INTRODUCCIÓN A LA CLASE

GO TESTING

Objetivos de esta clase

- Entender qué es Calidad de Software.
- Cómo evaluar Calidad.
- Comprender qué es la sustentabilidad y mantenibilidad del código.
- Tipos de test.
- Conocer los Principios de Calidad.

CALIDAD DE SOFTWARE

GO TESTING

// ¿Qué es Calidad?

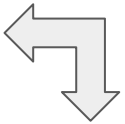
“Es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario.” - IEEE

Visión de Calidad

Los requerimientos en un software pueden ser “funcionales” (QUÉ) o “no funcionales” (CÓMO).

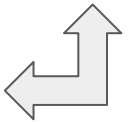
Cumplir con los requerimientos funcionales nos ayuda a saber si desde la perspectiva del cliente, el software tiene buena calidad. Por otro lado, cumplir con los no funcionales nos permite comprender la calidad desde una perspectiva más ingenieril. Para poder medir la calidad de un software de una manera más holística es que agrupamos los requerimientos en «dimensiones de calidad».

Satisfacción del Cliente



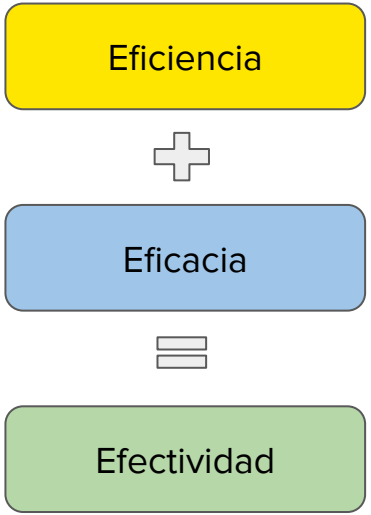
Calidad

Cumplimiento



Eficacia vs Eficiencia

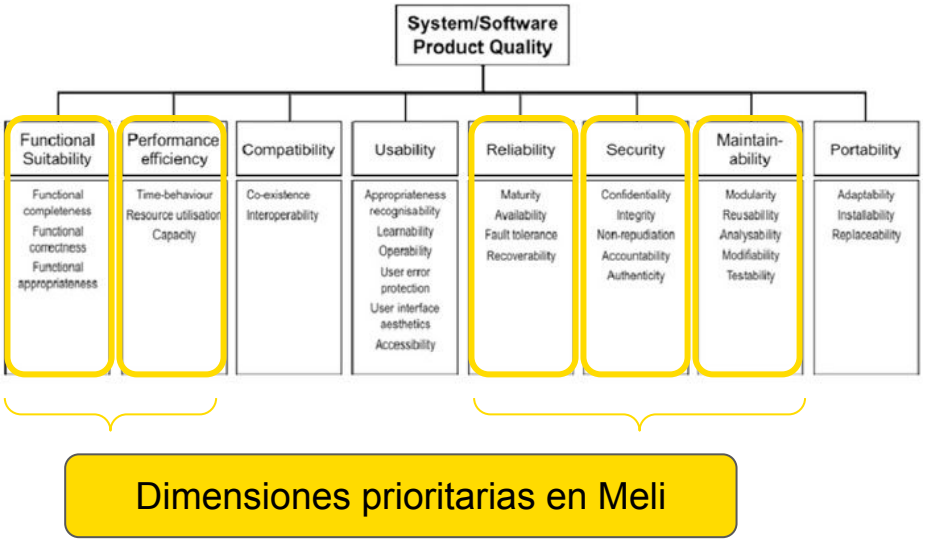
Mientras la **Eficacia (Qué)** busca el cumplimiento de un objetivo de la forma más directa posible, centrándose en el resultado y sin tener en cuenta los medios o recursos necesarios para lograrlo, la **Eficiencia (Cómo)**, agrega requisitos al cumplimiento del objetivo. Para que un objetivo se cumpla de forma eficiente, su resolución deberá hacer uso de la cantidad de recursos óptima.



¿Cuáles son las dimensiones de la calidad?

En la industria existen diversos estándares que enumeran las dimensiones a considerar para evaluar la calidad de un software. Por ejemplo el ISO/IEC 25010 que define ocho dimensiones.

