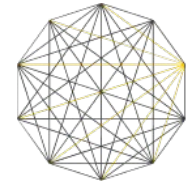


Mejorando IV - Testing



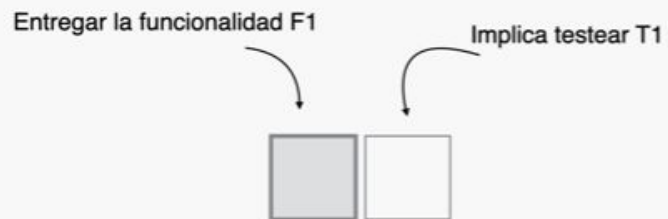
540
quinientoscuarenta



Contexto

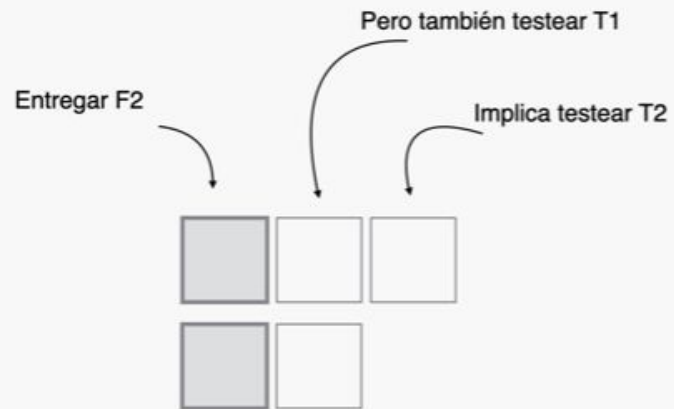


Contexto



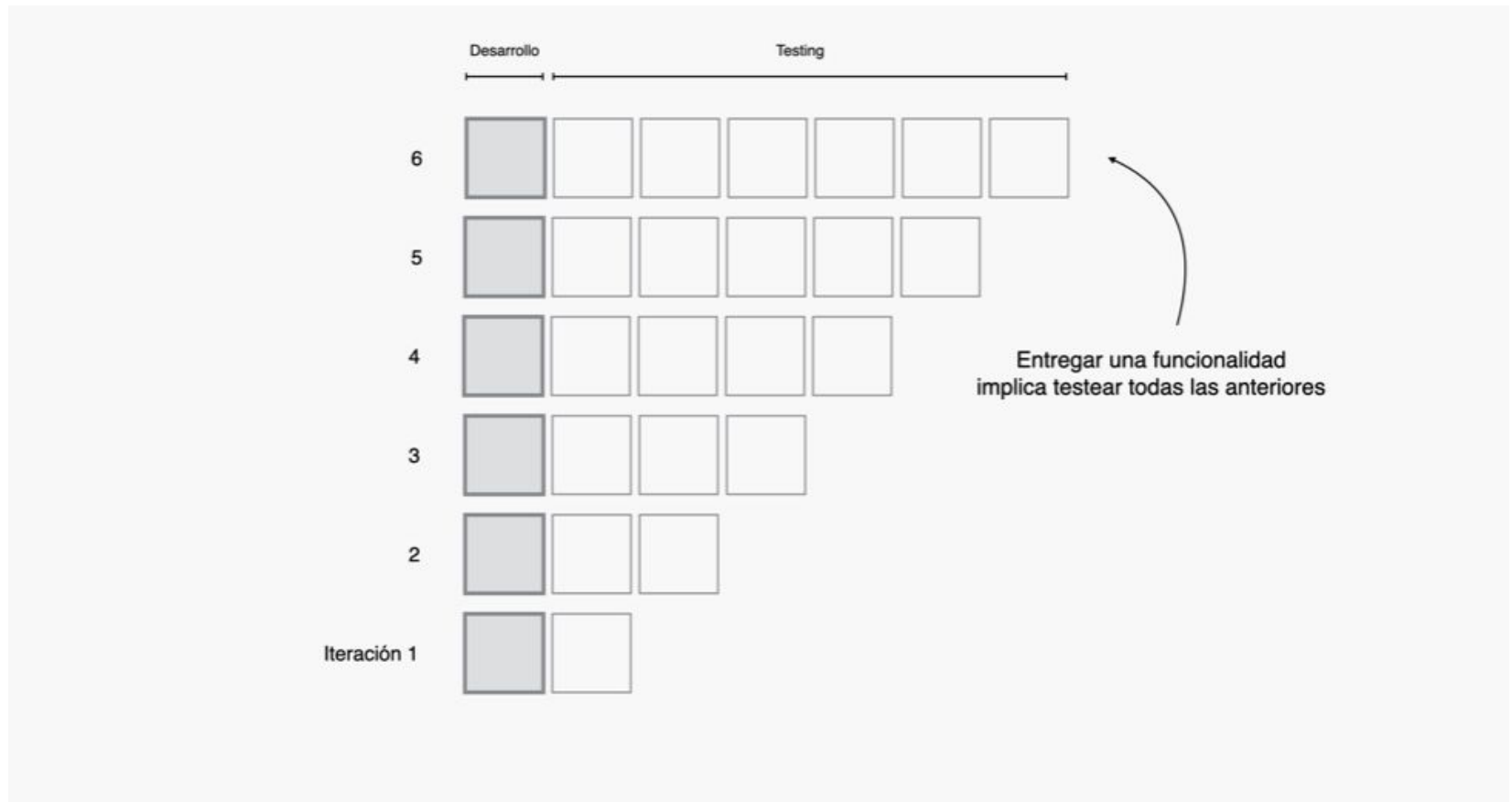
Fuente: "Software Economics", Luis Artola

