Y dijo Dios: Sea la luz; y fue la luz.

Y vio Dios que la luz era buena; y separó Dios la luz de las tinieblas.

Y llamó Dios a la luz Día, y a las tinieblas llamó Noche.

```
@test

$light = new Light(100, ‘Day’);
$darkness = new Light(0, ‘Night’);

$light->start();
$darkness->start();

$this->assertTrue($light->isGood());
$this->assertFalse($darkness->isGood());
$this->assertNotEqual($light, $darkness);
```

[√] Testing

Del Ojímetro™ al TDD

Problema

¿Cómo asegurarme de que mi código hace lo que se espera que haga?

¿Cómo asegurarme de que mi código se ajusta a las especificaciones?

Ojímetro™ based testing

Ojímetro™

No es reproducible.

No es automatizable.

No separa los efectos colaterales de los deseados.

No es objetivo.

No tengo ni idea de qué casos cubre.

Ojímetro	Testing
Subjetivo	Objetivo
No reproducible	Reproducible
No automatizable	Automatizable
Efectos colaterales	Aislamiento unidades
No sabemos qué mide	Sabemos qué mide

Testing

Es control de calidad del software.

Los tests son pruebas objetivas, expresadas formalmente, bien definidas, automatizables y reproducibles.

Es aplicar el método científico.

Tests

Fundamentalmente son programas muy sencillos que nos dicen si las unidades de software hacen aquello que esperamos.

Unidad de software: *una función, una clase, un método de una clase…*

Tests

Comunican lo que hace el software

Explican cómo usarlo
(documentación viva)

Nos permiten discutir sobre ello.

Tests

Los tests son sencillos de escribir y deben ser rápidos de ejecutar.

Si es complicado hacer un test nos está revelando un fallo en el diseño.

Tests & Debug

Los test no evitan los errores.

Pero sí nos pueden decir si los cambios en el código producen errores (regresión).

Tipos

Funcionales	No funcionales
Qué **hace el sistema**	**Cómo** **lo hace**
Mide el cumplimiento de especificaciones o requisitos	**Rendimiento, velocidad, escalabilidad, recuperación…**

Tests funcionales

Unitarios

Integración

Caracterización

Aceptación

Regresión

Tests funcionales

Unitarios

Integración

Caracterización

Aceptación

Regresión

Prueban las unidades de software en aislamiento.

Tests funcionales

Unitarios

Integración

Caracterización

Aceptación

Regresión

Prueban las unidades de software en interacción.

Tests funcionales

Unitarios

Integración

Caracterización

Aceptación

Regresión

Nos sirven para descubrir/ describir el comportamiento de un software existence (Legacy).

Es una red de seguridad para empezar a refactorizar.

Legacy es código sin tests

Michael Feathers

Tests funcionales

- Unitarios
- Integración
- Caracterización
- **Aceptación**
- Regresión

Responden a la pregunta de si la funcionalidad está implementada en términos de quienes están interesados en el software (*stakeholders*).

Herramientas específicas como Behat y lenguaje Gherkin.

Tests funcionales

- Unitarios
- Integración
- Caracterización
- Aceptación
- **Regresión**

Detectan consecuencias (malas) cuando hacemos cambios en el código.

Anatomía de un test

Dado…

Cuando…

Entonces…

Anatomía de un test

Dado…

Define el escenario.

Cuando…

Las condiciones en las que vamos a probar una funcionalidad.

Entonces…

Anatomía de un test

Dado...

Ejecutar la unidad de software en la que estamos interesados.

Cuando…

Entonces…

Anatomía de un test

Dado...

Cuando…

Entonces…

Aserciones: Comparar los resultados con los que esperamos obtener.

Problema

Control de los efectos colaterales

Cómo testear una unidad de software en aislamiento si tiene dependencias

```
class Service(
      Repository $repository,
      Mapper $mapper) {
…
}
```

Test doubles

Dummy	Stub	Spy	Mock	Fake

Test doubles

Dummy

No hace nada, pero necesitamos pasar la dependencia.

Fáciles de crear a partir de una interface.

Stub

Spy

Mock

Fake

Test doubles

Stub

Simula la funcionalidad de otro módulo de software, dándonos un resultado que nos interesa.

La funcionalidad consume recursos o podría no estar disponible. Ej: Mailer, DB, API...

Dummy

Spy

Mock

Fake

Test doubles

Spy

Similar a un Stub, recoge información sobre cómo ha sido usada.

¡Test frágiles, acoplados a la implementación!

Dummy

Stub

Mock

Fake

Test doubles

Mock

Tiene expectativas, es decir, espera ser llamado de cierta forma o frecuencia.

¡Test frágiles, acoplados a la implementación!

Dummy

Stub

Spy

Fake

Test doubles

Fake

Es una implementación ad hoc de una dependencia para el testing. Ej: InMemory*

¡Puede necesitar sus propios tests!

Dummy

Stub

Spy

Mock

Test frágil

Un test que puede romperse por razones por las que no debería hacerlo.

