TESTING

BUENAS PRÁCTICAS

TESTING

TDD

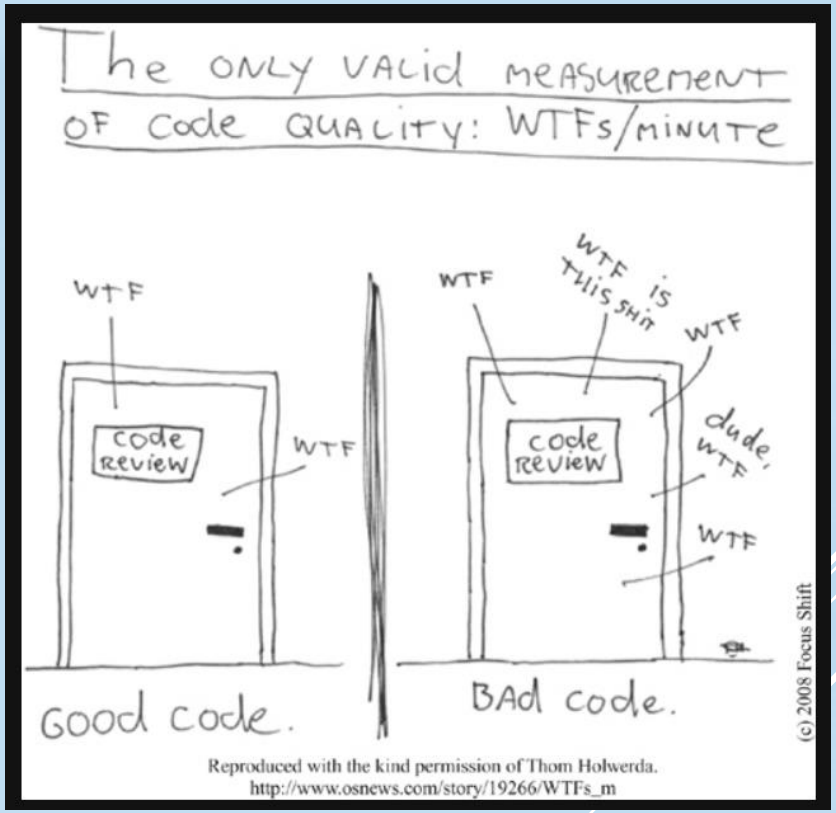
PATRONES DE DISEÑO

REFACTORIZACIÓN

BUENAS PRÁCTICAS

CLEAN CODE

(Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)



BUENAS PRÁCTICAS

NOMENCLATURA

¿Le pondrías cualquier nombre a tus hij@s?

- Nombres descriptivos
- Nombres afines (AccountList si no es una lista -> NO)
- Nombres fáciles de buscar
- Variables: sustantivos, métodos: verbos
- Sé específico (HolyHandGrenade para un método que borre -> NO)

BUENAS PRÁCTICAS

FUNCIONES

- Nunca debería pasar de las 50 líneas
- No debería haber más de dos bucles/condiciones anidadas
- Una función -> Una tarea
- Número de argumentos, nunca más de dos
- Nunca se debe duplicar Código (crea una función nueva)

BUENAS PRÁCTICAS

COMENTARIOS

- ¿Siempre se debería comentar código?
- Opiniones

BUENAS PRÁCTICAS

COMENTARIOS

- A favor:
  - Trabajo en equipo suele necesitar de comentar Código para que sea más legible
  - Algunas normativas legales obligan a añadir comentarios al Código
  - Las funciones abstractas que no se utilizan directamente, a veces pueden requerirlo
  - Comentarios en los test cases, pueden ser útiles
- En contra:
  - "No comentes mal Código, reescribelo" Brian W. Kernighan and P.J. Plaugher
  - Mantenimiento costoso
  - Generalmente, información redundante o innecesaria (fecha de creación de la función)
  - Comentarios confusos. Si lo vas a hacer, dedícale tiempo

BUENAS PRÁCTICAS

GESTIÓN DE ERRORES

- Usar siempre excepciones major que códigos concretos (PassportNotFound mejor que err321)
- Usar siempre un try-catch-finally para tener claro el alcance del error y el resultado final
- Dedica tiempo a definirlas, crea un protocol (un error 5xx no es lo mismo que un error 4xx en un servidor)

TESTING

¿ES NECESARIO CÓDIGO QUE PRUEBE TU PROPIO CÓDIGO?

TESTING

SÍ

La estadística dice que sin tests, el desarrollo suele ser de 30-70, siendo 30% del tiempo dedicado a crear y 70% del tiempo dedicado a corregir.