Contexto



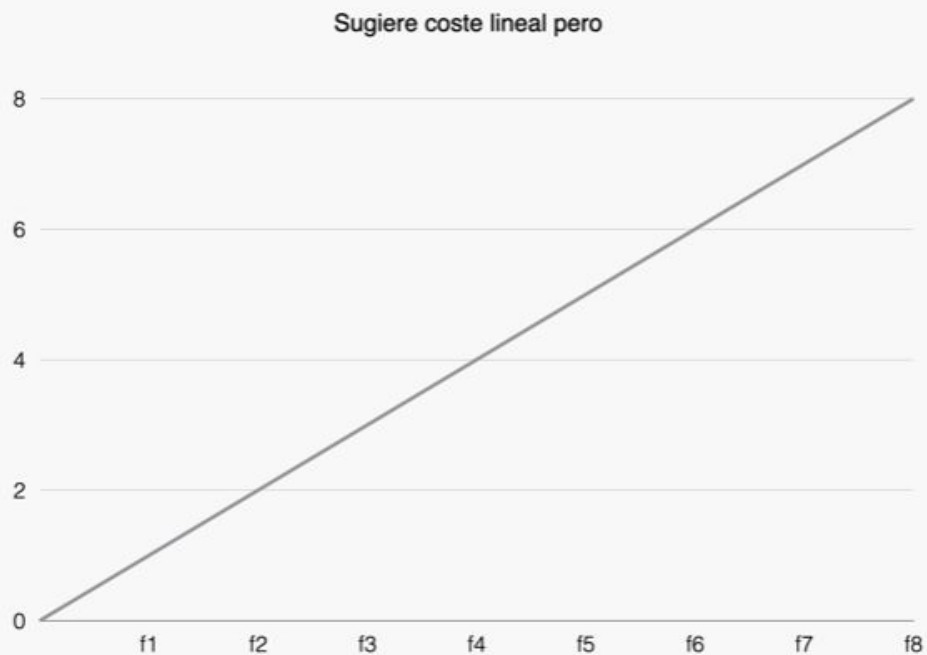
Fuente: "Software Economics", Luis Artola

Contexto



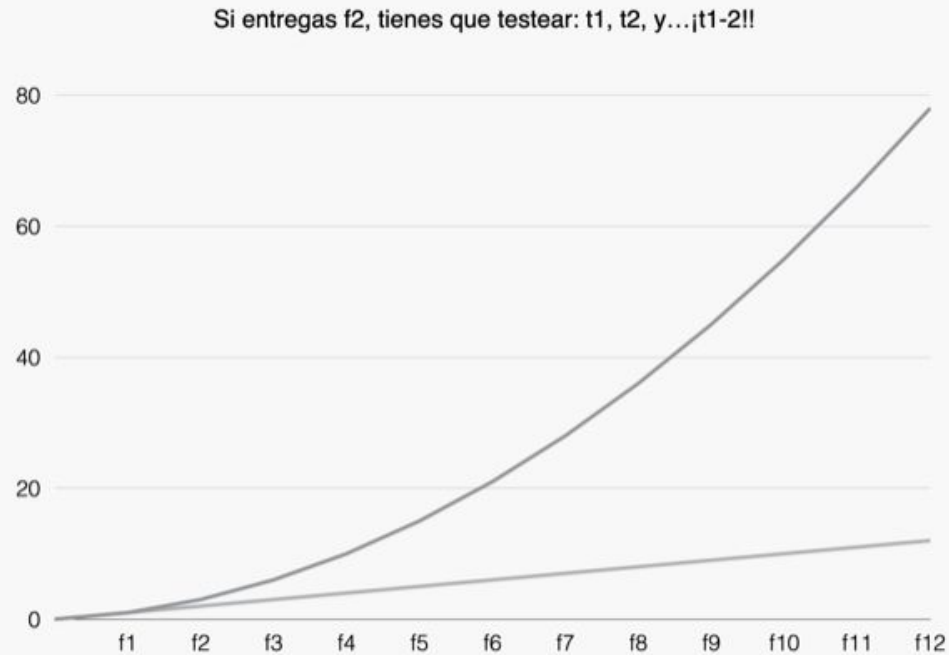
Fuente: "Software Economics", Luis Artola

Contexto



Fuente: "Software Economics", Luis Artola

Contexto



Fuente: "Software Economics", Luis Artola

El ideal

- Modificar nuestro código con seguridad de no romper nada
- Añadir funcionalidad asegurándonos que funciona
- Evitar pruebas manuales, automatizar
- Aportar valor a negocio de forma continua y con un ritmo sostenible
- Tener una documentación viva y ejecutable



Tipos de tests



Tipos de tests

- Unitarios.
- Integración.
- Aceptación o extremo a extremo (end to end, E2E)
- Regresión visual / snapshot

Test unitarios

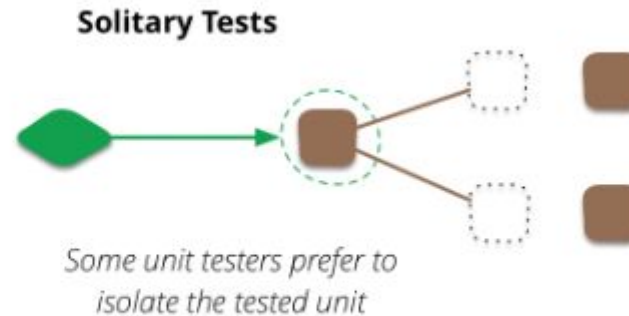
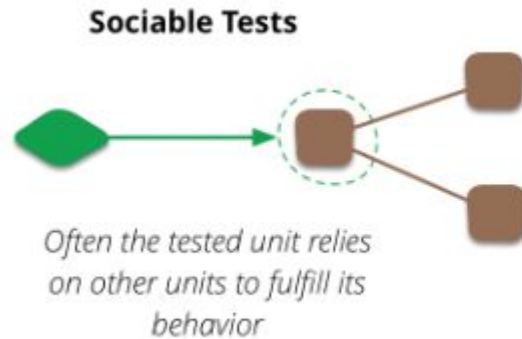
“Una unidad es cualquier cosa que un test de unidad puede probar fácilmente”.