- **Funcionalidad (Functionality)**
- **Rendimiento (Performance)**
- Compatibilidad (Compatibility)
- Usabilidad (Usability / Operability)
- **Fiabilidad (Reliability)**
- **Seguridad (Security)**
- **Mantenibilidad (Maintainability)**
- Portabilidad (Portability / Transferability)



CÓDIGO DE CALIDAD

GO TESTING

Mantenibilidad/Sustentabilidad

Es una dimensión que observa la **calidad** desde la perspectiva del **código fuente** del software.
Para entender el estado de esta dimensión es necesario tener en cuenta métricas como la **complejidad ciclomática** de nuestro código (cyclomatic complexity), la **cobertura de código** (code coverage), cantidad de **código duplicado**, la adhesión a **buenas prácticas** del lenguaje, etc.

// ¿Qué es Código Limpio?

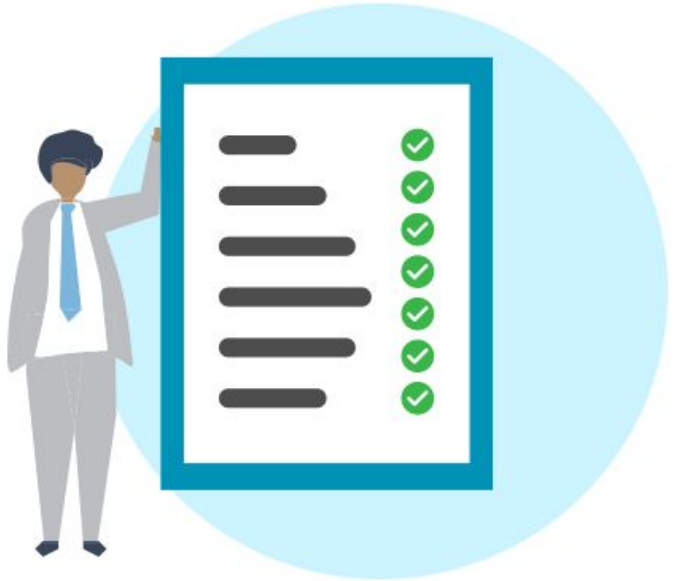
“El código limpio es simple y directo, y se lee como prosa bien escrita que no oscurece las intenciones del diseñador, sino que está lleno de abstracciones claras.” **Grady Booch, creador de UML.**

“El código limpio parece estar hecho por alguien que le importa” **Michael Feathers, autor de Working Effectively with Legacy Code.**

“Pasa todos los tests. No tiene duplicidades. Expresa las ideas de diseño del sistema. Minimiza el número de entidades como clases, métodos y similares” **Ron Jeffries, autor de Extreme Programming Adventures in C#” IEEE**

Código Funcional

El desarrollo de cualquier producto debe hacerse teniendo una idea clara del objetivo o resultado final esperado, y del comportamiento definido para dicho producto. Garantizar que nuestro código cumpla con los requerimientos, es garantizar una de las dimensiones de calidad: la Funcionalidad.



Performance en el Código

Otra dimensión de calidad es el Performance. Los tiempos de respuesta para la correcta ejecución de un proceso, son claves para un mejor desempeño del aplicativo. Mientras más rápido responda o procese un software, mejor será su performance. Durante el desarrollo nos encontramos con la necesidad de hacer elecciones que pueden afectar el performance, uso de dependencias externas, metodos de codificacion que pueden generar fugas de memoria y en consecuencia disminución del desempeño. Siempre debemos considerar cómo afecta el performance cada decisión que tomemos durante el desarrollo.

Código Confiable

El software es confiable cuando demuestra una alta probabilidad de funcionamiento sin fallas durante un período específico y en un entorno específico. La confiabilidad es una visión de la calidad del software orientada al cliente. Se relaciona con la operación más que con el diseño del programa y, por lo tanto, es más dinámico que estático. Tiene en cuenta la frecuencia con la que las fallas causan problemas. Medir y predecir la confiabilidad del software forma parte de una de las dimensiones prioritarias de Calidad.

Código Seguro

El código seguro es un atributo más de la calidad del producto que construimos y como tal es una responsabilidad **de todos**.