TESTING

Test unitarios

- El primer escalón de los tests
- Test de funciones
  - Casos extremos
  - Input que no pueden ser
  - Recorrer todos los caminos

TESTING

Test integración

- Testean funcionalidad, una labor
- Ejemplo: Detector de esquinas de un carné en una imagen

TESTING

Test aceptación

- Test de cliente
- Verifican objetivos que se presentan al cliente
- Ejemplo: en un servidor, todos los endpoints

TESTING

Test de carga

- Test de producto
- Brutales, Sistema sometido a la máxima presión
- Ejemplo servidor: muchos usuarios a la vez con una tasa de petición y descarga concreta

TESTING

Test de sistema

- Test de varios productos
- Comunicación entre varios servicios (móvil – servidor)

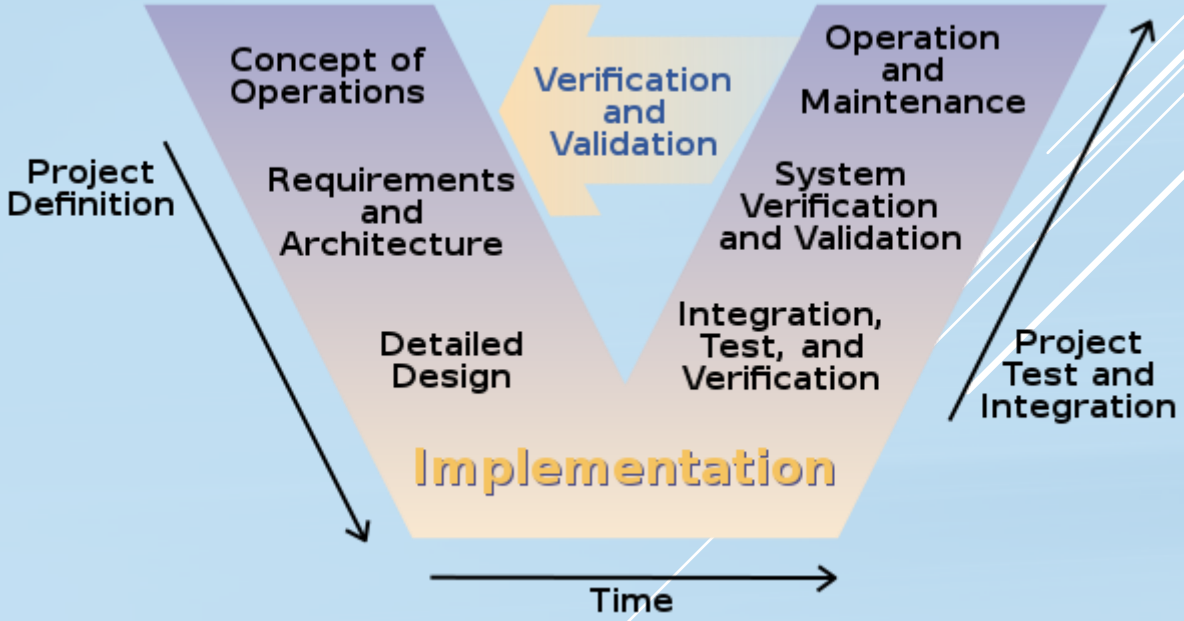
TESTING

Test “screenshot”

- Test web
- Se simulan comportamientos de cliente sobre una web
- Herramienta: Selenium

TESTING

Sistema de test en “v”



TDD

¿SE PUEDE HACER UN TEST SOBRE UNA FUNCIÓN O FUNCIONALIDAD QUE NO EXISTE?

- DEBATE

TDD

TEST DRIVEN DEVELOPMENT

- Los test guían el Desarrollo
- Código más objetivo y concreto
- ¿A veces modificas el test para que cuadre la función?

TDD

ALGUNAS DIRECTRICES

- Testea funciones mínimas posibles
- Sólo un assert por test
- Given When Then
- Red Green Refactor
- Mantenlo simple
- Límite de los diez minutos

PATRONES DE DISEÑO

BLOQUES DE CÓDIGO QUE CUMPLEN UNA MISMA CARACTERÍSTICA Y QUE SE HAN IDO REPITIENDO EN MUCHOS Y VARIADOS PROGRAMAS.

- **Singleton**: instancia única de clase
- **Event queue**: separa el envío/recepción de mensaje y el cuándo se procesa
- **Object pool**: ahorra recursos definiendo una pila de objetos
- **Prototype**: se define un objeto base para muchísima variedad de clases.
- **Observer:** una clase con muchos objetos como dependencia

REFACTORIZACIÓN

¿POR QUÉ Y CUÁNDO?

- Durante el Desarrollo, si una función se atasca
- Código legacy
- Ineficiencia
- Errores

REFACTORIZACIÓN

DETECTAR REFACTORIZACIONES

- Código duplicado
- Funciones grandes
- Clases grandes
- Funciones con muchos parámetros de entrada/salida
- Dificultad al cambiar algo en una función
- Switch statements
- Clases que no hacen nada
- Comentarios excesivos

BIBLIOGRAFÍA

1. Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)
2. https://www.guru99.com/v-model-software-testing.html, 20/03/2022
3. Modern c++ Programming with Test-Driven Development, Jeff Langr
4. https://es.wikipedia.org/wiki/M%C3%A9todo_en_V, 01-04-2022
5. Refactoring: Improve the design of existing Code, Martin Fowler
6. Game programming patterns, Robert Nystrom