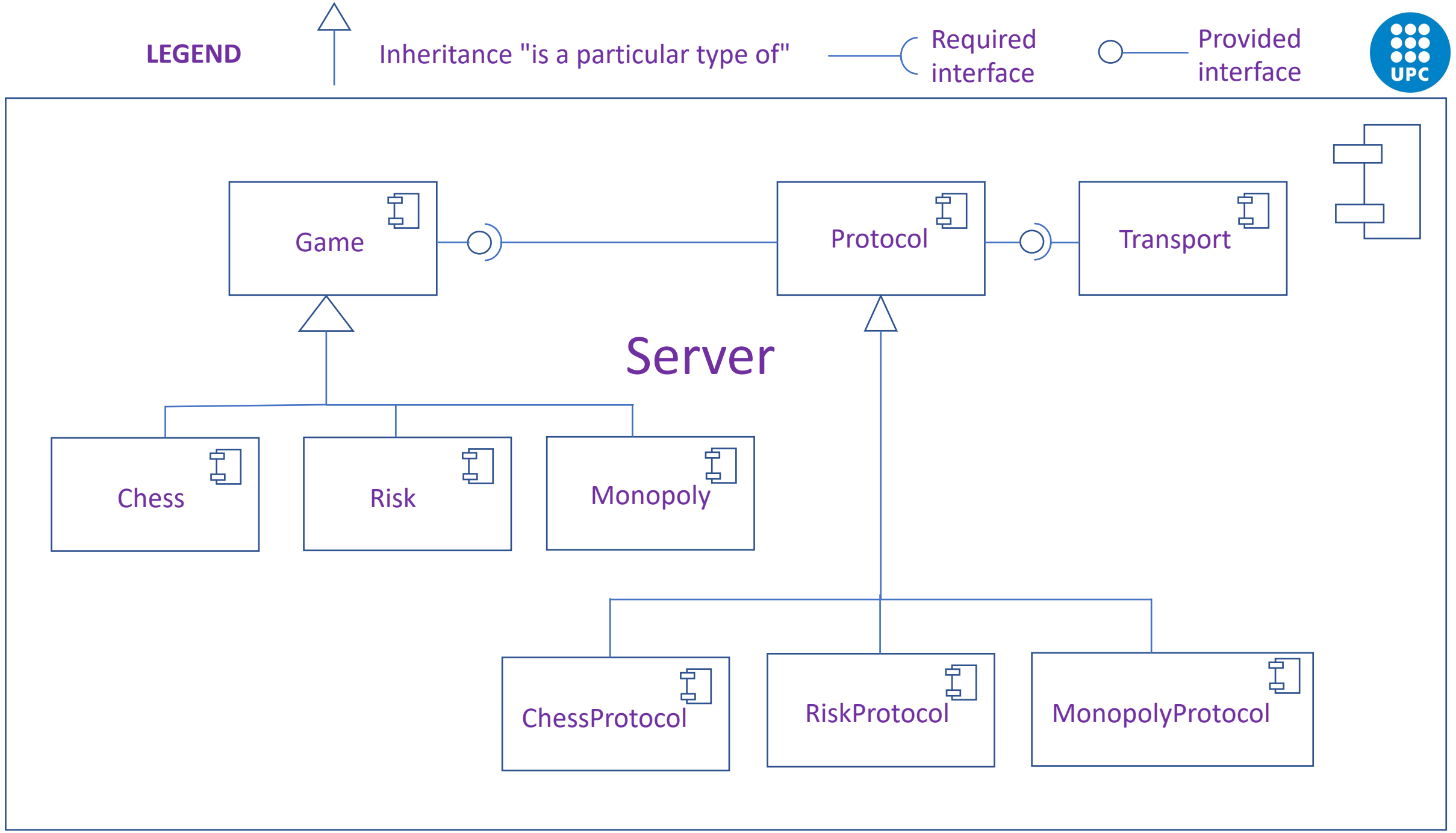
- If a software system electronically signs and validates signed documents, its architecture (its shape) should allow you immediately know that **without having to scrutinize all the source code files**.