Secure Coding como práctica de programación tiene como fin anticiparse a todos los posibles puntos de fallas que podrían ser aprovechados por un atacante. Entendiendo que los defectos, errores y fallas lógicas son constantemente la causa principal de las vulnerabilidades de software comúnmente explotadas.

Código Seguro

Detectando vulnerabilidades:

- Una buena práctica de código seguro es **validar** todos los **inputs** del usuario.

- Mantener actualizadas - **testeando compatibilidad** - las librerías usadas. Esto nos permite reducir fricciones al intentar actualizar versiones depreciadas.

- **Evitar dependencias (librerías) con vulnerabilidades conocidas:**
  Según estudios el 70% de nuestro código es de terceras partes, en su mayoría librerías y frameworks open source.

TESTING

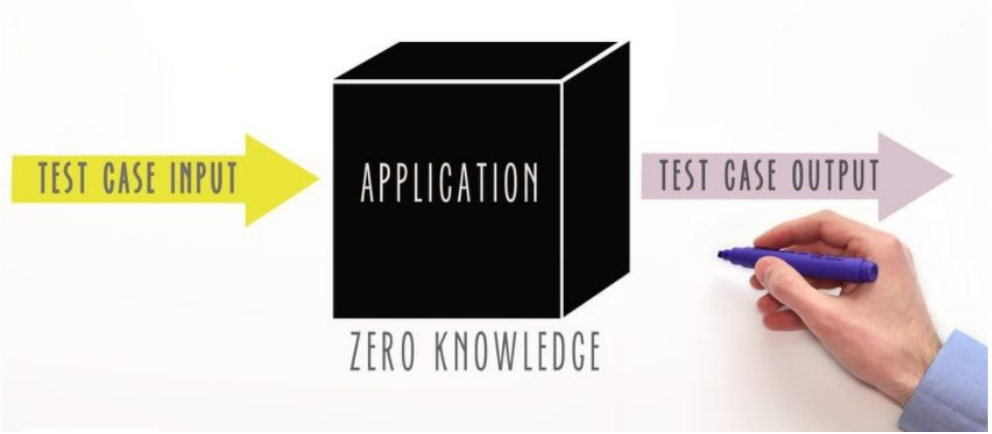
GO TESTING

// ¿Qué es Testing?

“Conjunto de procesos, métodos y herramientas para identificar defectos en el software alcanzando un proceso de estabilidad del mismo. El Testing no es una actividad que se piensa al final del desarrollo del software, va paralelo a este.”

Black Box Test

Es un método de testing en el que la estructura, diseño o funcionamiento **interno**, es **desconocida** por quien ejecuta la prueba. Su objetivo principal es probar la funcionalidad del código, evaluando las respuestas y reacciones **(comportamiento)** del componente testeado ante distintos escenarios.



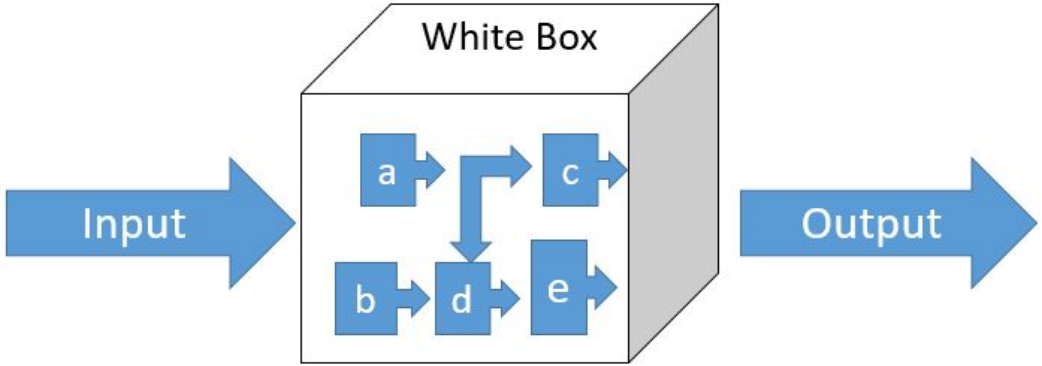
Black Box Test

Se aplica generalmente para:

- Tests Funcionales: validar que el software cumpla con un requerimiento dado.
- Tests No Funcionales: **únicamente** para medición y evaluación de performance, usabilidad y escalabilidad.
- Test de Regresión: pruebas que se ejecutan luego de cualquier cambio en el código, para verificar que dicho cambio no haya afectado o dañado el comportamiento esperado.