Michael Feathers

Test unitarios

- Un test unitario asegura de que una unidad (el SUT) de la base de código funciona según lo previsto.
- Una unidad es la parte testable más pequeña del software.
- No es un test de una clase.
- Muy rápidos.
- Muy limitados en su alcance. Si fallan, es obvio dónde está el problema.
- Puede ser “sociable” o “solitario”.

Test unitarios “sociables” o “solitarios”



<https://martinfowler.com/bliki/UnitTest.html>

Test de integración

“Los tests de integración prueban si las unidades de software desarrolladas de forma independiente funcionan correctamente cuando están conectadas entre sí.”

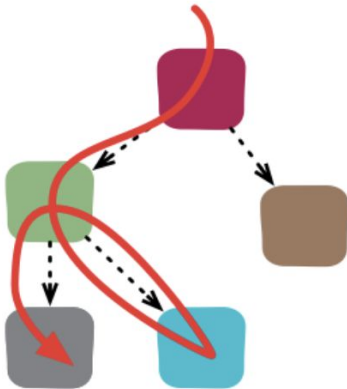
Martin Fowler

Test de integración

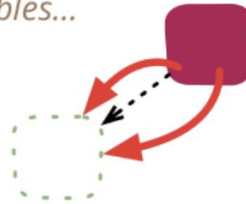


Test de integración

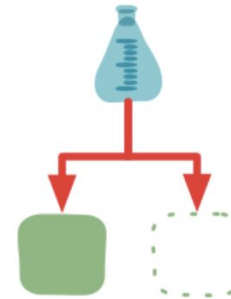
Integration Testing commonly refers to broad tests done with many modules active...



...but it can be done with narrow tests of interactions with individual Test Doubles...



...supported by Contract Tests to ensure the faithfulness of the double



<https://martinfowler.com/bliki/IntegrationTest.html>

Test unitarios vs Test de integración

- Diseño.
- Fragilidad
- Bugs.
- Mantenibilidad.
- Documentación.

Tests E2E

- Prueban el comportamiento de principio a fin.
- Interactúan con la “vista” para realizar los tests.
- Lentos.
- Frágiles.
- Costosos de escribir y mantener.

Tests E2E

Attività Google Chrome ven 24 nov, 11.52

lara-kb x

127.0.0.1:8000/_/#/tests/integration/app_spec.js

Tests 3 10.13 http://127.0.0.1:8000/home

Run All Tests

Knowledge Base Application

- ✓ Shows a placeholder when there are no articles
- ✓ Should be able to login: admin
- ✓ Should be able to create a new article: admin

SERVERMANAGED Learning Center

Create a new article

Title

SEO Title

Slug

SEO Meta Description

Body (0 words so far)

Post Status

Published

Save

Articles

Title	Published
-------	-----------

Tests E2E

- Prueban el comportamiento de principio a fin.
- Interactúan con la “vista” para realizar los tests.
- Lentos.
- Frágiles.
- Costosos de escribir y mantener.

