

Tipos de Pruebas

Hay una prueba para cada situación.

"Escribe tests. No demasiados. Principalmente de integración."

 Kent C. Dodds

Tipos de Pruebas



Manuales -> Programadas

- ✗ Dependemos de las personas
- ✓ Se pueden configurar y lanzar automáticamente

Técnicas -> Funcionales




- ● Se puede comprobar el rendimiento, la seguridad, usabilidad...
- ✓ La función del software, su utilidad.

Unitarias -> De integración -> De inicio a fin

- ✓ **unitarias**: Pruebas de caja blanca que verifican una función, una clase o un componente.
- ● **de integración**: Pruebas de caja blanca que verifican que varios componentes funcionan bien juntos.
- ✓ **de inicio a fin**: Pruebas de caja negra que replican el comportamiento de un usuario ante un sistema completo.

| Otras: de regresión, de humo, de aceptación...

Después -> Durante -> Antes

-  **Después** o mucho después *legacy*. Es costoso, pero imprescindible para un *refactoring* y muy habitual en un *end to end*
-  **Durante** es aburrido pero necesario para las pruebas de integración.
-  **Antes** El conocido como *TDD* para pruebas unitarias o *BDD* para las de integración. Menos costoso, más divertido y con mucho mejor diseño resultante.



Qué hay que saber para programar tests.

1

Mantra

- El código de prueba no es como el código de producción: diseñalo para que sea simple, corto, sin abstracciones, agradable de leer. Uno debe mirar una prueba y obtener la intención al instante.





2 Siglas y conceptos

- SUT: *System (Subject) Under Test*. Lo que se está probando.
- DOCs: *Depended On Components*. Lo que se necesita para que funcione el SUT.

3 Secciones: Arrange, Act & Assert (AAA Pattern)

- **Arrange:** Prepara y organiza lo que necesitas.
- **Act:** Ejecuta el código y obtén una respuesta.
- **Assert:** Verifica que la respuesta es la esperada.

4 Cuestiones: Given, Should, Actual, Expected.

- **Given:**  Texto. Condiciones de la prueba. (*Arrange*)
- **Should:**  Texto. Funcionalidad esperada.
- **Actual:**  Variable. El resultado obtenido. (*Act*)
- **Expected:**  Variable. La respuesta esperada. (*Assert*)

5 Test Doubles: Simuladores para no depender de las dependencias DOC.

- **Dummy:** Datos requeridos para que el SUT funcione, pero que no se usan durante la prueba. *(Carga previa de una base de datos)*
- **Stub:** Un objeto que cumpliendo una interfaz de un DOC tiene una respuesta constante y predeterminada. *(Responder como lo haría un llamada http)*
- **Fake:** Un objeto que realiza una funcionalidad coherente pero simplificada de un DOC. *(Simular una base de datos en memoria)*
- **Spy:** Cuenta las llamadas a una función o método. *(Comprobar que se ejecuta una acción un determinado número de veces)*
- **Mock:** Monitoriza el uso de un objeto y las llamadas a una función junto con sus argumentos. *(Simular un envío de correo completo)*

6 Comprobaciones: igualdad, existencia, comparación, pertenencia, excepciones y negación

- **igualdad:** El valor actual es igual al esperado.
- **existencia:** El valor actual existe.
- **comparación:** El valor actual es mayor o menor que el esperado.
- **pertenencia:** El valor actual contiene o está contenido en el esperado.
- **excepciones:** Se espera que una excepción sea lanzada.
- **negación:** Niega cualquiera de los anteriores.

7 Consejos generales

- incorpora herramientas: Puedes empezar de cero, pero hay muchas ayudas.
- evita arreglos globales: Cada prueba deber ser autónoma e independiente.
- datos realistas en los fakes: Nada de *foo bar baz asdf*
- usa etiquetas o códigos: Útil para buscar resultados o pre filtrar pruebas.
- public black box: Prueba los métodos públicos.
- evita los mocks: Mejor usa *Stubs* y *Spies*.
- haz alguna prueba: Esto no va de todo o nada.

Herramientas

Utilidades para probar aplicaciones desarrolladas con tecnología web.

Puppeteer

[Puppeteer](#) es excelente para manipular y simular cualquier actividad con el navegador ideal para e2e no funcional.

Cypress

[Cypress](#) es un framework de pruebas funcionales de integración o e2e. Se ejecuta en el navegador independiente del código bajo prueba.

Jest

[JEST](#) es un framework muy popular porque requiere *zero configuration*. Es muy ligero y sencillo. Ideal para *unit testing* y *TDD*.

Otros

- **Playwright** automatizador de diversos navegadores al estilo Puppeteer.
- **Karma** es un ejecutador de pruebas muy interesante para integración continua.
- **Jasmine** muy completo y bueno para user-behavior por su expresividad
- **Mocha** muy utilizado para NodeJS.
- **Chai** librería muy adecuada para BDD con NodeJS.