Group 3:
Pablo Valdés Fernández
Andrés Álvarez Murillo
Diego Moragón Merallo

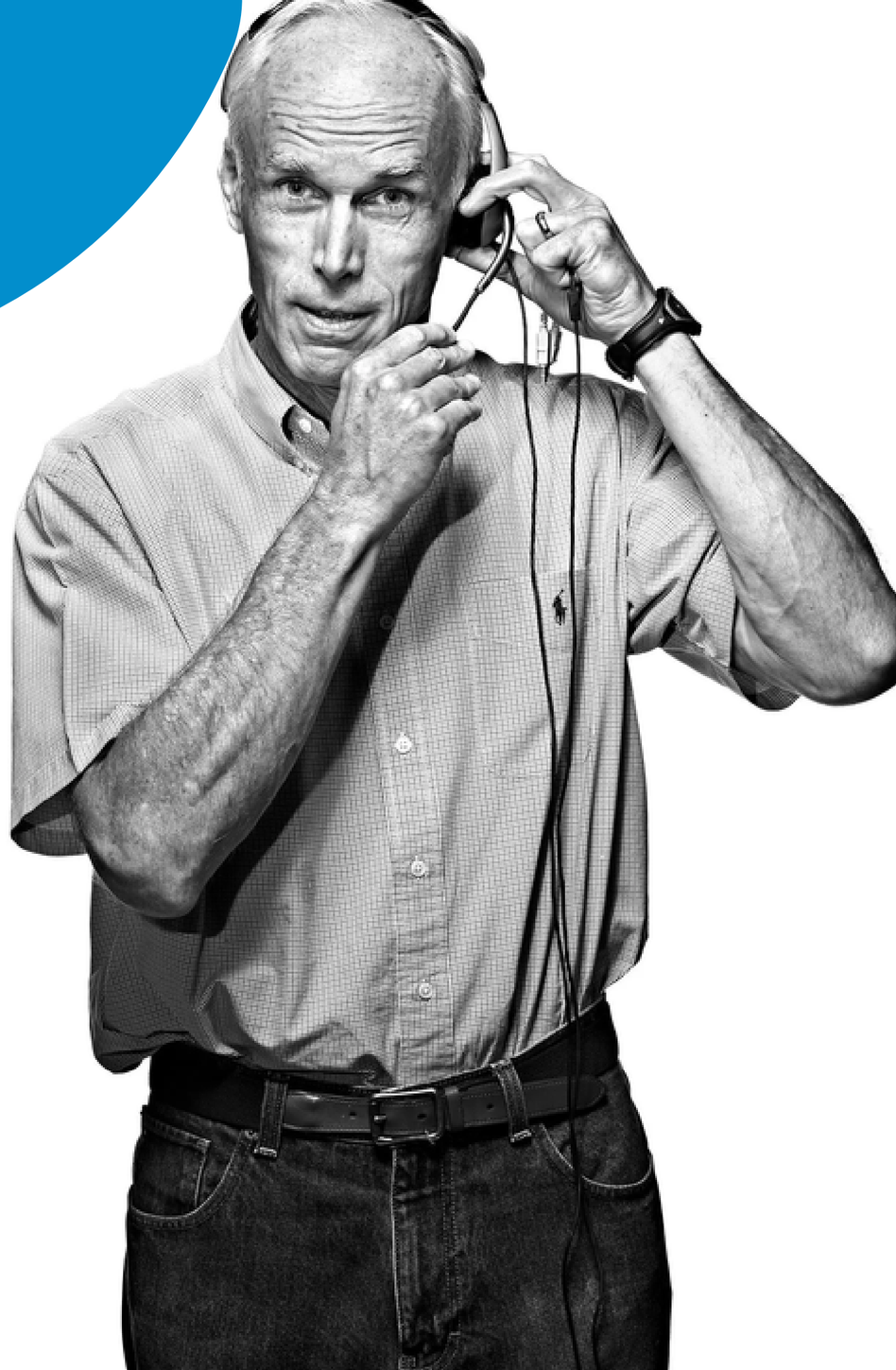
SOFTWARE DESIGN & MODULARITY

SOFTWARE ARCHITECTURE COURSE,
UNIVERSITY OF OVIEDO

CONTENTS