White Box Test

En este método, quien ejecuta la prueba conoce y tiene visibilidad sobre el código. Es la contraparte del Black Box, ya que las pruebas no se ejecutan desde una perspectiva de usuario final. Se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente.



White Box Test

Este método se aplica para probar el **flujo, seguridad y estructura** del código. Y así poder detectar vulnerabilidades en el código, comprobar la correcta implementación de cada método o función y validar que el flujo de datos se comporte de acuerdo a lo esperado (condicionales, iteraciones y respuestas).

Los White Box Tests pueden aplicarse por medio de:

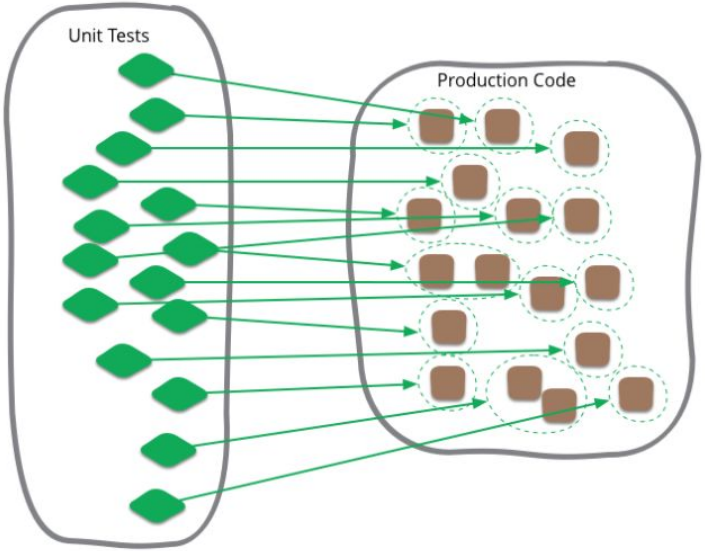
- Tests Unitarios.
- Tests Integración.

TIPOS DE TEST

GO TESTING

Test Unitarios

Son las pruebas aplicadas a sólo una parte del código. Las pruebas unitarias o unit testing son una forma de comprobar que un fragmento de código funciona correctamente, o algún método o función específica. **Consisten en aislar una parte del código y comprobar que funciona a la perfección.** Son pequeños tests que validan el comportamiento de un objeto y la lógica.
Este tipo de testing consiste en probar de forma individual las funciones y/o métodos (componentes y/o módulos que son usados por nuestro software).



Test Unitarios

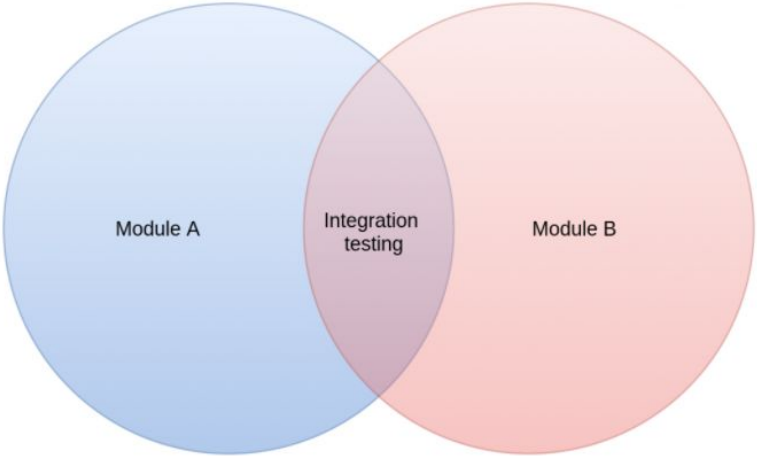
Sus beneficios principales son:

- **Facilitar los cambios en el código** al detectar modificaciones que pueden romper el contrato en el caso de refactorizaciones. Es más fácil hacer un cambio y probar instantáneamente si está afectando alguna funcionalidad.
- **Encontrar bugs** probando componentes individuales antes de la integración, así los problemas pueden ser solucionados antes de que impacten otras partes del código. Reducen el tiempo de debugging.
- **Proveen documentación,** ayudan a comprender qué hace el código y cuál fue la intención al desarrollarlo.
- **Mejoran el diseño y la calidad del código** invitando al desarrollador a pensar en el diseño del mismo, antes de escribirlo (Test Driven Developement - TDD).

