TDD Y DESARROLLO ÁGIL

OBJETIVOS

- Comprender conceptos generales sobre desarrollo ágil
- Aplicar la técnica de TDD para programas básicos

1

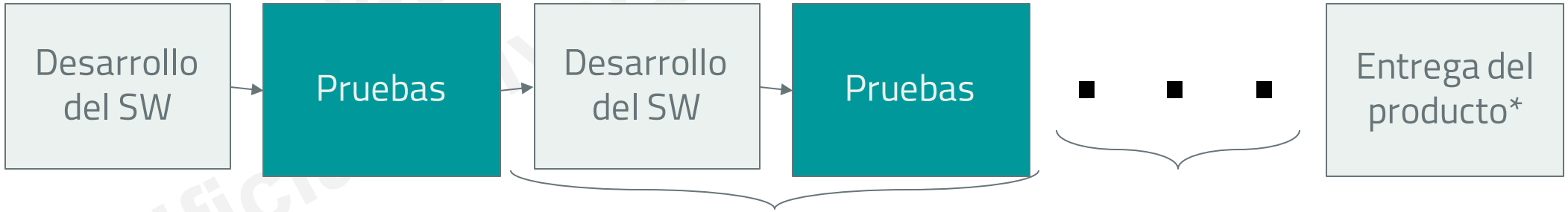
REPASO CICLO DE VIDA DEL SOFWARE

Conceptos de Ciclo de Vida del SW

Ciclo de Vida Cascada

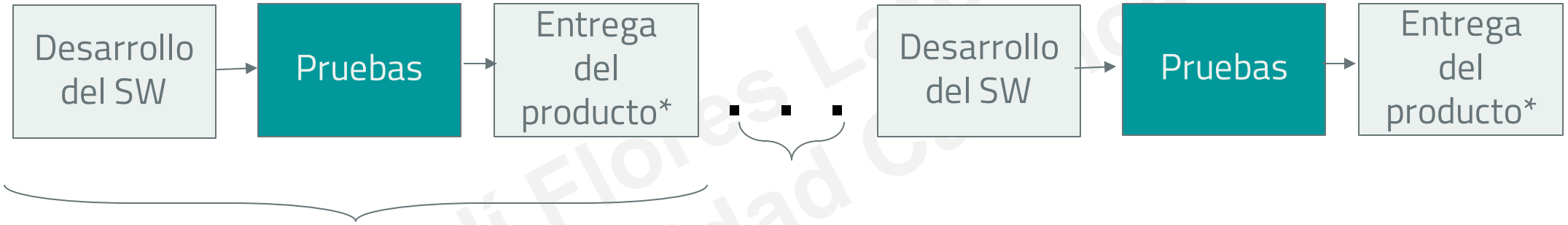
* Incluyendo pruebas de aceptación / validación

Ciclo de Vida Incremental

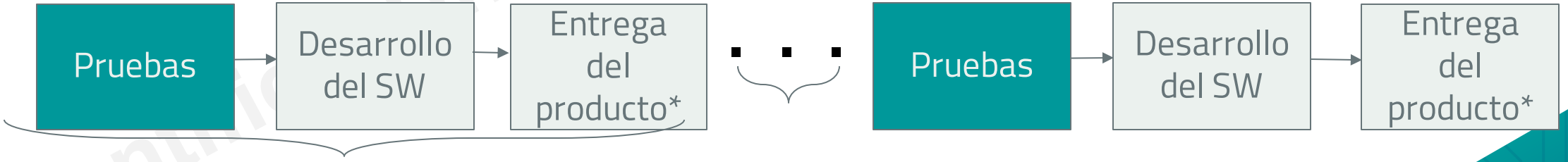


Conceptos de Ciclo de Vida del SW

Ciclo de Vida Ágil (Modo 1) / Evolutivo



Ciclo de Vida Ágil (Modo 2)



2

CONCEPTOS EN DESARROLLO ÁGIL

Algunos Conceptos Ágiles

- Pequeños Releases:
  - Se entrega un SW funcional después de cada Sprint.
- Pruebas de Aceptación:
  - Cada historia de usuario (requisito) debe tener una (o más) pruebas de aceptación.
- Propiedad Colectiva:
  - No existe dueño del código.
  - Todos pueden modificarlo.
- Espacio informativo:
  - Todos tienen acceso a la información del producto y proyecto
- Integración Continua:
  - Se compila/integra varias veces al día.