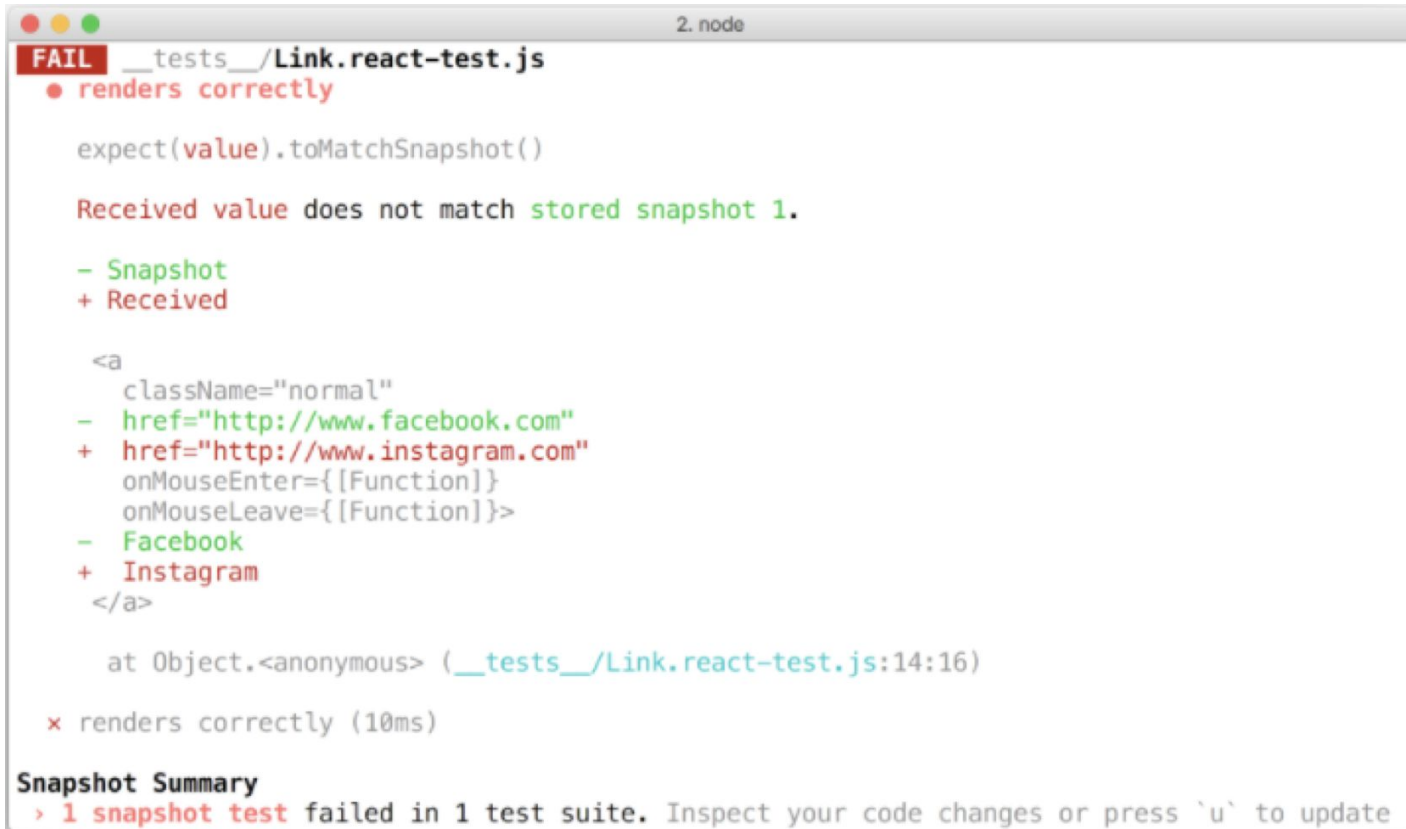
Snapshot

- Sacar una “foto” del código y a partir de ahí se compara el código con la “foto” guardada.
- Sirven para asegurar que una funcionalidad sigue funcionando como lo hacía cuando la foto se sacó.
- Se serializan valores, se guardan en archivos de texto y se comparan con un algoritmo.

Snapshot

- Tests de interfaz.
- Sirven para asegurar que la interfaz no cambia de forma inesperada.
- Sacar una “foto” del código que se pinta y a partir de ahí compara el código con la “foto” guardada.

Snapshot



```
2. node
FAIL __tests__/Link.react-test.js
  ● renders correctly

    expect(value).toMatchSnapshot()

    Received value does not match stored snapshot 1.

    - Snapshot
    + Received

    <a
      className="normal"
    - href="http://www.facebook.com"
    + href="http://www.instagram.com"
      onMouseEnter={ [Function] }
      onMouseLeave={ [Function] }>
    - Facebook
    + Instagram
    </a>

    at Object.<anonymous> (__tests__/Link.react-test.js:14:16)


    x renders correctly (10ms)

Snapshot Summary
  > 1 snapshot test failed in 1 test suite. Inspect your code changes or press `u` to update
```

Screenshot


- Tipo de test que captura una imagen de una url o de un estado concreto de la aplicación y la compara con otra imagen de la misma url o mismo estado en un momento diferente.
- El uso más común del Screenshot testing es el de comprobar si la Interfaz de Usuario (UI) no se ha roto, o para comprobar si la web se renderiza correctamente entre diferentes navegadores.

[Gmail](#) [Images](#) [Sign in](#)



[Google Search](#) [I'm Feeling Lucky](#)

Google offered in: [polski](#)

 A privacy reminder from Google


[REVIEW NOW](#)

Poland

[Advertising](#) [Business](#) [About](#)

[Privacy](#) [Terms](#) [Settings](#)

[Gmail](#) [Images](#) [Sign in](#)



[Google Search](#) [I'm Feeling Lucky](#)


Google offered in: [polski](#)

Poland

[Advertising](#) [Business](#) [About](#)


[Privacy](#) [Terms](#) [Settings](#)

[Gmail](#) [Images](#) [Sign in](#)



[Google Search](#) [I'm Feeling Lucky](#)

Google offered in: [polski](#)

 A privacy reminder from Google

[REVIEW NOW](#)

Poland

[Advertising](#) [Business](#) [About](#)

[Privacy](#) [Terms](#) [Settings](#)

Otros tipos de test

- Funcionales o aceptación -> comprueban que las reglas de negocio se cumplen.
- Property based -> test para probar múltiples casuísticas.

