Lección 1: Introducción.

Principios de arquitectura.

“Si los constructores construyeran los edificios de la misma forma que los programadores escriben los programas, el primer pájaro carpintero que apareciera destruiría la civilización”

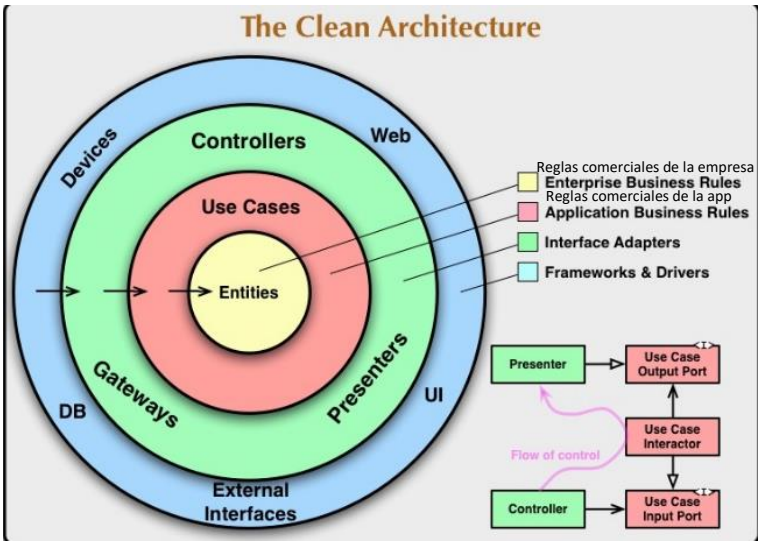
Gerald Weinberng.

Separación de Intereses.

El software debe ser separado en función de los tipos de trabajo que realiza.

Se hace referencia a la separación de responsabilidades en módulos según las acciones que realiza, ejemplo “Capa de datos, Capa de Presentación, Capa de Modelo de Negocios”, esto hace independiente entre si las piezas de código.

“*En clean arquitecture esto es diferente”.*



Ejemplo con un módulo.

“Identificar los elementos de interés que se van a mostrar al usuario y dar formato a estos elementos para hacer más notables”

En este requermiento tenemos dos comportamientos del sistema que deben ser separados, por que son dos funciones diferentes.

1. *Identificar los elementos de interés que se van a mostrar al usuario.*
   Modulo se implementaría en “casos de usos” por que se trata de conseguir datos según la lógica de negocios.

2. *Dar formato a los elementos para hacerlos más notables.*
   Función que se implementaría en presentación por que se trata de dar formato a la información.

Con esto podremos tener un orden en:

- **Lógica principal:** ¿Qué elementos deben ser mostrados al usuario? “Lógica y reglas de negocio, que se recomienda tener en un proyecto independiente, bueno en este caso de .net”.

- **Lógica de UI:** Dar formato a los datos a mostrar.
  “hay que pensar en dar formato a los datos, pero no en quien y como te darán los datos”

- **Infraestructura:** ¿Cómo obtener los datos?

Encapsulación:

Las diferentes partes de una aplicación deben de aislarse de otras partes de la aplicación.

Las capas y los componentes de la apelación deben poder ajustar su implementación sin afectar a sus colaboradores.

Esto es importante ya que así podremos obtener código independiente a otros módulos, por ejemplo: *El repositorio tiene la función de entregar datos, no importa el como debe pasar los datos si en duro o desde BBDD, para darlos a Casos de Uso mismo que solo le importan los datos y ambos módulos están encapsulados entre si…*

Ejemplo con ejercicio.

El la practica de gatos tenemos el módulo de “Objetos de Negocio”, mismo donde encontramos lo siguiente.

- **Entidades:** “Presentación de las estructuras de datos con las que trabajaremos”.

- **Interfaces:** “Contratos que el Repositorio por ejemplo debe de cumplir, por eso se define el **que debe hacer y entregar** y no el **cómo lo debe hacer**, claro esto por medio de interfaces”.

Pero también tenemos el Modulo de Repositorio, donde se hace la implementación del la interfaz o contrato correspondiente, en este modulo se crean las funciones de la interfaz y notros vemos como lo mandamos los datos ya sea con una BBDD o en código duro.

Con la encapsulación podremos lograr:

- *La encapsulación contribuye a lograr el acoplamiento flexible.*
    - *Los objetos y paquetes pueden ser remplazados con implementaciones alternativas. “por eso la importancia de la encapsulación en POO”.*
    - *Los componentes de aplicación y las propias aplicaciones deben exponer interfaces bien definidas*