Algunos Conceptos Ágiles

- Desarrollo (Diseño) dirigido por Pruebas:
  - Técnica de diseño, funcionalidades requeridas.
  - Minimizar el número de errores.
  - Desarrollo modular y reutilizable.
  - TDD: TFD + Refactorización
  - TFD: Test-First Development

Algunos Conceptos Ágiles

- Refactorización:
  - Desarrollo iterativo -> diseño simple
  - Mejora Contínua del Diseño
  - Procesos se enfocan en:
    - Remover duplicaciones
    - Incrementar Cohesión (modularidad)
    - Disminuir acoplamiento

- Diseño Emergente:
  - El diseño surge de la revisión y refactorización del código.
  - No hay un gran diseño anticipado (aparte de la Arquitectura)

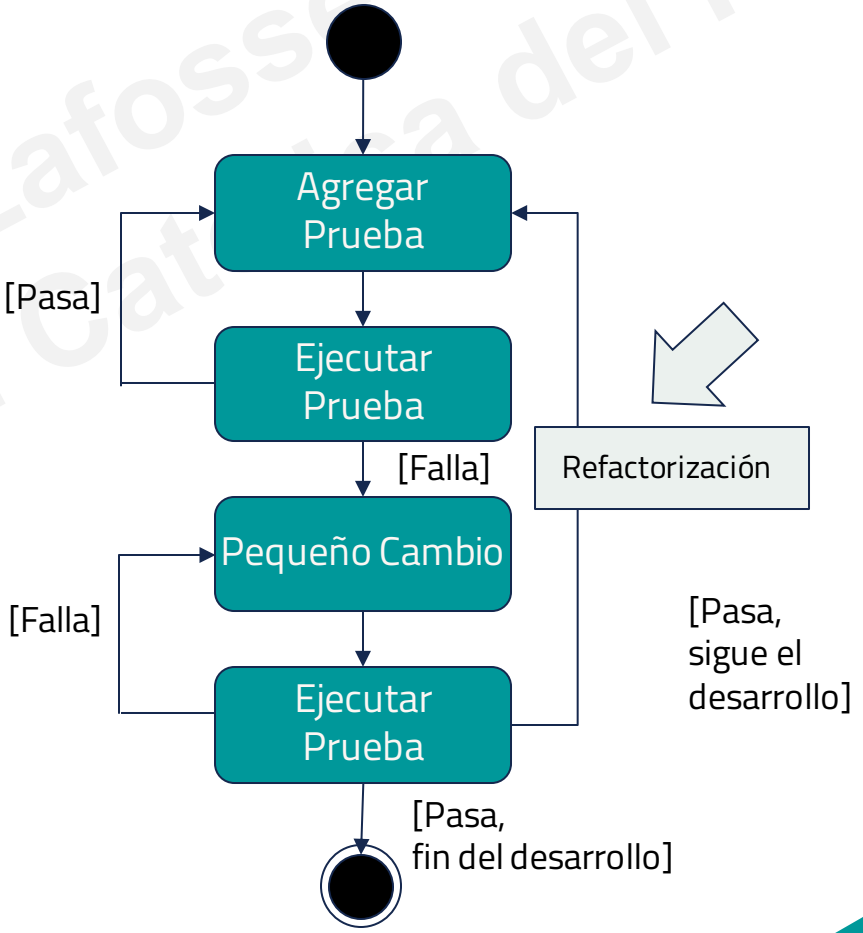
3

DESARROLLO DIRIGIDO POR PRUEBAS

Desarrollo dirigido por pruebas

Para aplicarlo, se sigue una serie de pasos:

1. Escribir test (según requisitos)
2. Ejecutar test (debe fallar!!)
3. Escribir **sólo lo necesario**
4. Verificar que pase el test
5. Refactorizar código
6. Repetir



Desarrollo dirigido por pruebas

- Diseño simple
- Escenario Nuevo → próxima prueba.
- Refactorizar => no afectar la interfaz.
- ¡No romper pruebas anteriores!