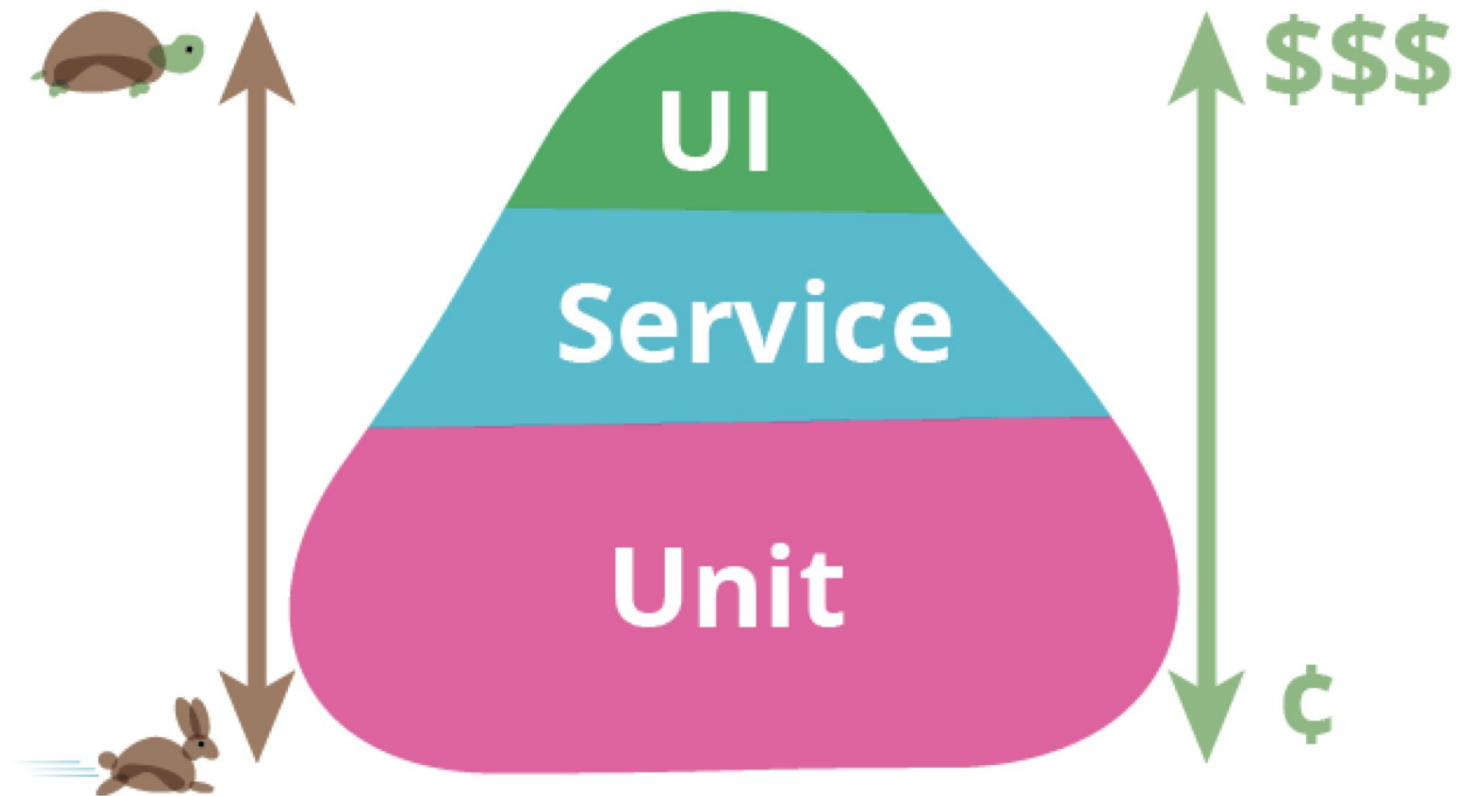
Otros conceptos

- Tests parametrizados.
- Mutation testing.
- Cobertura de código.