Test de Integración

Son aquellos test que prueban la comunicación entre distintos componentes o capas de la aplicación. El objetivo es comprobar que todos aquellos bloques de código que fueron probados de forma unitaria, interactúen y se comuniquen entre sí generando los resultados esperados.

Los test de integración exponen el funcionamiento general de la aplicación, permitiendo evaluar su diseño, desempeño y comportamiento.



Test de Integración

Las pruebas de integración verifican que los diferentes módulos y/o servicios usados por nuestra aplicación funcione en armonía cuando trabajan en conjunto.

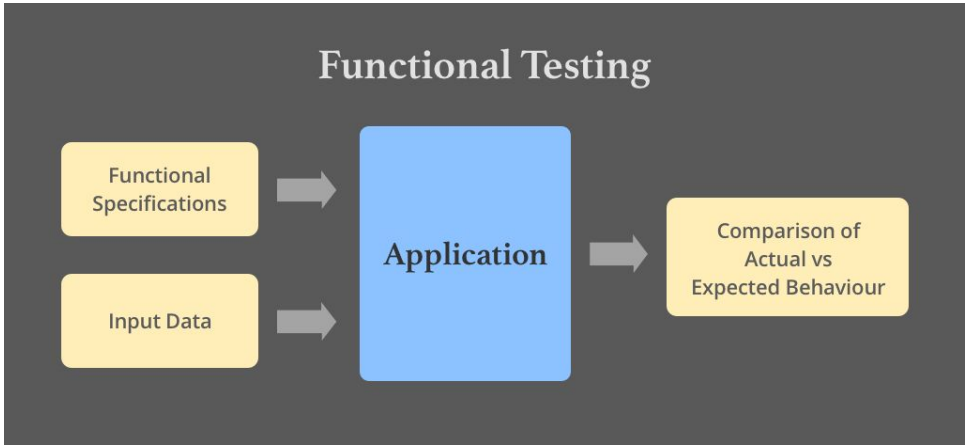
Por ejemplo

- Pueden probar la interacción con una o múltiples bases de datos,
- Asegurar que los microservicios operen como se espera.

Test Funcionales

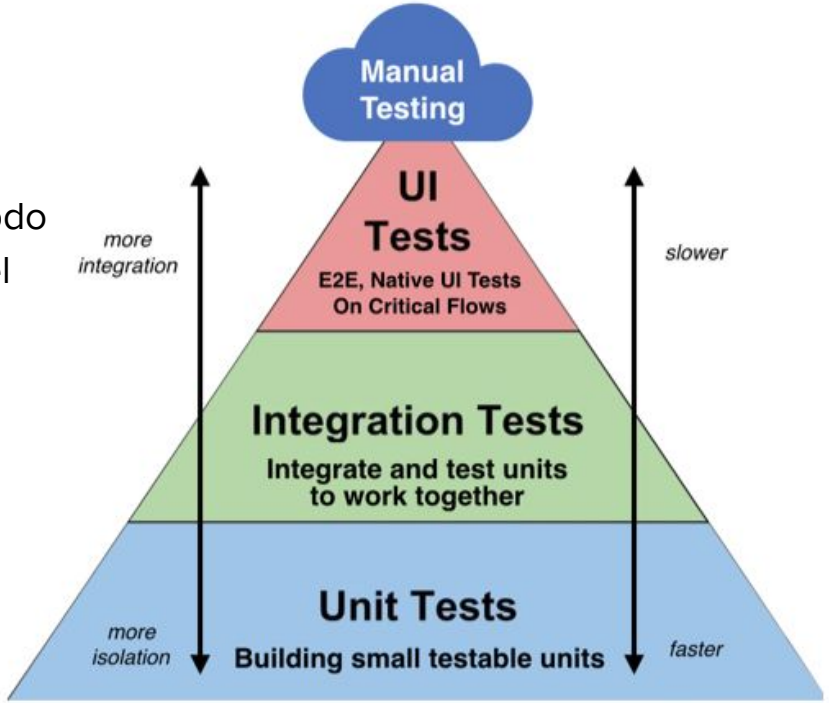
Son las pruebas que se basan en la entrada y salida del software, con el objetivo final de comprobar que la respuesta del software ante cada escenario, coincida exactamente con el resultado esperado. Son test que validan el comportamiento funcional del software.

