

---

# Introducción al testing y TDD

— Adalab, 4 de septiembre de 2017 —  
Isidro López (@islomar)

---

# Ego slide



@islomar



islomar@gmail.com

Desarrollador de software



alea soluciones

# ¿Para qué estamos aquí?

- Para divertirnos, aprender y compartir
- Qué son los **tests automatizados**
- Qué es **TDD** (Test-Driven Development)
- Qué es una **kata**
- Trabajar en **parejas** y con pomodoros



# Disclaimer

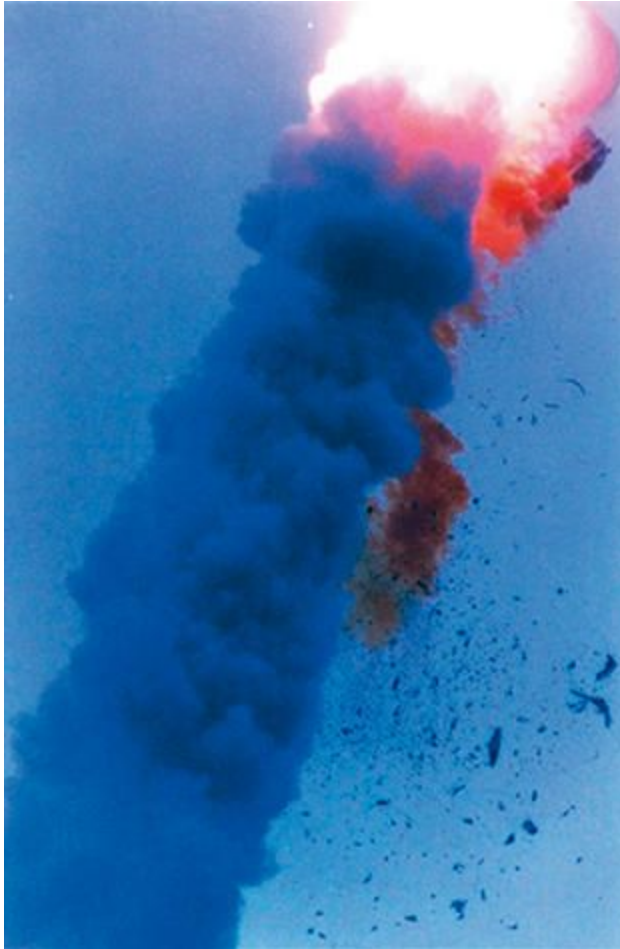


# Disclaimer

**Shu-Ha-Ri:** si me “creéis”, hacedlo poco... y sólo de momento

**Probad y verificad** siempre todo por vuestra cuenta

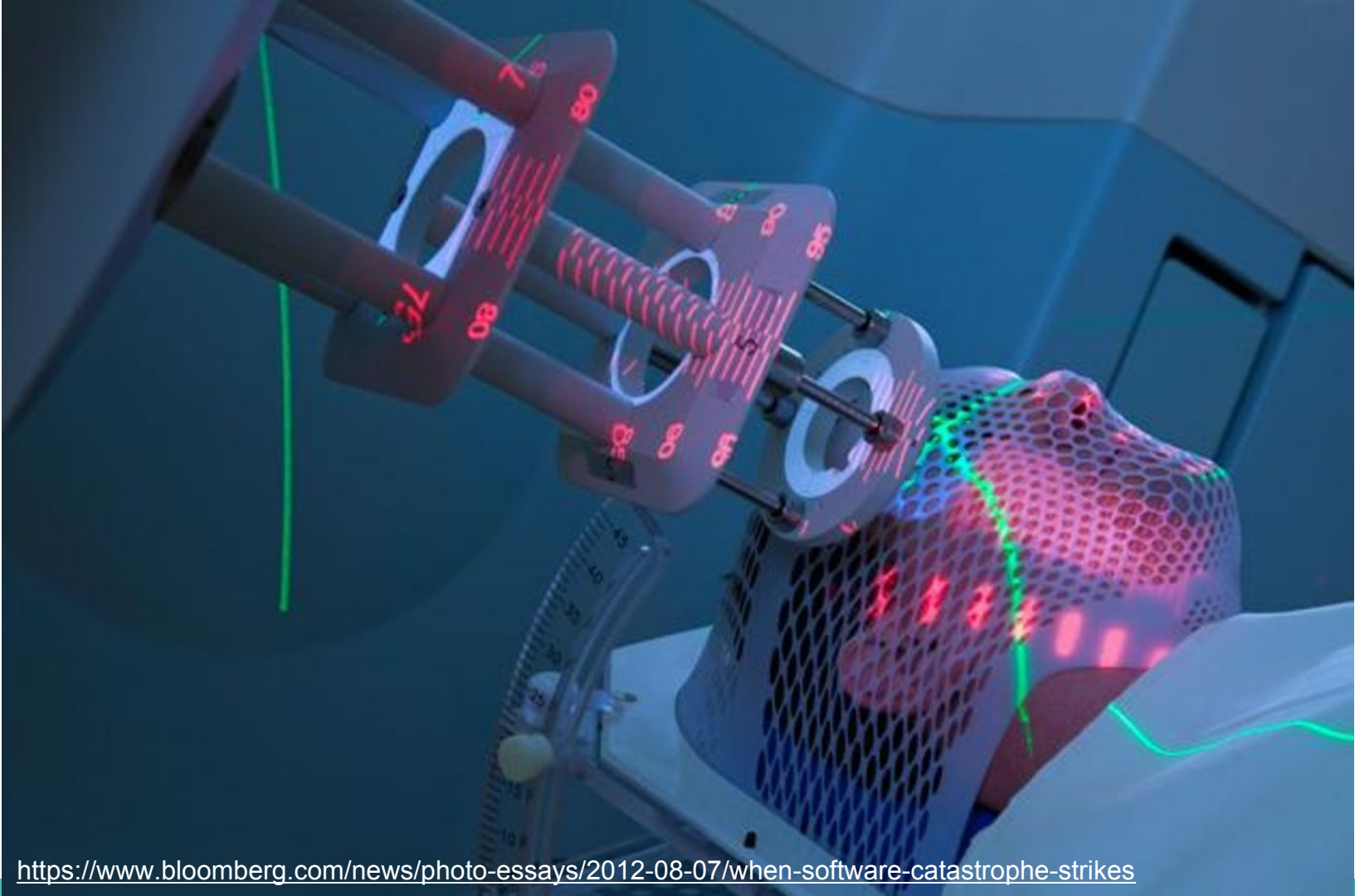
**El problema**



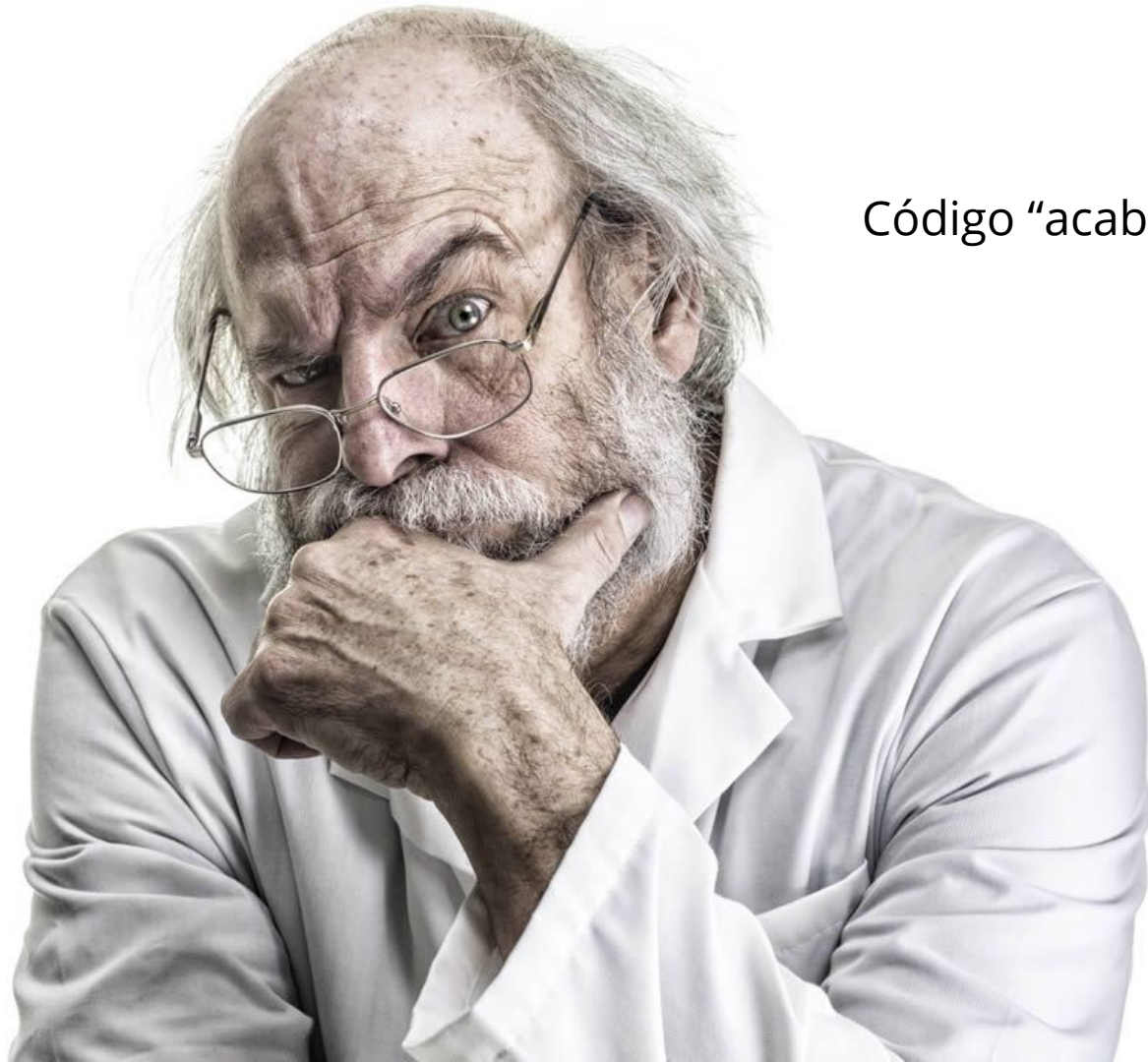
Ariane 5 (1996)  
40 segundos

\$500MM









Código “acabado”... ¿funcionará?