Pirámide de tests



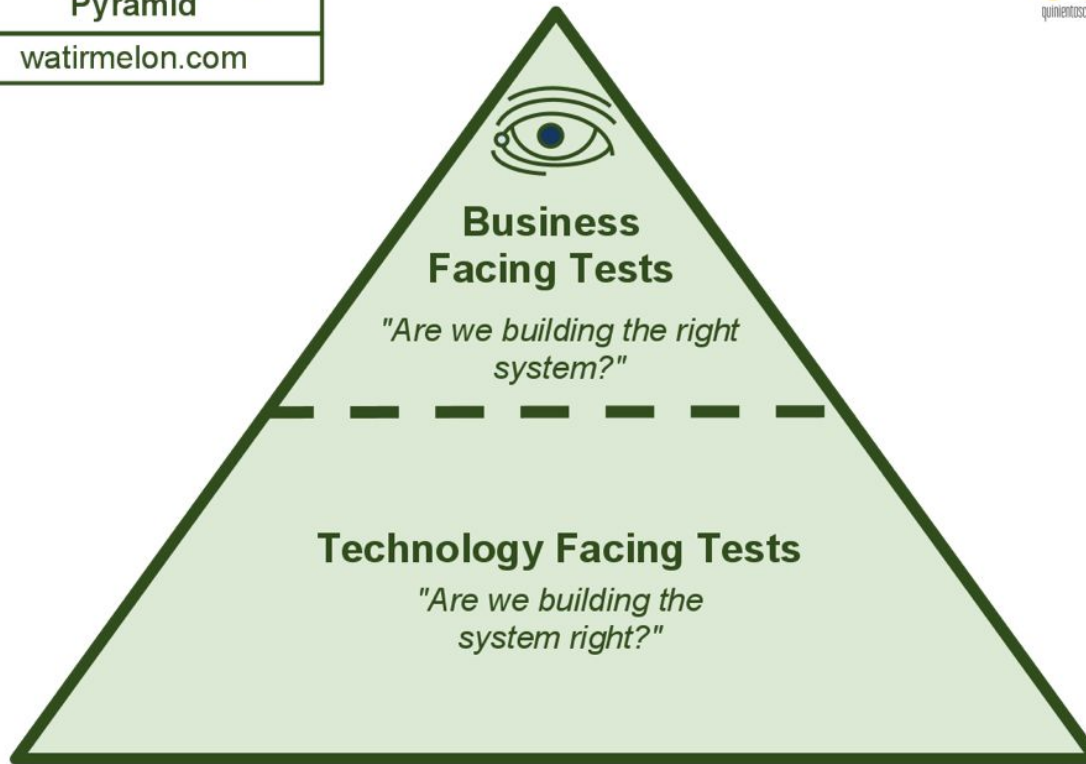
Pirâmide de tests



Pirámide de tests

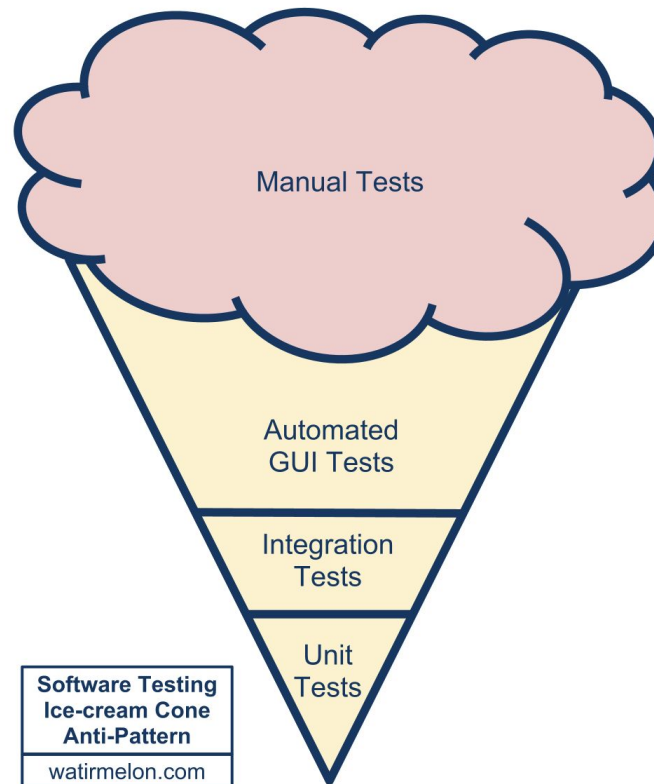


540
quintoscuenta



540deg.com

Ice-cream anti-pattern





Otros conceptos



Conceptos sobre testing

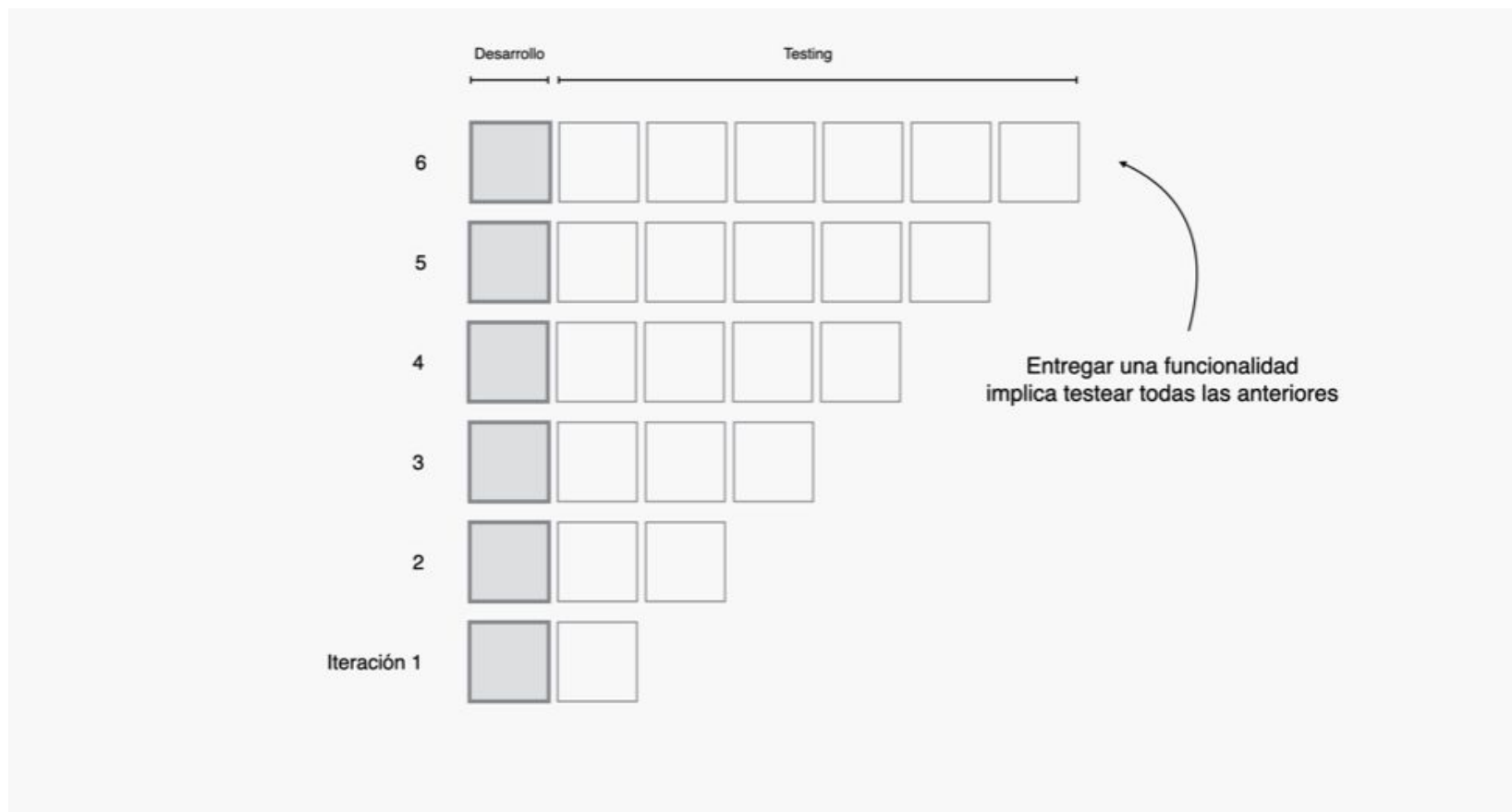
- Cobertura de tests
- Framework de tests
- Integración con IDE