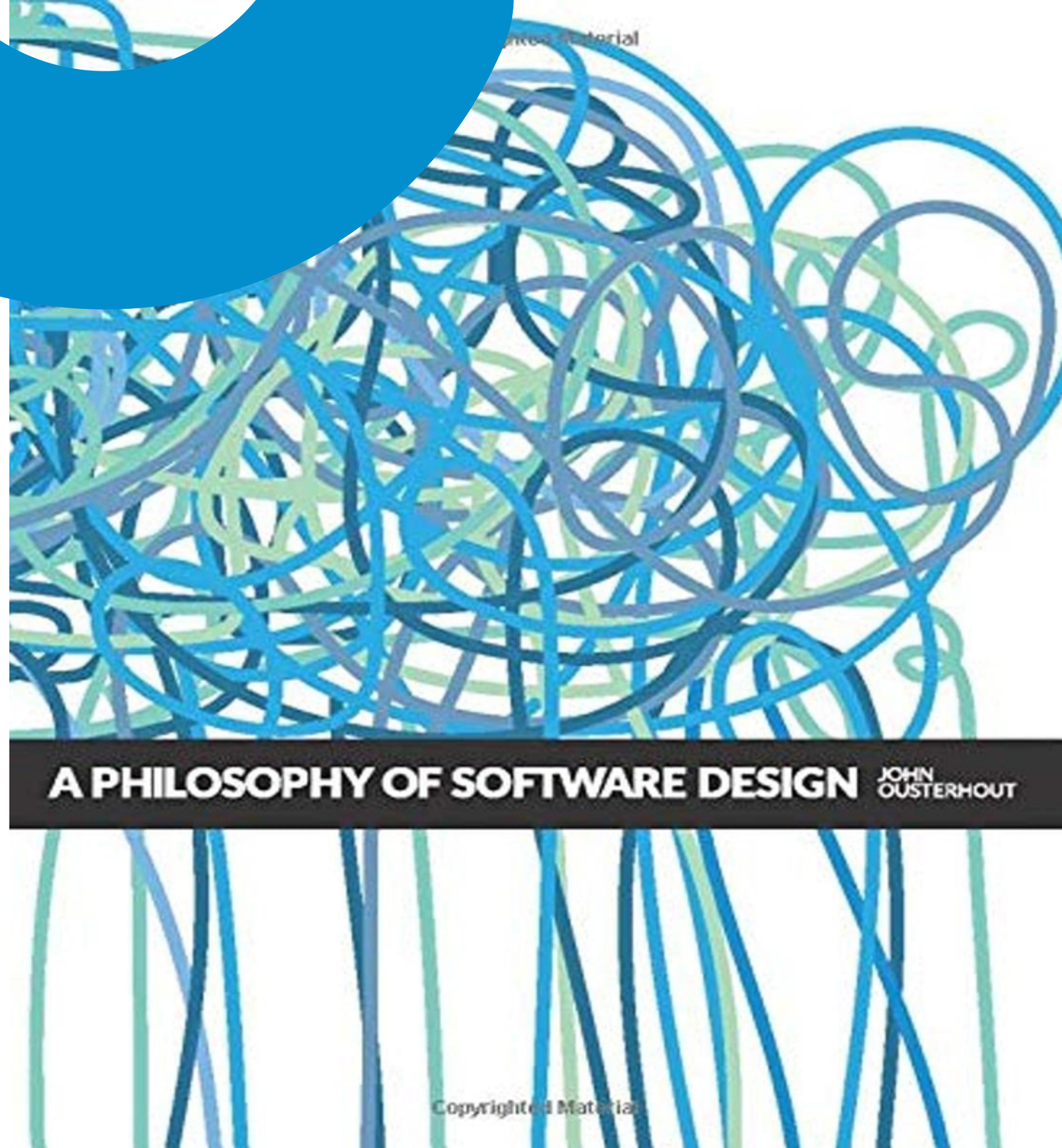
JOHN OUSTERHOUT

Professor of Computer Science in Stanford University
Founder of Electric Cloud
Interviewed about Software Design And Modularity in episode 520 of Software Engineering Radio



