ISW

Jueves 02-10-25 - MELES

TDD- **Test Driven Development**

Es un método para desarrollar sw. NO es una metodología de testing.

`Requerimiento – Análisis – Diseño – Implementación – Prueba – Despliegue`
Sw es humano intensivo. Lo más caro es el trabajo , el esfuerzo (la gente que tiene que hacer el sw)
El costo más significativo es el costo del esfuerzo.

IMPLEMENTACION - entre 33 y 35 horas del total del proyecto

PRUEBA - es lo más costoso. Entre el 30 y 50% del costo total del proyecto

- depende del tipo de sw.
- (es medio utópico eso del costo, la mayoría de las empresas lo hacen al 30 sobre las horas de implementacion, no sobre el proyecto total)

- **Casos de prueba**:
  - son más especificas que los escenarios de casos de prueba. Esos escenarios son el input para que se diseñe el caso de prueba.
  - Un caso de prueba es la identificación de situaciones concretas + los datos de seseo necesarios para poder determinar si uno o más criterios de aceptación se cumplen o no para cierto criterio.
  - ES EL ELEMENTO MÁS SIGNIFICANTE QUE SALE DE LA ACT DE PRUEBA.
  - Es más detallado que el requerimiento

¿Si los casos de prueba los diseños en base a los requerimientos, por que no adelantarlo?

- No esperar a que finalice la implementación. Sería más eficiente en términos de calendario, no de esfuerzo (esfuerzo es lo mismo)
- Es un cambio cultural

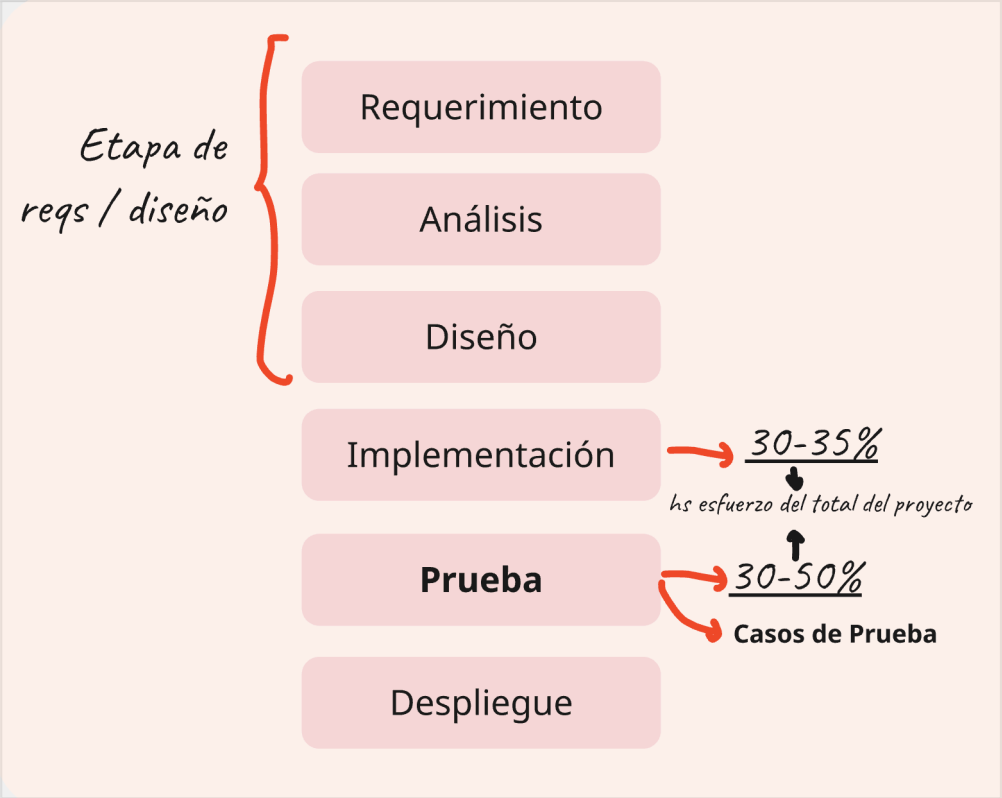
La confirmación de la US está materializada en los casos de prueba