Sosteniendo y mejorando nuestro código

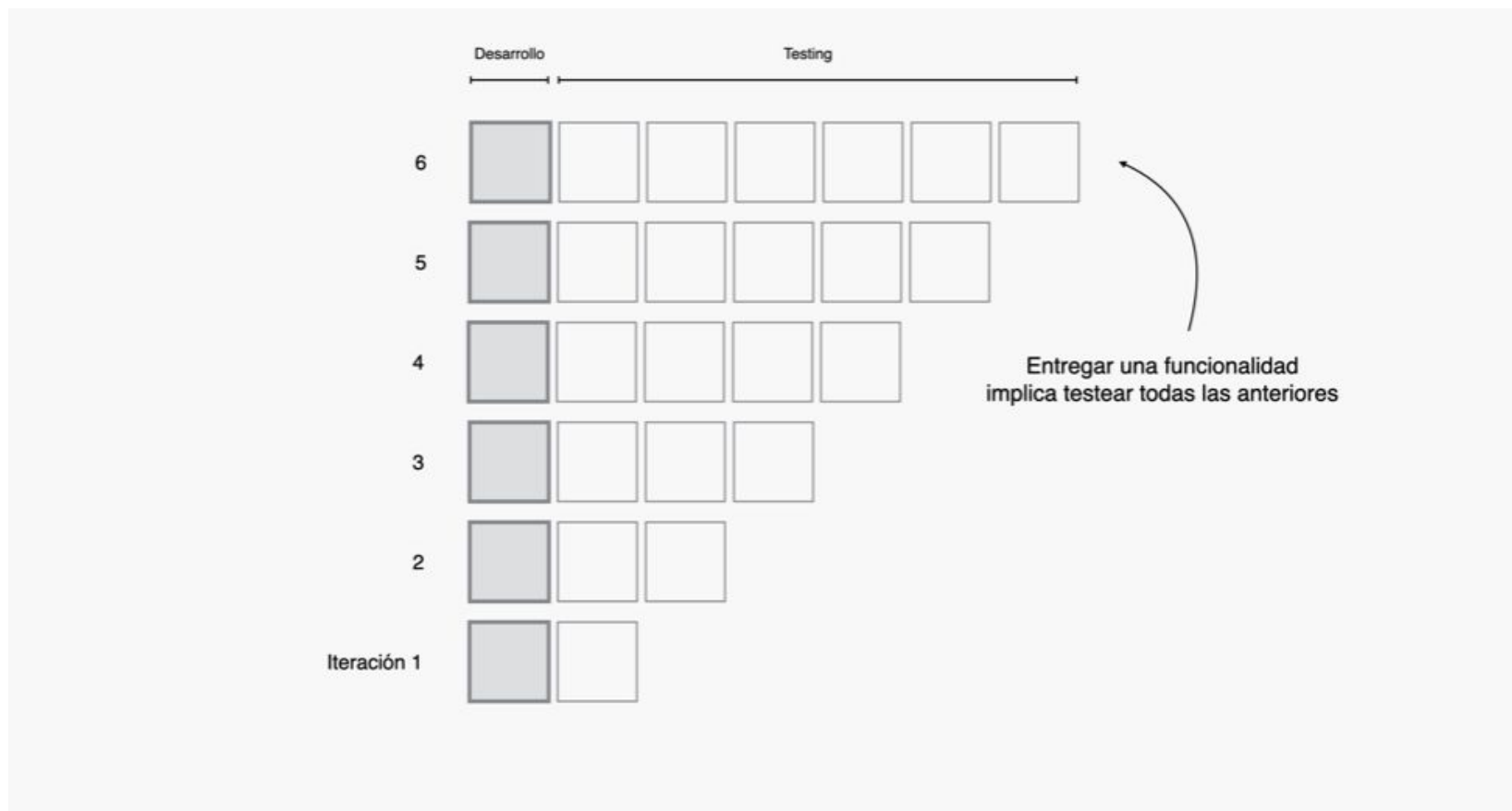


Contexto



Fuente: "Software Economics", Luis Artola

Contexto



Fuente: "Software Economics", Luis Artola

Contexto

- Tenemos gran parte de solución implementada
- El código es difícil de modificar
- Difícil asegurar que no rompemos nada
- Queremos seguir construyendo una solución que sea sostenible en el tiempo



Test unitarios



Tests unitarios

- Son un tipo de tests
- SUT = unidad
- Una unidad es la parte testable más pequeña del software
- Se tiene que ejecutar rápido para poder ejecutarse muchas veces...sino no se ejecuta
- Muy limitado en su alcance. Si falla, es obvio dónde buscar el problema
- Prueban un único comportamiento

Tests mantenibles

- Test sucios -> No test -> !Flexibilidad código de producción
- Los tests tienen que cambiar a medida que lo hace el código de producción
- El código de los tests es tan importante como el de producción

Legibilidad de los tests

- Igual de importante que en código de producción
- Claridad, simplicidad y densidad de la expresión
- Eliminar detalles innecesarios
- Utilizar un lenguaje de dominio para los tests

```
public void IsDeliveryValid_InvalidDate_ReturnsFalse()  
  
public void Delivery_with_invalid_date_should_be_considered_invalid()
```

Estructura

- Preparación / Ejecución / Aserción
- Arrange / Act / Assert
- Given / When / Then

FIRST

- Fast (rápido)
- Independent (independiente)
- Repeatable (repetible)
- Self-Validating (auto-validado)
- Timely (oportuno)

Puntos clave

- En cada test prueba UNA sola cosa que funciona
- Testear el qué, no el cómo
- Testear la interfaz pública de cada unidad. No testear métodos privados
- No son gratis, son tan importantes como el código de producción, hay que mantenerlos, deben ser fáciles de entender, comprensibles y legibles y sólo deben fallar cuando aquello que quieres probar no funciona

Puntos clave

- No testear código trivial, que no tiene lógica (p.ej. getters, setters)
- Utilizar nombres descriptivos, basados en comportamientos y relacionados con tu dominio
- Eliminar detalles innecesarios
- Hacer tests no significa no tener bugs ni tener buen código
- No escribir tests por buscar el 100% de cobertura, hacer test útiles que ayuden a comprobar que el sistema funciona y den fiabilidad a la hora de refactorizar.
- Los tests son una buena documentación



TDD



¿Qué es?