A PHILOSOPHY OF SOFTWARE DESIGN

Written by Ousterhout in 2018
Deals with relevant principles, problems and techniques that are relevant to design more efficient software
Used to prepare this presentation



SOFTWARE DESIGN DEFINITION

Designing a software system's structure, its components, and the interactions between them to satisfy the user's requirements

DESIGN TYPES

There are several types of software design, based on the abstraction level

1 **INTERFACE DESIGN**

2 **ARCHITECTURAL DESIGN**

3 **HIGH-LEVEL DESIGN**

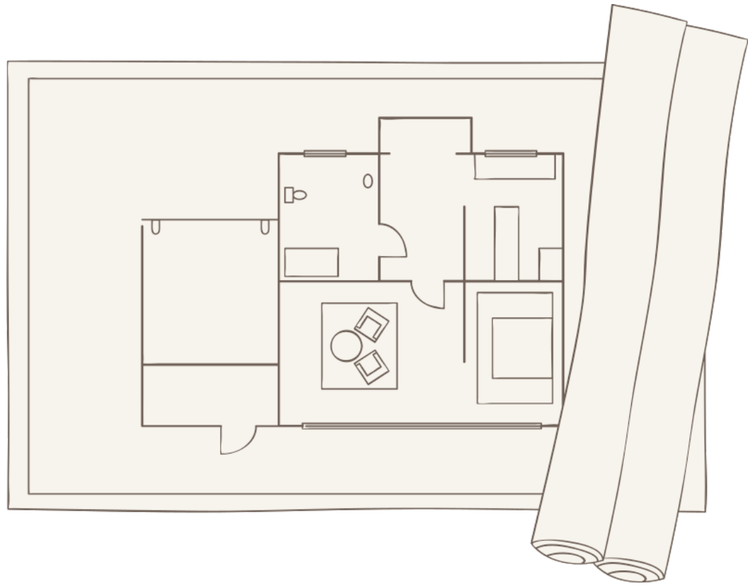
4 **DETAILED DESIGN**

INTERFACE DESIGN

Model the interaction between a system and its environment.
The inner components of the system are ignored.

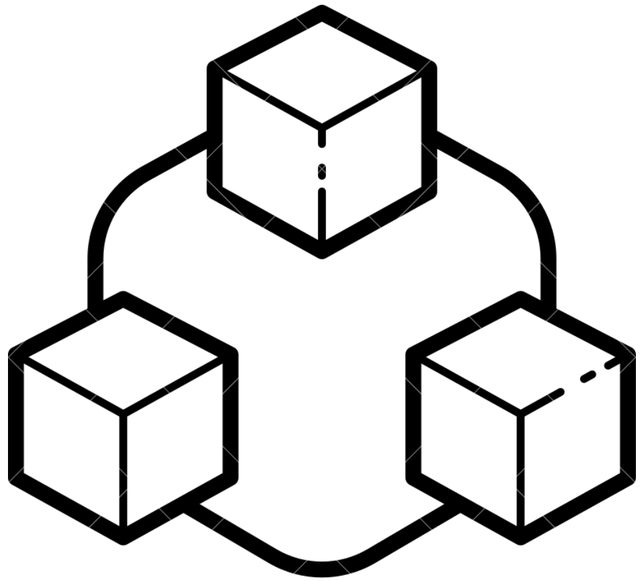
ARCHITECTURAL DESIGN

Analyze the components of the software system.
Identify the components of the system, as well as the interactions between them.
An idea of the solution domain is formed.