4

BENEFICIOS Y LIMITACIONES DEL TDD

Beneficios del T.D.D

- Mayor calidad del SW
- Mayor cobertura de código
- Código reutilizable
- Comunicación entre el equipo
- Tests como documentación
- Integración continua asegura regresión.
- Se evita trabajo innecesario.

Limitaciones y Problemática

- Alta dependencia entre tests y estructuras internas
- Nuevos requerimientos → tests inmantenibles.
- Abandono de los tests.
- El ahorro de tiempo → del diseño de tests
- Equipo con habilidades para crear pruebas
- La prueba es inservible si se interpreta mal el requerimiento
- Las pruebas se hacen largas y complejas. (Escalabilidad)

5

DESARROLLO DIRIGIDO POR COMPORTAMIENTOS (BDD)

A.T.D.D/B.D.D.

- B.D.D: Behavior driven development
- A.T.D.D: Acceptance T.D.D.
- Ambas son consideradas sinónimos.

El B.D.D. se centra en el comportamiento del sistema (más alto nivel)

A.T.D.D/B.D.D.

Se basa en escenarios completos, especificados por el patrón:

- **Given (Dado)**: Estado inicial
- **When (Cuándo)**: Acción/Evento
- **Then (Entonces)**: Cambios.

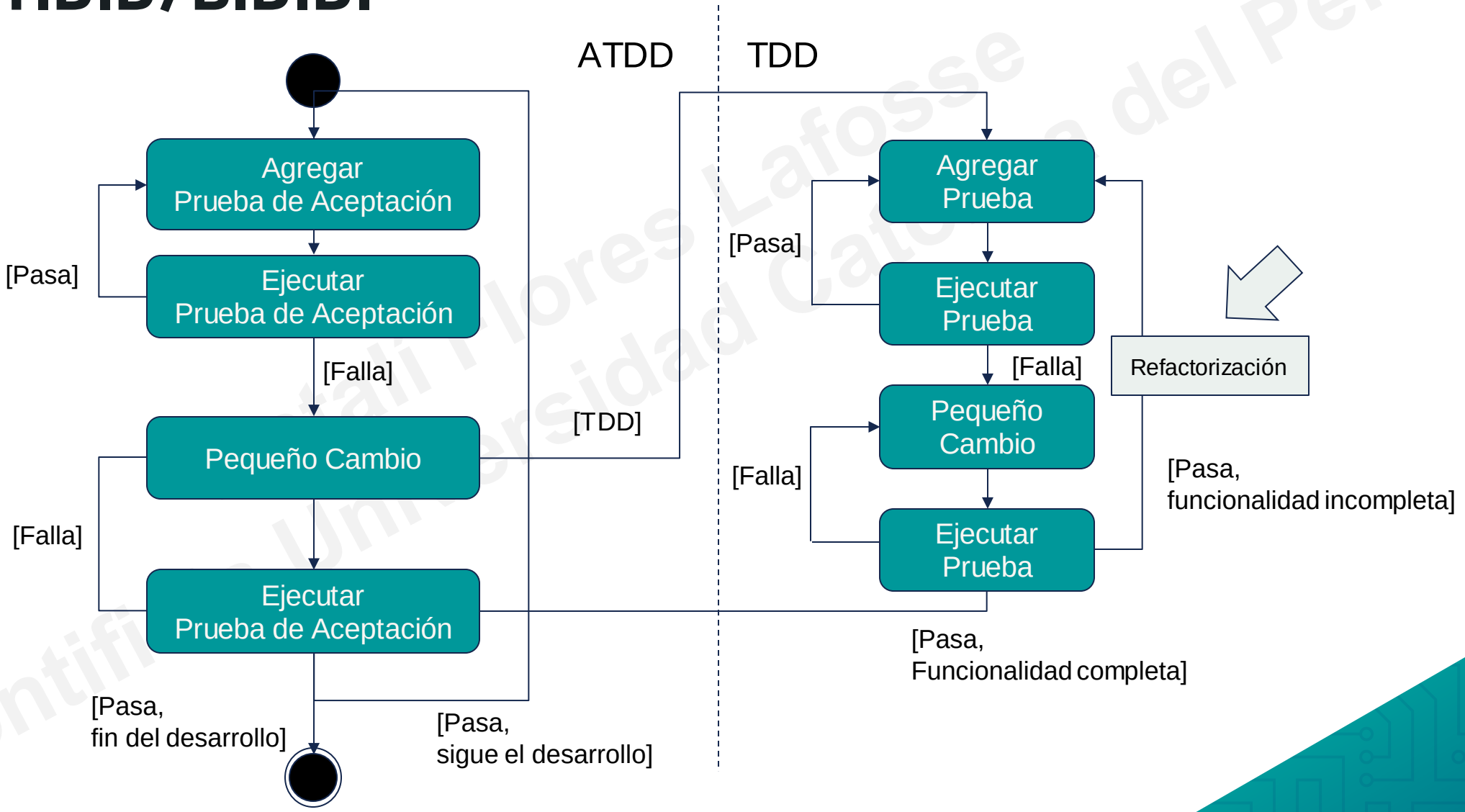
Cucumber:
Lenguaje Gherkin

Se relaciona con el patrón “Role-Feature-Reason” (Historias de Usuario)

- **As a (Como)**: Rol del usuario
- **I want (Quiero)**: Necesidad
- **So that (Entonces)**: Razones de la necesidad

Traducido y adaptado de Scott W. Ambler (2003-2008)

A.T.D.D/B.D.D.



T.D.D - Unit Testing – B.D.D.

Los tres términos se encuentran interrelacionados:

- Prueba Unitaria: Única funcionalidad o módulo
- TDD: Desarrollo a partir de las pruebas
- BDD: Desarrollo a partir de comportamiento

Las pruebas unitarias te dicen qué probar, TDD cuándo probarlo y BDD cómo probarlo.

“

Si vale la pena construirlo, vale la pena probarlo.

Si no vale la pena probarlo, ¿para qué perder el tiempo en trabajar en ello?”

http://agiledata.org/essays/tdd.html

6

EJEMPLO TDD

Ejemplo T.D.D