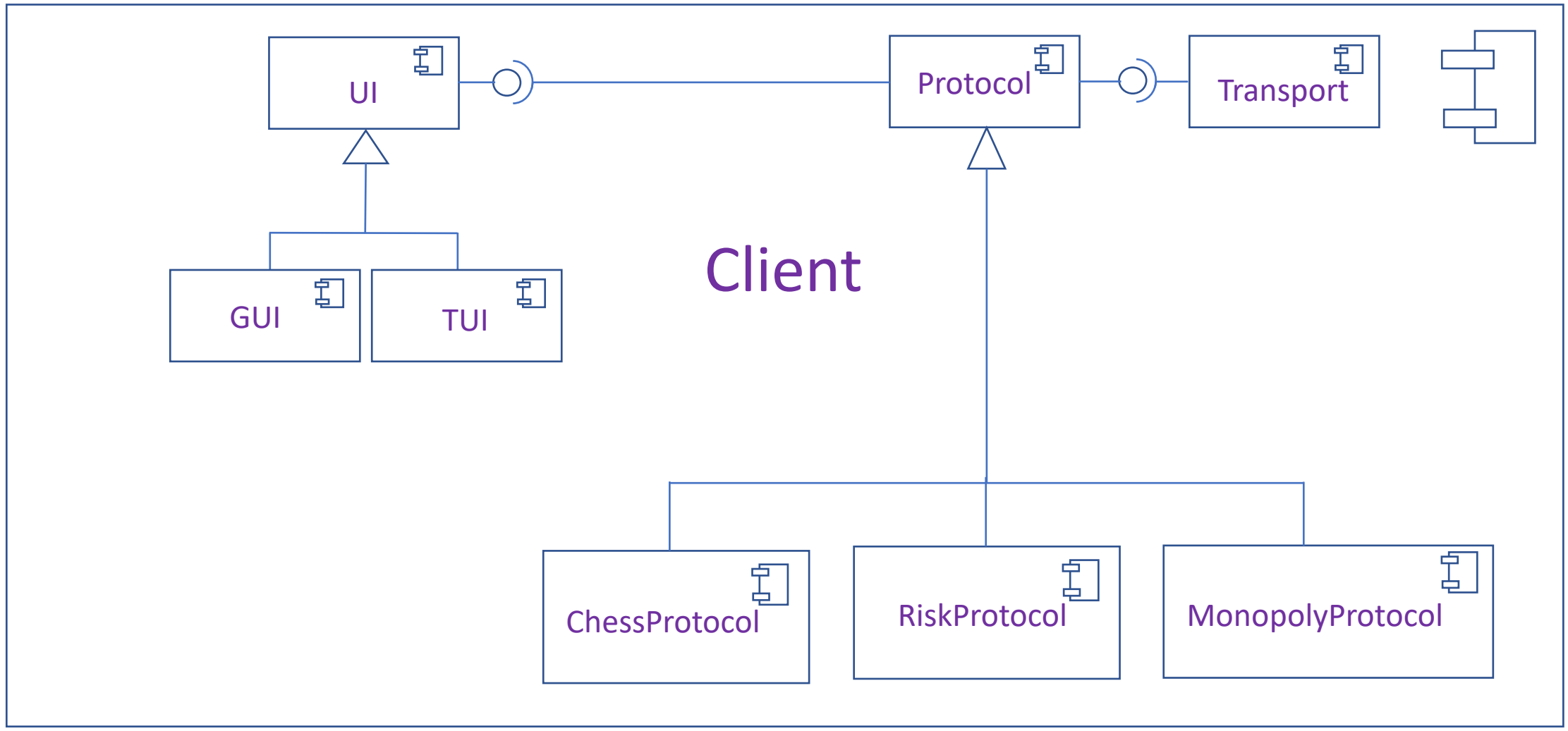
Screaming architecture

- But, also, as in building architecture, a single family home blueprint shows you where the kitchen, living room, dining room, bedrooms, bathrooms, etc (i.e. the different components, each one with its own purpose, of the home) are,
- A good software system architecture must show you:
  - The different components.
  - What the different components likely do.
  - Without having to analyse hundred/thousands of lines of code.

Screaming architecture

Look the two following slides and see if you are able to guess what the software system is about.


LEGEND
Inheritance "is a particular type of"
Required interface
Provided interface
UPC
Game
Protocol
Transport
Server
Chess
Risk
Monopoly
ChessProtocol
RiskProtocol
MonopolyProtocol