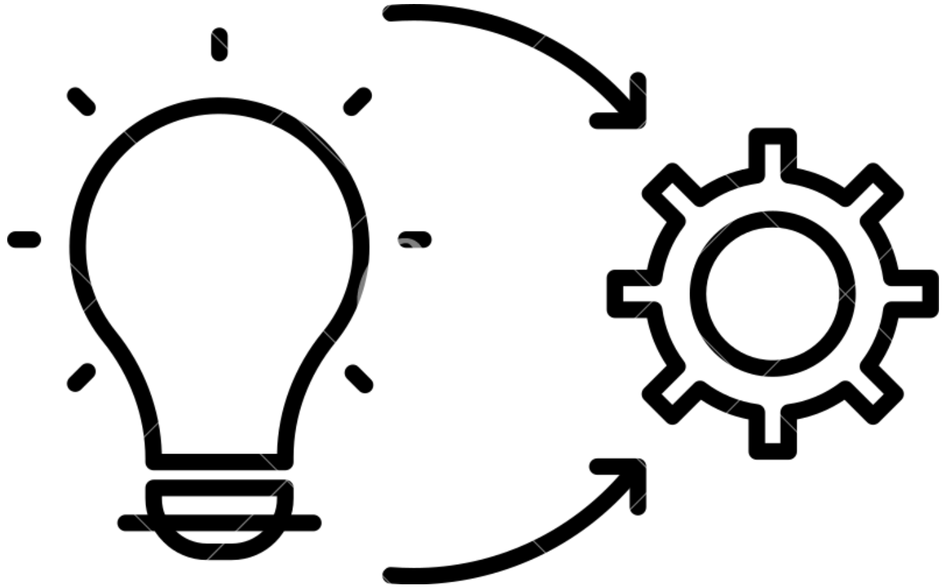
HIGH-LEVEL DESIGN

Break down the different components of the system. Identify the modules that form the components, and the interactions between them.