P.e.: acoplamiento a la implementación: un cambio funcionalmente irrelevante como cambiar una llamada o su orden lo puede romper.

¿Qué testear y cuánto testing es necesario?

Depende...

Cuanto testing

Complejidad ciclomática de la unidad de software: evaluar todos los caminos.

Casos límite: atención a las desigualdades estrictas (>= | >).

Code coverage: nos ayuda a saber qué estamos y qué no estamos testeando.

No hay que obsesionarse con esa medida.

No tests

Métodos privados/protegidos

Lo que ya tenga sus propios tests

Me pagan por hacer código que funcione, no por hacer tests

Ken Beck

Test no prioritarios

Código trivial (accessors, controllers)

Código dependiente de librerías bien cubiertas por tests (testear interacción)

Si test se hace muy difícil podría estar revelando fallos de diseño.

TDD

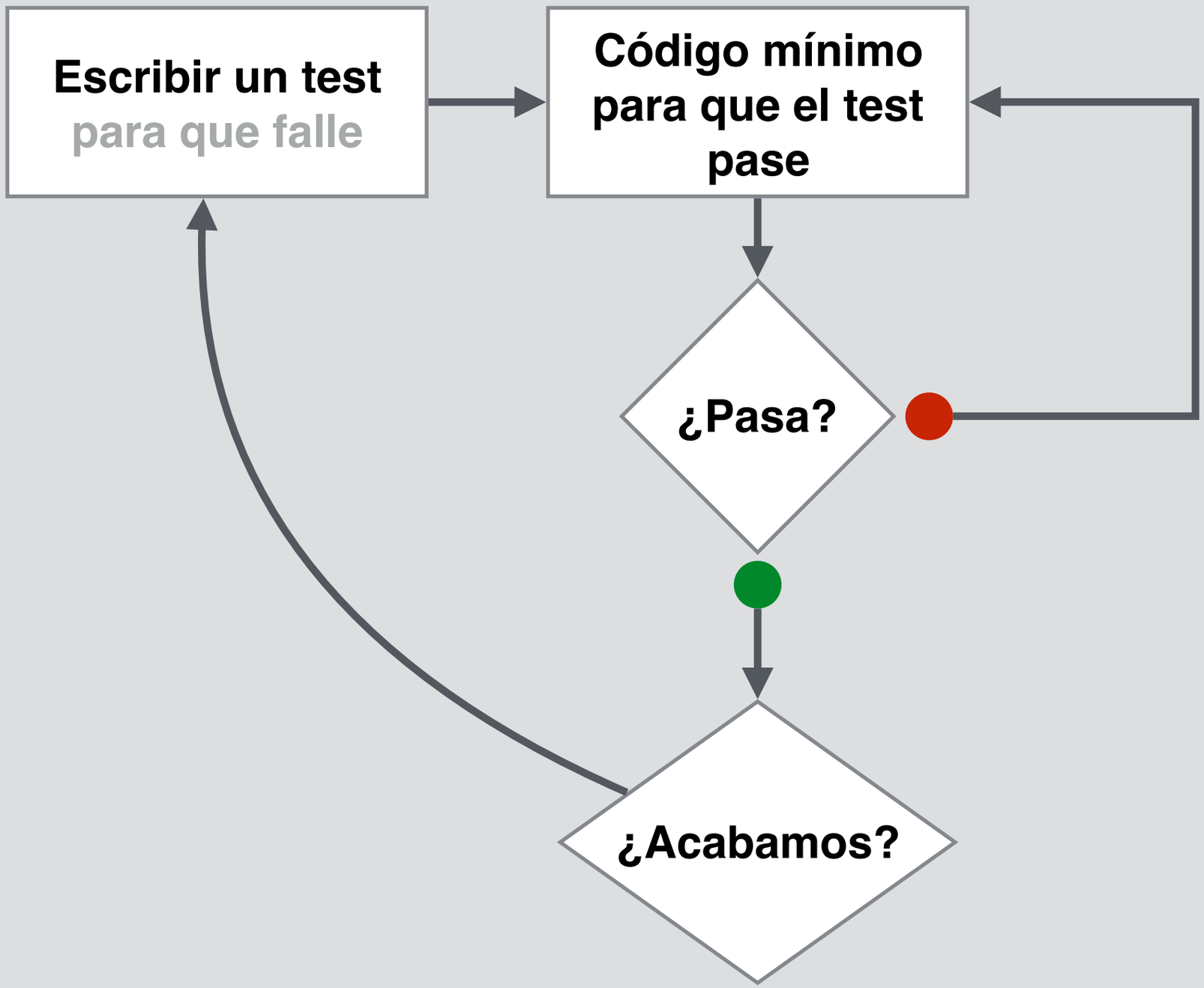
TDD

Test Driven Development.

Escribir los tests antes que cualquier código.

Se basa en un bucle de cambios mínimos.

TDD


Escribir un test
para que falle
Código mínimo
para que el test
pase
¿Pasa?
¿Acabamos?