UI
Protocol
Transport
Client
GUI
TUI
ChessProtocol
RiskProtocol
MonopolyProtocol

Screaming architecture

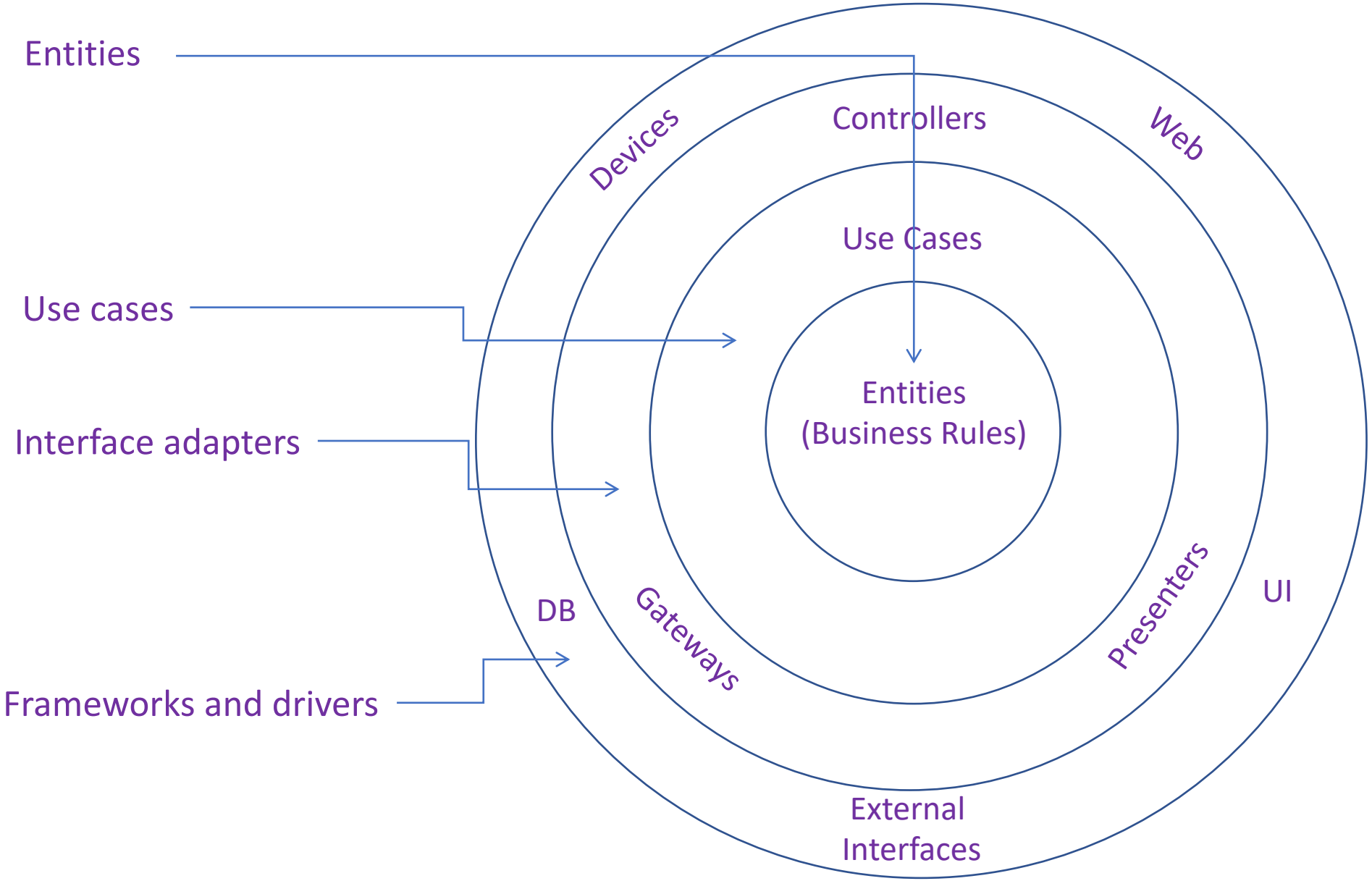
If you have been able is because that was a screaming architecture

Screaming architecture

But in software systems architecture, there is another element extremely important: **layering**

Regarding layering we will try to be aligned with the so-called (by Robert C. Martin) Clean Architecture

The Clean Architecture
(Robert C. Martin: "Clean Architecture").



Architectural concerns

- DESIGN.
  - It is impossible to achieve a good architecture without a former design of the solution.
  - Good architects generate designs properly applying well-known Design Principles and using well-known Software Design Patters.
  - **INTRODUCTION** to both OO DESIGN PRINCIPLES AND SOFTWARE DESIGN PATTERNS will be around 2/3 of our course roughly speaking.

Architectural concerns

- **Division of the system into components** (distribution of code), taking into consideration:
  - Their **internal cohesion**
  - Their **coupling with other components**
  - Good architects produce divisions taking into account ARCHITECTURAL PRINCIPLES that have proved to lead to good architectures when properly applied.
  - WARNING: time constraints will make that we can not deal very much with this very relevant issue during this course. References will be given for further reading.

Architectural concerns

- Proper processing of the former issues require an intensive **analysis of the problem to be solved** by the software system, which in real life, is anything but easy to conduct.
  - **Introduction to OO Analysis** shall be a relevant part of our course, 1/4 roughly speaking.

Final remark

- Skeptical? Thinking that in the end spending time in analysing the problem, designing the solution and structuring its components for getting an architecturally good software system is a waste of time?
- It is very unlikely that you think so if you have some labour experience in some serious firm that develops soundful software systems.
- If you haven't and you are skeptical, you very likely are forgetting something which is extremely relevant:
- If a software system is successful the longest phase of its life cycle is **maintenance**, i.e. fixing problems and/or adding functionality, and this implies **changing it**. **A bad architecture makes extremely expensive to properly maintain the software system**.

Final remark

- Finally, listen what **real experts** say.
- Brian Foote and Joseph Yoder (you can search them in Internet and see their levels of expertise):

  "**If you think good architecture is expensive, try bad architecture**".
- And listen also Robert C. Martin (you can also search him in Internet):

  "The only way to go fast, is to go well".
- Or, in other words:
  - We do not save money if we do not spend time in doing a proper analysis and design, on the contrary.
  - If we do not spend time in doing a proper analysis and design, we will have to spend much more time in fixing bad architecture and bugs.