Usando TDD escribir una clase Calculadora que incluya como métodos de suma, resta, multiplicación y división de 2 números naturales hasta el 99.
Los errores deben regresar “999”.

Empecemos con:

```
1  public class Main
2  {
3      public static void main(String args[])
4      {
5          Main pruebas = new Main();
6          pruebas.correrPruebas();
7      }
8
9      public void correrPruebas()
10     {
11         Calculadora calc=new Calculadora();
12         System.out.println("Prueba 1:"+(calc.suma(2,3)==5));
13         System.out.println("Prueba 2:"+(calc.suma(2,1)==3));
14         System.out.println("Prueba 3:"+(calc.suma(101,1)==999));
15         System.out.println("Prueba 4:"+(calc.suma(2,234)==999));
16         System.out.println("Prueba 5:"+(calc.suma(-1,1)==999));
17         System.out.println("Prueba 6:"+(calc.suma(2,-23)==999));
```

7

PREGUNTAS

Aplicando

Preguntas

¿Cómo se diferencia un ciclo incremental de uno ágil?

A. Son iguales
B. Solo en el orden de las pruebas
C. Al terminar cada Sprint (en ágil), ya se tiene un elemento funcional, lo cual no se cumple en el incremental.

¿Se puede aplicar T.D.D en un sistema legacy?

Preguntas

¿Por qué se habla de un diseño emergente?

A. Porque el diseño detallado va surgiendo de las iteraciones de TDD.

B. Porque no se define una arquitectura.

C. Porque el diseño no es importante en TDD.

¿Cómo relacionamos las historias de usuario con el TDD?

A. No están relacionadas.

B. Cada historia debe tener criterios de aceptación, cada uno de los cuales genera una o más pruebas para TDD.

C. Las historias de usuario se usan para generar la arquitectura previa al TDD.

8

REFERENCIAS

BIBLIOGRAFÍA

- Kleer Agile Coaching & Training (2010) "Desarrollo Ágil de Software"
- Kniberg, H. (2007) "Scrum y XP desde las trincheras", C4Media

Créditos:

- Plantilla de la presentación por SlidesCarnival
- Diseño del fondo Hero Patterns