# SOFTWARE DESIGN AND MODULARITY

Software architecture

Pablo Valdés Fernández – UO282655
Andrés Álvarez Murillo – UO278249
Diego Moragón Merallo - UO284016

# Software design and modularity

- ## Software design definition

Software design can be defined as the process of designing a software system’s structure, down to its objects and functions, as well as the way they interact, to make sure it will satisfy a series of predetermined requirements.

- ## Types of software design

  - ### Interface design

Modeling the interaction between the system and its environment. The system is a black box

  - ### Architectural design

Model the components of a system, as well as the interactions between them, to get an idea of the solution domain

  - ### High-level design

Identify the modules forming the components and model their interactions.

  - ### Detailed design

Detail each module’s structure, responsibilities, and interface.

- ## Complexity

This is the main design limiting factor, and it can force simple changes to require excessive modifications, excessive knowledge about the system, or make it difficult to know for sure what to modify.

It’s incremental, adding up in small chunks with every dependency and code line added to a system.

To prevent complexity from building up, a developer should handle as much complex functionalities as possible inside a module, and make the interfaces the users will need to use it as simple as possible. This is called pulling complexity downwards, and is also known as the “martyr principle”.

Main idea: It is more important for a module to have a simple interface than a simple implementation

- ## Software design philosophy

It could be defined as the set of principles that are important for a team and are believed to create a good design, as well as their implementation. There is no perfect one, and the best solution is always relative.

The philosophy and mindset in which developing is approached can affect even the smaller things like writing simple code.

It is necessary to plan an approach before beginning to write code, which is heavily dependent on the team’s philosophy. Choosing what is important (e.g., the quality requirements, the constraints and the desired features) should be the basis of the design and should be emphasized over what is not.

Code should be written in a way that allows it to be easily maintained, reviewed and understood. In some approaches, the code should speak for itself, not even needing comments. This works as an example for the importance of the mindset while developing software.

The principle of "defining errors out of existence” is based on eliminating the conditions that could allow for an exception to arise, allowing the problem to solve itself naturally. This way we can separate what is important and what should create an exception and what should not.

## • Modularity

A module is an independently created and maintained piece of code that encapsulates a set of related functions to be used and reused in different systems. It reduces complexity by providing a simple way to think of its functionality.

## • Module types

### ▪ Shallow

Complex interface, little functionality.

### ▪ Deep

Simple interface, complex and vast functionality. This is the ideal approach towards designing modules.

## • Module characteristics

### ▪ Independence

Creating and maintaining modules should have little to no impact on the others.

### ▪ Reusability

It should be possible to implement a same module in different projects.

### ▪ Interchangeability

It should be possible to replace modules while keeping the necessary changes in the rest of the system to a minimum.

## • Abstraction

Software should be incremented in abstractions, rather than features, in order to hide as much complexity as possible. This allows programmers to focus on what’s more important and relevant.

When improving code, entire abstractions, not just partial ones, should be built.

Hiding complexity not only does not damage a machine’s power and versatility, but in fact, creates more robust and useful results.

## • Layers

Logical component or code separations. The components of a layer depend on the components of a different one (layer A depends on layer B, layer B depends on C, C on D, and so on).

Applications are usually very complex, which makes it difficult to visualize all dependencies.

The more layers there are, the more dependencies, and the more complex a system is. Big systems, don't usually decompose into perfect layers naturally.

The most important thing is to use modularity to have properly encapsulated classes, with simple interfaces that streamline their usage as much as possible. A good example of this is HTTP modularity, with its basic protocols and simple yet great interconnectivity over the web.

## • Bibliography

Ousterhout, J. (2018). *A Philosophy of Software Design*. Yaknyam Press.

https://www.se-radio.net/2022/07/episode-520-john-ousterhout-on-a-philosophy-of-software-design/

https://sea.ucar.edu/best-practices/design

https://www.geeksforgeeks.org/software-engineering-software-design-process/

https://www.tutorialspoint.com/software_engineering/software_design_basics.htm

https://home.cs.colorado.edu/~kena/classes/5828/s07/lectures/21/lecture21.pdf

https://medium.com/@nathan.fooo/8-notes-pull-complexity-downwards-2b3b8075aed7

https://www.enginarslan.com/posts/a-philosophy-of-software-design

https://www.baeldung.com/cs/layered-architecture

https://venturebeat.com/programming-development/abstraction-in-programming-taming-the-ones-and-zeros/

https://www.modularmanagement.com/blog/software-modularity#:~:text=Software%20modularity%20is%20measured%20by,once%20and%20then%20maximize%20reuse

https://alexkondov.com/the-pholosophy-of-software-development/

https://royaldanishacademy.com/cephad/philosophy-design-introduction

https://uizard.io/blog/what-is-a-design-philosophy-and-how-to-create-one/#what-is-a-design-philosophy