**DO THE  
THING  
RIGHT**

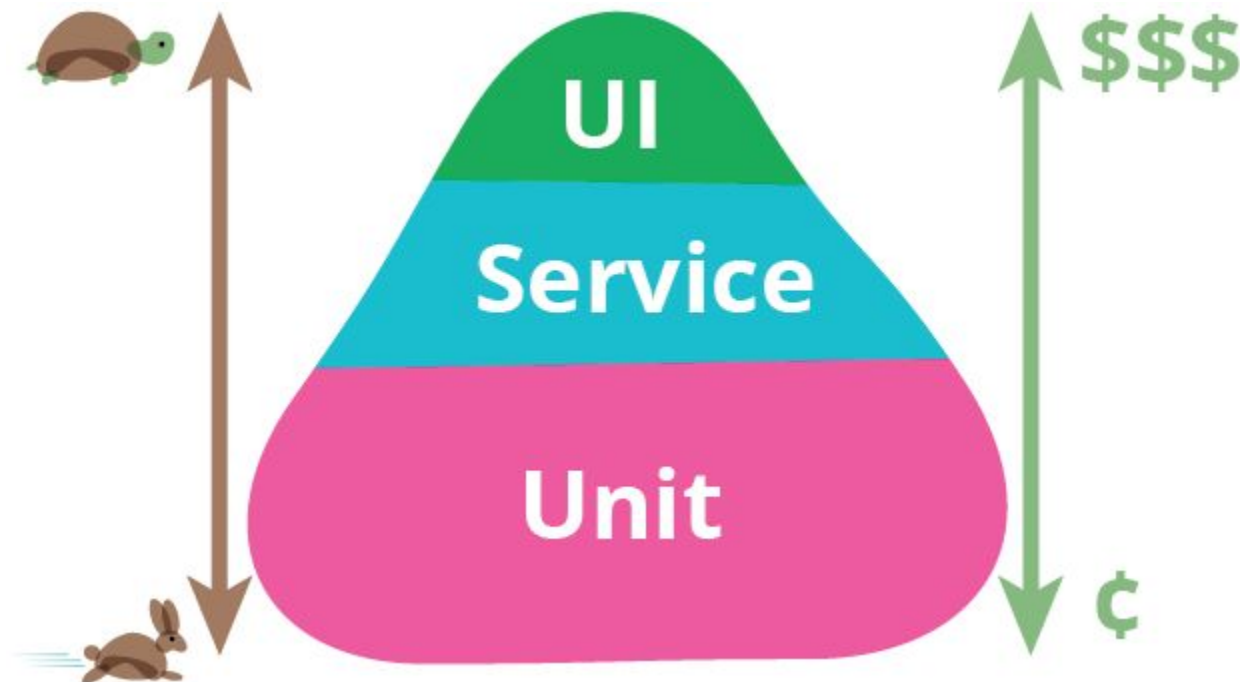


**DO THE  
RIGHT  
THING**

# Tests automatizados

- Red de **seguridad** = Tranquilidad
- Automatizar = Ahorro de **tiempo**
- **Documentación**
- Demuestran la **presencia de errores**, no su ausencia
- ¿Testear **todo**? Imposible y no deseable
- **QA** como rol, no como departamento

# Pirámide de testing



**UNIT TESTS PASSING**



**NO INTEGRATION TESTS**

## Testing by experiment

Feedback  
from  
humans  
(Meaning)

### User Behavior

- Usability tests
- Guerilla tests
- A/B tests
- MVP tests
- User bug reports

### Correct Functionality

- Manual UI tests
- Code review
- Explorative tests
- Legal or business requirements

### Software Behavior

- Stress tests
- Soak tests
- Error logs

### Stable Functionality

- Unit tests
- Compiler checks
- Automated UI tests
- Linters
- Static code analysis

Feedback from  
automation  
(Consistency)

## Testing against a specification



## ... y mucho más

- Chaos **testing**
- Tests **end-to-end**
- **Screenshot** testing:
  - e.g., <https://css-tricks.com/automating-css-regression-testing/>
  - Android: <https://github.com/karumi/shot>

# Estructura de un test: AAA

calculadora = MiCalculadora()      # **Arrange**

resultado = calculadora.suma(2, 2)      # **Act**