Se decía que Rea, análisis y diseño eran **etapa de requerimientos o de diseño**

Poquito de historia:
Antes desarrollador actuaba como cirujano. El tipo labura y después se desentendía de todo el resto, del soporte y de la prueba
Surge Devops, se involucra más el dev.
VOLVER A MENCIONAR ESTO CUANDO YA HAYAMOS VISTO CASO DE PRUEBA

Etapa de diseño medio que se puede evitar, se puede hacer oral, y no meter tanto detalle. Caso de prueba no se debería evitar.
Agilismo establece hacer la etapa de diseño sin tanto detalle (las US) y pasar

el esfuerzo de meter detalle a los casos de prueba.
Estoy obligado a entender lo que voy a programar. Mejora la calidad del código, y me aseguro que el código que voy a entregar va a hacer lo que tengo que hacer.



Hay cuatro niveles de prueba (seguimos viendo el martes)

1. Prueba unitaria: lo hace el propio desarrollador .
2. Prueba de Integración: (Prueba de interfaces). Si junto todo lo que ya probé, debería andar. Veo que es lo que pasa cuando se tienen que comunicar.
3. Prueba de sistema o versión: pruebo contra los requerimientos funcionales y no funcionales.
4. Prueba de aceptación de usuario (**UAT**):

- la gente no suele probar los requerimientos no funcionales
- UAT lo debería probar cliente (PO)
- Las primeras tres, los debería hacer el técnico
- Prueba unitaria es la única que se hace en la etapa de implementación
- Sistemas e integracion, en la etapa de prueba

El TDD es uno de los 13 principios del **Extreme Programming (XP)**. Empezó de ahí, después evolucionó a ser una metodología propia.

Evolucionó a **ATDD** (Acceptance Test Driven Development) , y **BDD** (Behavior Driven Development)

- en simple, todos buscan adelantar para que tiempo sea más eficiente

TDD
Tiene un ciclo de tres ... con el código (RED-GREEN-REFACTOR).
Primero construyo algo para que falle, veo en que puedo mejorar y después

refactoreo.

- No escribir línea de código hasta que hayas hecho un test case que falla
- Con un test que falle es suficiente
- No vas a escribir más código que el necesario para que pase el test (YAGNI!)

—> REFACTORING
Forma de desarrollar que apunta a mejorar la calidad interna del componente que construiste. No le cambias la funcionalidad, sigue haciendo lo mismo que antes, nada más que ahora lo hace mejor.
BAJO ACOPLAMIENTO, ALTA COHESIÓN
No alcanza con que compile. Lo aplicamos para que quede lo más prolijo posible.
Está metido en el TDD

Cada cosa que se corrige introduce 3

PRÁCTICO de TAXI MOBILE

Pre condición: seteo

Nombres deben ser claros, no cortos. Que quede bien claro la situación de prueba.

```
def test_nombre_us_camino_resultado(success/fail)()
```

Cuando ya definimos las precondiciones, definimos los pasos

Pasos del caso de prueba
Le paso a lo que definimos en la precondicion, parámetros

```
objeto= {aa: "", bb: ""}
objeto= funcion(bla:param1, blabla:param2)
```

Resultado
Podemos ver que se cumplió con las condiciones

```
assert equals // veo si resultado coincide con lo esperado
assert all   // veo si cumple todos las condiciones
assert any  // veo si alguna ""
```

Es pseudocódigo. Como mínimo un assert.

Mensaje final
Si todas las verificaciones anteriores con assert son verdades, se imprime este mensaje, indicando que la prueba fue exitosa.

```
print("")
```

Excepciones
tenemos que mostrar la vinculación de las US con el ...