La funcionalidad debe evaluarse cuando una pieza de software ya previamente cumplió con la aprobación de pruebas unitarias y funcionales.



¿Cuándo aplicar cada Tipo?

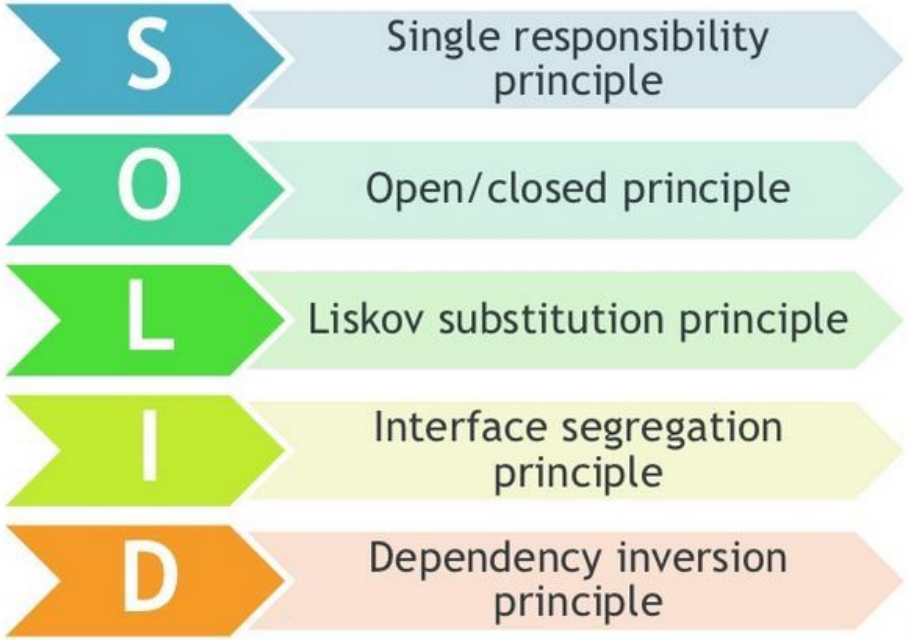
El testing no es una actividad que se ejecuta después del desarrollo, sino que acompaña todo el proceso de ingeniería de software. Desde el desarrollo inicial, hasta la aceptación como software productivo.



PRINCIPIOS DE CALIDAD

GO TESTING

Principio S.O.L.I.D.



Un objeto solo debería tener una única responsabilidad.

Los objetos deben estar abiertos para su extensión, pero cerrados para su modificación.

Los objetos deberían ser reemplazables por objetos hijos (de sus subtipos) sin alterar el correcto funcionamiento del programa.

Muchas interfaces más específicas son mejores que una interfaz de propósito general.

Desacoplar los objetos abstractos de sus implementaciones.

Principio F.I.R.S.T.

F	**Fast**	**Rápidos**: Es posible tener miles de tests en tu proyecto y deben ser rápidos de correr.
I	**Isolated/Independent**	**Aislados/Independientes**: Un método de test debe cumplir con los **"3A" (Arrange, Act, Assert)** o lo que es lo mismo: **Given, when, then**. Además no debe ser necesario que sean corridos en un determinado orden para funcionar.
R	**Repeatable**	**Repetibles**: Resultados determinísticos. No deben depender de datos del ambiente mientras están corriendo (por ejemplo: la hora del sistema).
S	**Self-Validating**	**Auto-Validados**: No debe ser requerida una inspección manual para validar los resultados.
T	**Thorough**	**Completos**: Deben cubrir cada escenario de un caso de uso, y no sólo buscar un coverage del 100%. Probar mutaciones, edge cases, excepciones, errores,

Gracias.

IT BOARDING

BOOTCAMP

Autor: Nelber Mora
Email: nelber.mora@digitalhouse.com
Última fecha de actualización: 03-07-21

IT BOARDING
BOOTCAMP