DETAILED DESIGN

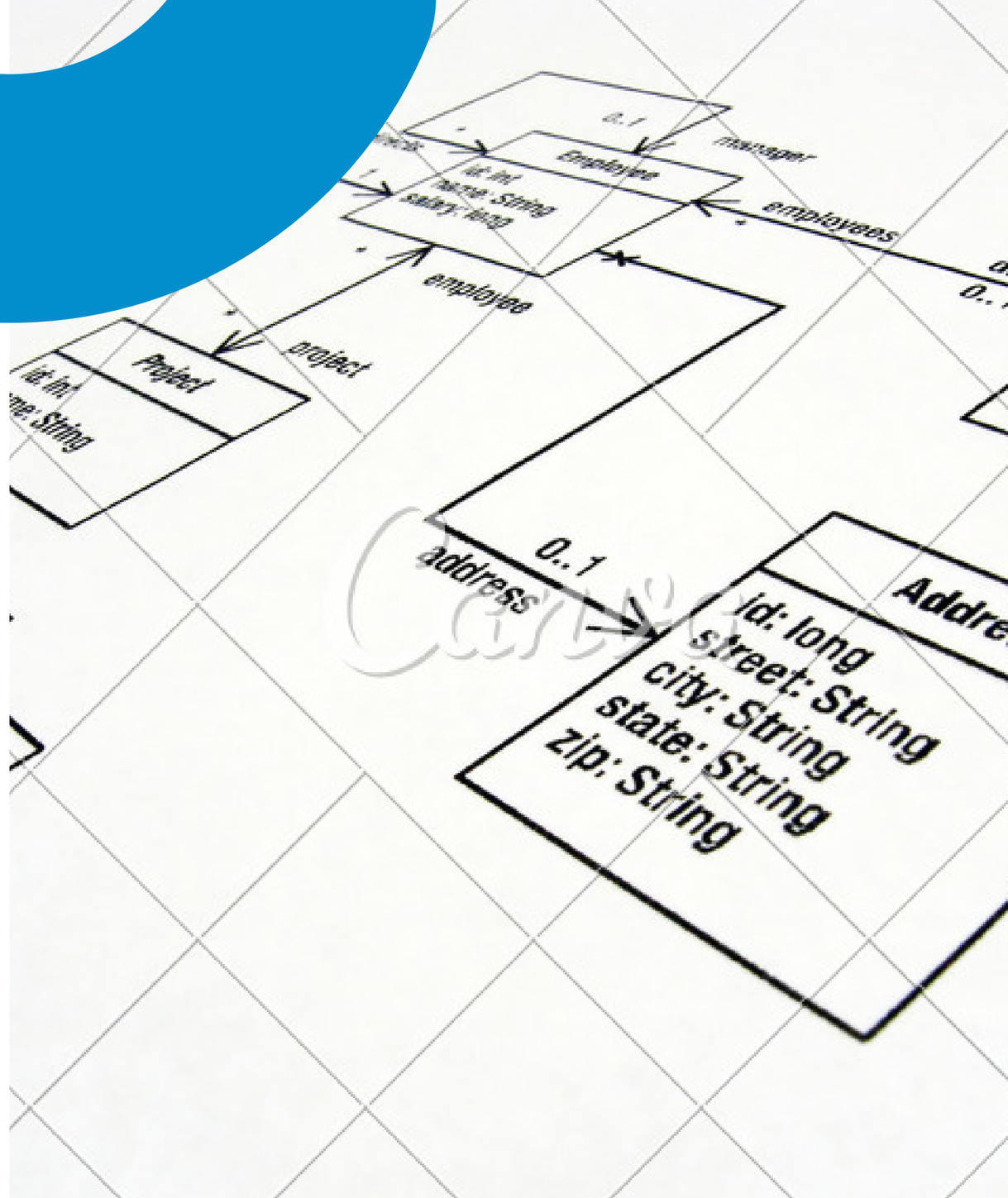
Study the implementation of the different modules.
Identify each module's logical structure and responsibilities.
Determine the interfaces each module will use to communicate with the rest



COMPLEXITY

Main factor limiting what can be built.

It's incremental, and increases as dependencies are added and functionalities are implemented.



HOW DOES COMPLEXITY AFFECT DEVELOPERS?

1

CHANGE AMPLIFICATION

Implementing a solution requires modifying code in many different places.

2

COGNITIVE LOAD

The higher the cognitive load, the more information developers require to successfully solve their tasks

3

UNKNOWN UNKNOWNS

It may not be obvious neither what code needs to be modified nor what information developers need to solve their tasks

PULLING COMPLEXITY DOWNWARD

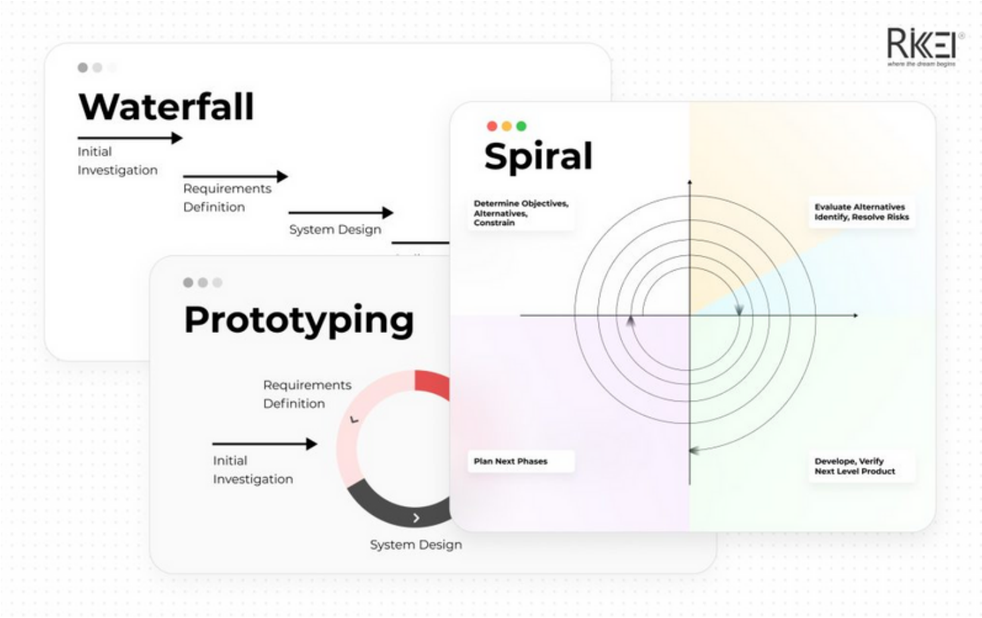
"It is more important for a module to have a simple interface than a simple implementation"
Unavoidable complexities should be dealt with in the module, rather than have it's users handle them.
There usually are more users than developers, so if the users deal with them complexity will be increased far more than if the developer does.