expect(resultado).to(equal(4))      # **Assert**

# ¿Cómo debería ser un test unitario/integración?

- **Fast**
- **Isolated/Independent**
- **Repeatable**
- **Self-Validating**
- **Thorough and Timely**

[https://github.com/ghsukumar/SFDC\\_Best\\_Practices/wiki/F.I.R.S.T-Principles-of-Unit-Testing](https://github.com/ghsukumar/SFDC_Best_Practices/wiki/F.I.R.S.T-Principles-of-Unit-Testing)

<https://pragprog.com/magazines/2012-01/unit-tests-are-first>

# ¿Cuándo ejecuto los tests?

- Integrado en el **IDE/editor/entorno**: ante un cambio, se reejecutan
- Cada **commit/push** al repo: hooks o <https://github.com/typicode/husky>
- **Integración continua** o Despliegue Continuo (Continuous Deployment)
  - Idealmente: entorno idéntico de tests y producción (e.g. con Docker)
  - Ejemplo: <https://github.com/islomar/url-shortener-islomar>
- Tareas configuradas vía **Webpack**, Gulp, Grunt, Gradle, Maven, etc.



Cobertura

Mutation testing

# Librerías y frameworks en JavaScript

- Jasmine
- **Mocha**
- **Chai**: librería de assertions
- Karma
- **Sinon.js**: [dobles de tests](#)
- Jest
- Protractor: end-to-end para Angular
- Nightwatch.js: tests funcionales web



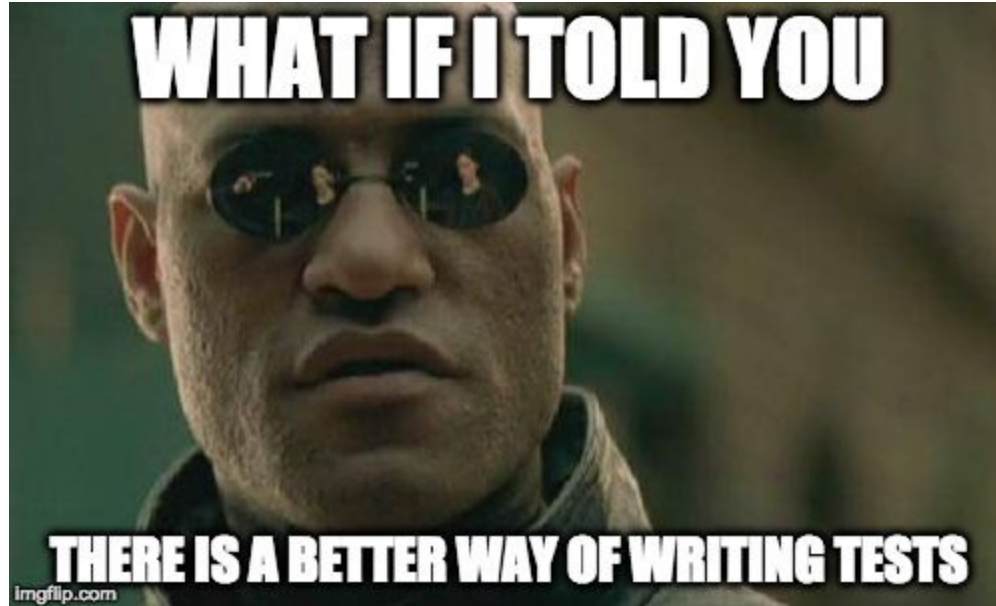
# Cuándo no escribiría tests automatizados