Leyes de TDD

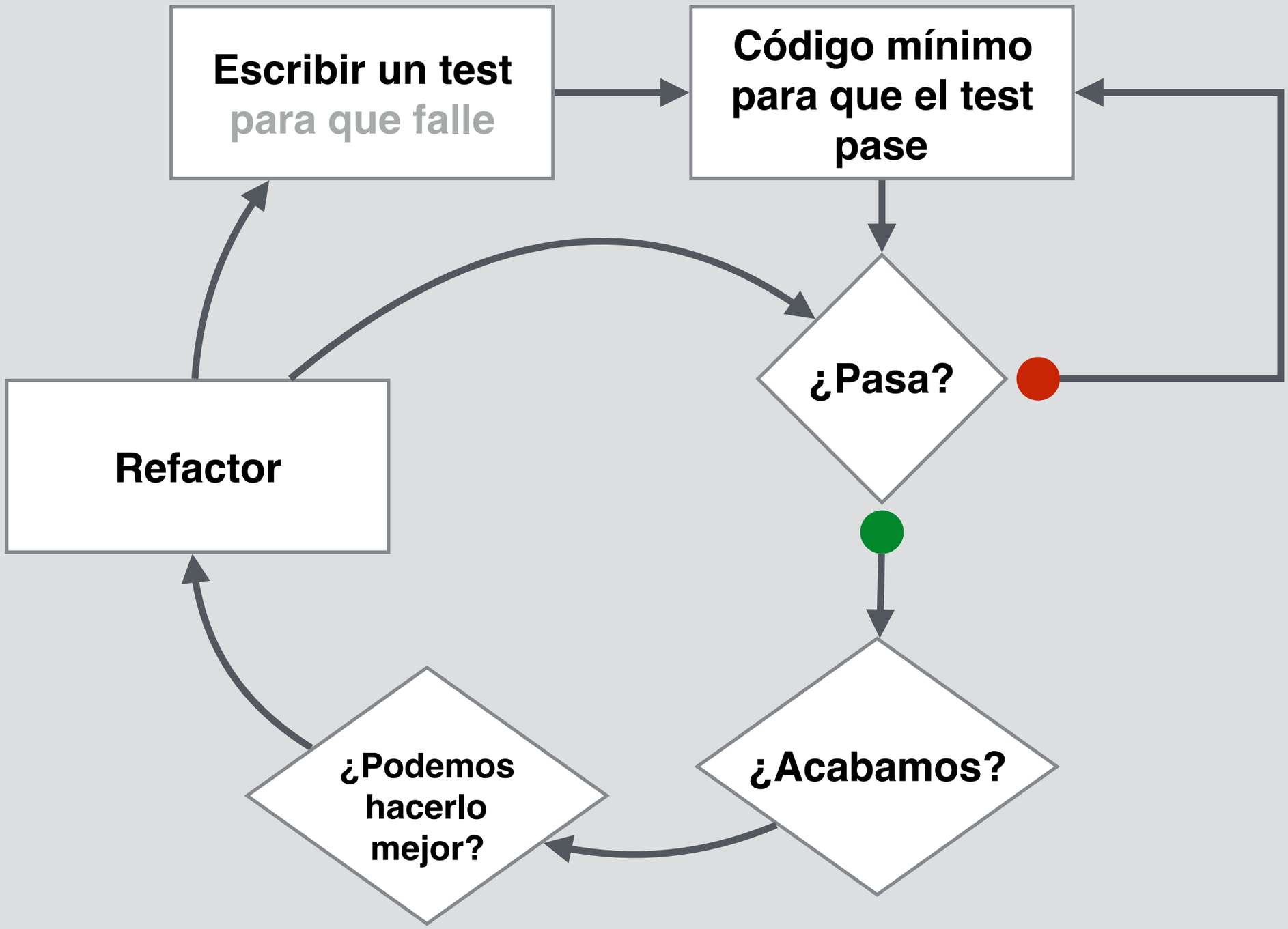
No escribirás código de producción sin antes escribir un test que falle

No escribirás más de un test unitario suficiente para fallar (incluye class not found, etc.)

No escribirás más código del necesario para hacer pasar el test.

Robert C. Martin

TDD & Refactoring



Beneficios TDD

El software está constantemente respaldado por tests.

En todo momento tenemos tests de regresión.

La necesidad de “testabilidad” nos empuja a mejores prácticas (principios SOLID)

Herramientas

PHPUnit

De la familia *Unit

La herramienta de test estándar

Todo tipo de tests, especialmente unitarios.

Total libertad (=> gran responsabilidad)

PHPSpec

TDD/BDD en base a ejemplos

Generadores de código y restricciones

Ideal para modelar, no para post-test

Tiene sus propias opiniones sobre cómo escribir y subirá la calidad de tu código

Behat

BDD basado en User Stories o Features, orientado a la comunicación con los expertos de dominio

Genera tests de aceptación a partir de la descripción de Features en lenguaje quasi-natural (Gherkin)

Port de Cucumber

Codeception

Combina tests de aceptación, integración y unitarios.

Se basa en PHPUnit.