SOFTWARE DESIGN PHILOSOPHY

Set of design practices and ideas the development team has decided are important and will lead to a good design and how these principles should be implemented.

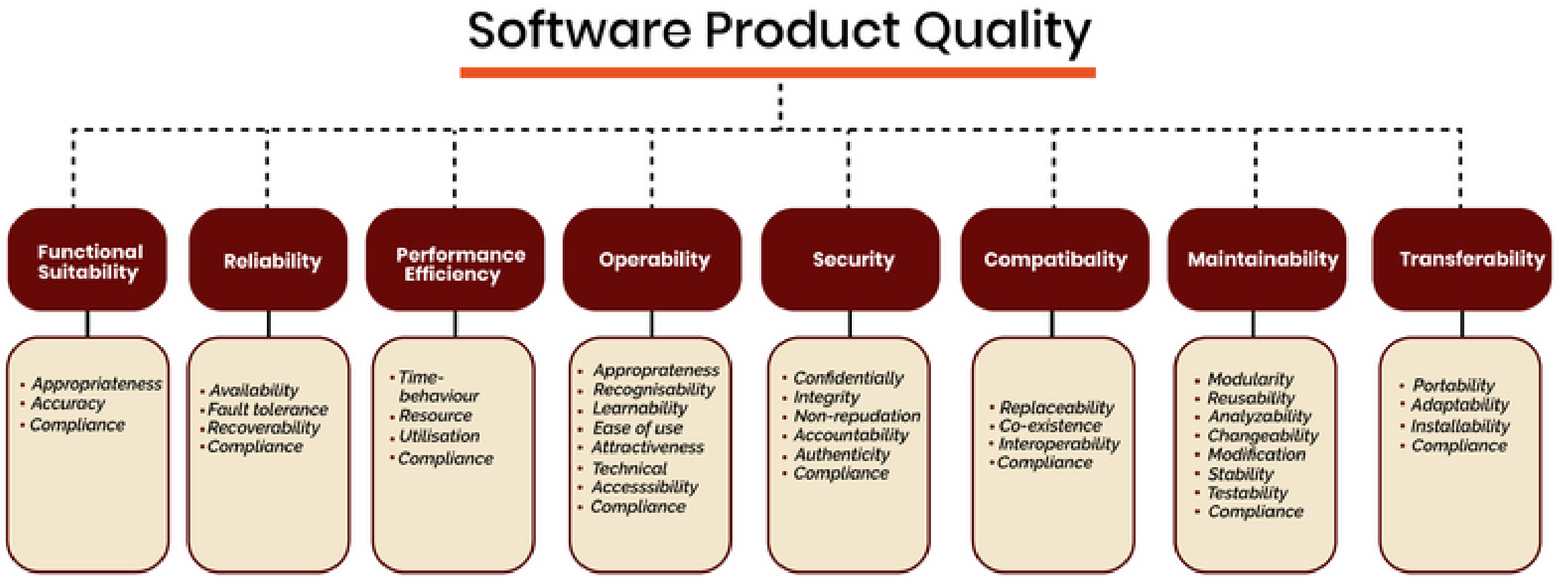
HOW SHOULD YOUR DESIGN BE?