Test-Driven Development es una técnica de desarrollo software guiada por tests

¿Qué es?

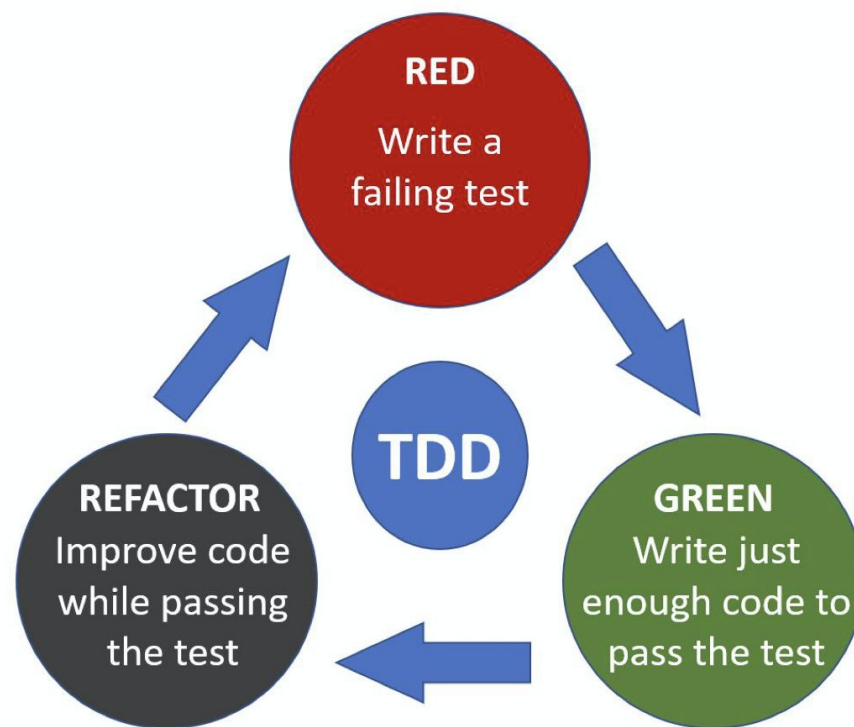
“TDD es una inversión que debe llevarte a menos riesgo, menos deuda, y más facilidad a la hora de añadir funcionalidad.”

Fuente: “Software Economics”, Luis Artola

El ciclo

1. Escribir un test que falla
2. Escribir la implementación más sencilla que haga que el test pase
3. Refactorizar código (de producción y de test)

El ciclo



Características

- Hace el refactoring económicamente viable
- Produce tests automáticos, pero no es una técnica de testing
- Ayuda a generar documentación básica de nuestro código
- TDD no es una técnica que aporte valor a negocio de forma directa
- Respecto al diseño...

Respecto al diseño

- Chequea el diseño para responder de forma rápida y barata al cambio
- Permite posponer decisiones de diseño e implementación
 - “Make it pass, make it right, make it fast”
- Ayuda a aflorar problemas en el diseño, no los soluciona

“When you write tests, you´re writing a program to understand your program. If your program is difficult to understand, your tests will be difficult to write. So if a test is difficult to write, it is telling us the design of our code should be improved.”

Michael Feathers”

Refactoring y Rule of three

- No se entiende TDD sin refactoring.
- En la fase de refactoring, nos fijamos en la duplicidad siguiendo la “Rule of Three”.
- Rule of Three: extraemos la duplicidad cuando la vemos tres veces.
- Mejor código duplicado que una mala abstracción.

Baby steps

- El feedback temprano no viene de la velocidad, viene de dar pasos pequeños
- Cada test representa un vertical funcional que nos lleva a cumplir las reglas de negocio
- Reducimos el riesgo
- Conseguimos pequeñas victorias
- Buscamos el aporte de valor continuo e incremental

La realidad de TDD

- TDD no testea ni diseña por nosotros
- Es muy difícil de aplicar en código que no haya sido construido sobre “buenas prácticas” -> Requiere de un proceso, hay que hacer también una inversión
- Hay que saber cuándo aplicarlo
- Requiere de experiencia y conocimiento
- Requiere una cultura y alineamiento de equipo y de negocio

Situaciones reales

- Montaje costoso
- TDD no asegura nada
- Los tests hay que mantenerlos

¿Cómo aporta valor?

- Feedback temprano y continuo
- Fuerza pensar antes de programar
- TDD aporta reducción de costes mediante la mejora del diseño
- Ayuda a escribir tests basados en reglas de negocio que permiten entregar valor temprano



Kata FizzBuzz



FizzBuzz

- Definición ejercicio:
<https://github.com/540/Katas/blob/master/FizzBuzz.md>
- <https://github.com/540/docker-php-boilerplate>
- <https://github.com/LeWricka/php8-testing-boilerplate>



<https://540deg.com/>

@540deg