One of the first things when starting a development should be thinking about the design even before writing any code an approach to the problem



CHOOSING WHAT IS IMPORTANT

A good design emphasizes what is important over what is deemed not important and designs around it



HOW WE APPROACH DESIGN

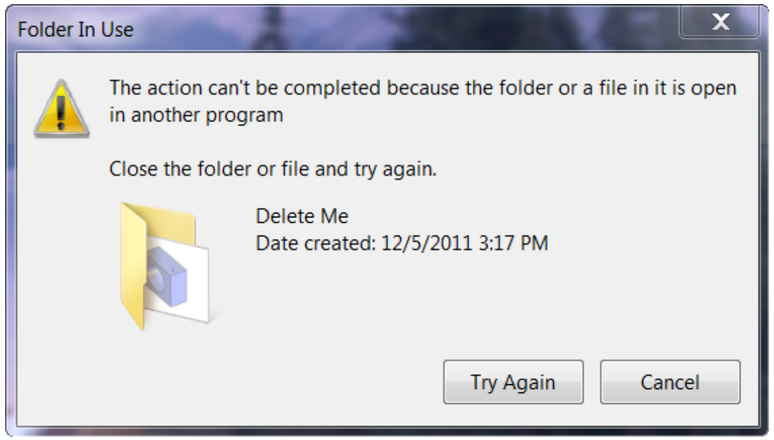
The design philosophy and the mindset in which we approach software development has great importance even on the smaller things that are not related to the overall design like just writing simple code.

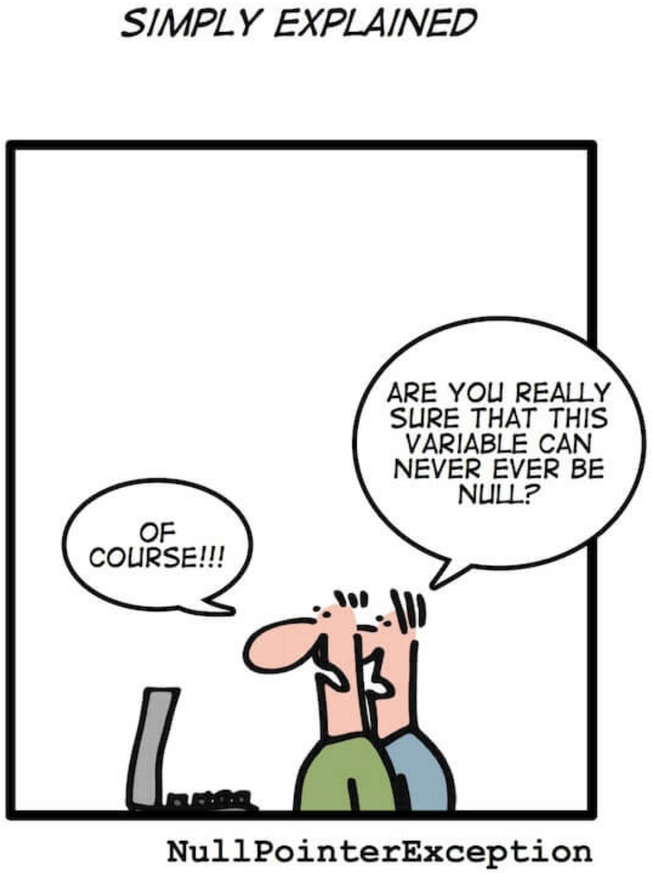
TACTICAL APPROACH

STRATEGIC APPROACH

DEFINING ERRORS OUT OF EXISTENCE

Principle based on eliminating the conditions where an exception would arise and let the problem resolve in a natural way if there is no need to throw an exception.

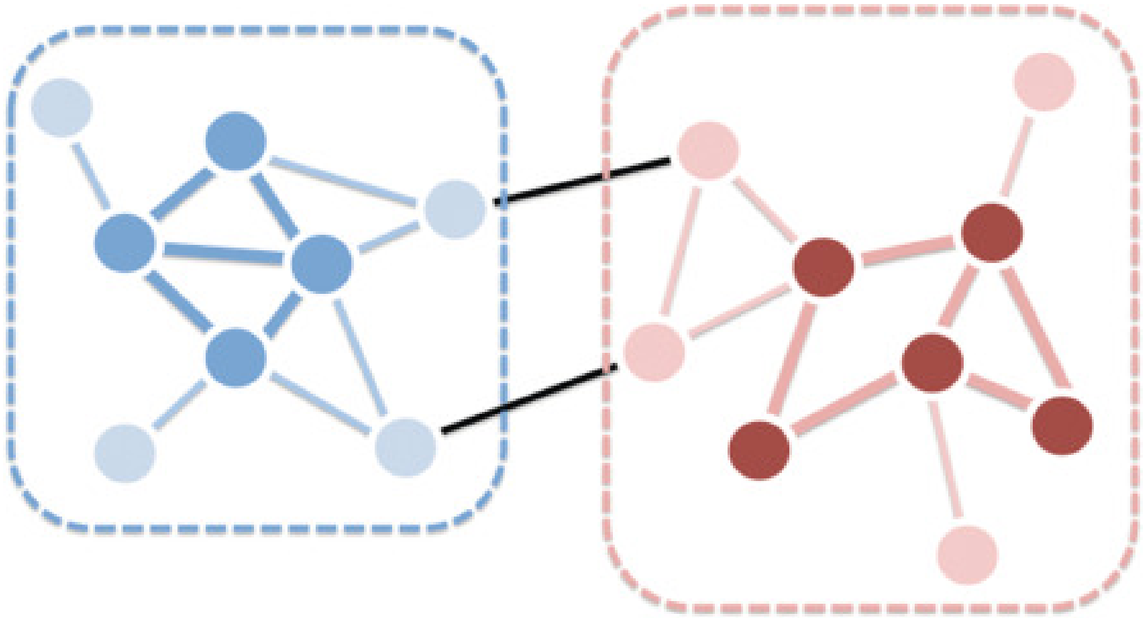




MODULARITY

MODULES

Pieces of code that can be independently created and maintained to be used and reused in different systems.

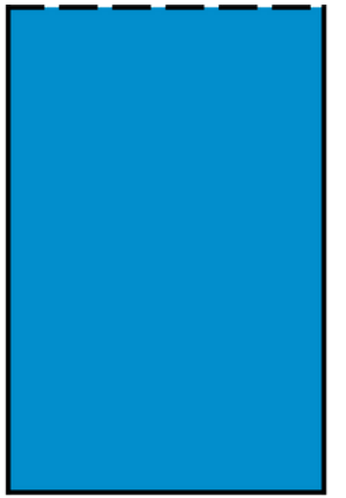


SHALLOW AND DEEP MODULES

The interface of a module should include everything that anyone needs to know to use the module.
We should avoid implementing shallow modules.

FUNCTIONALITY

INTERFACE



CHARACTERISTICS OF A MODULE

A software module has to be:

1

INDEPENDENT

2

REUSABLE

3

INTERCHANGEABLE

ABSTRACTION

The main goal of abstraction is to hide as much complexity as possible to allow programmers to focus on what's most important and relevant.
It's used to camouflage much of what is vital to making a program work.

LAYERS

They are logical separations of components or code.
Big systems don’t decompose naturally into perfect layers.

THE END

THANK YOU FOR LISTENING