- **Spikes**, pruebas de concepto
  - Muy **alta incertidumbre**, exploración de una posible solución
- Otros: e.g. un script simplón que voy a ejecutar **una única vez**, no crítico

# Apreciaciones varias

- Código de test: **ciudadano de primera**, como el de producción
- Idealmente un **único assert** por test

# TDD: Test-Driven Development



# ¿Qué es TDD?

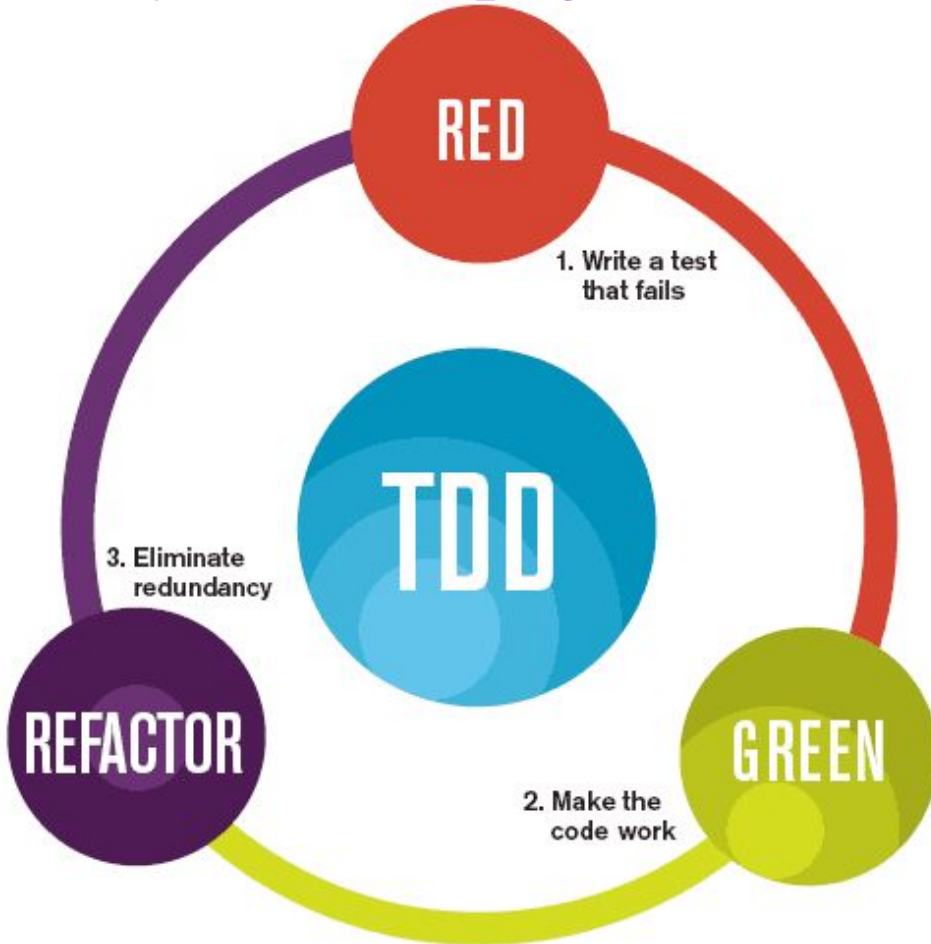
**TL; DR:** escribir el test antes que el código de producción

TDD es una **técnica de desarrollo de software**, una **disciplina**

# ¿Qué **NO** es TDD?

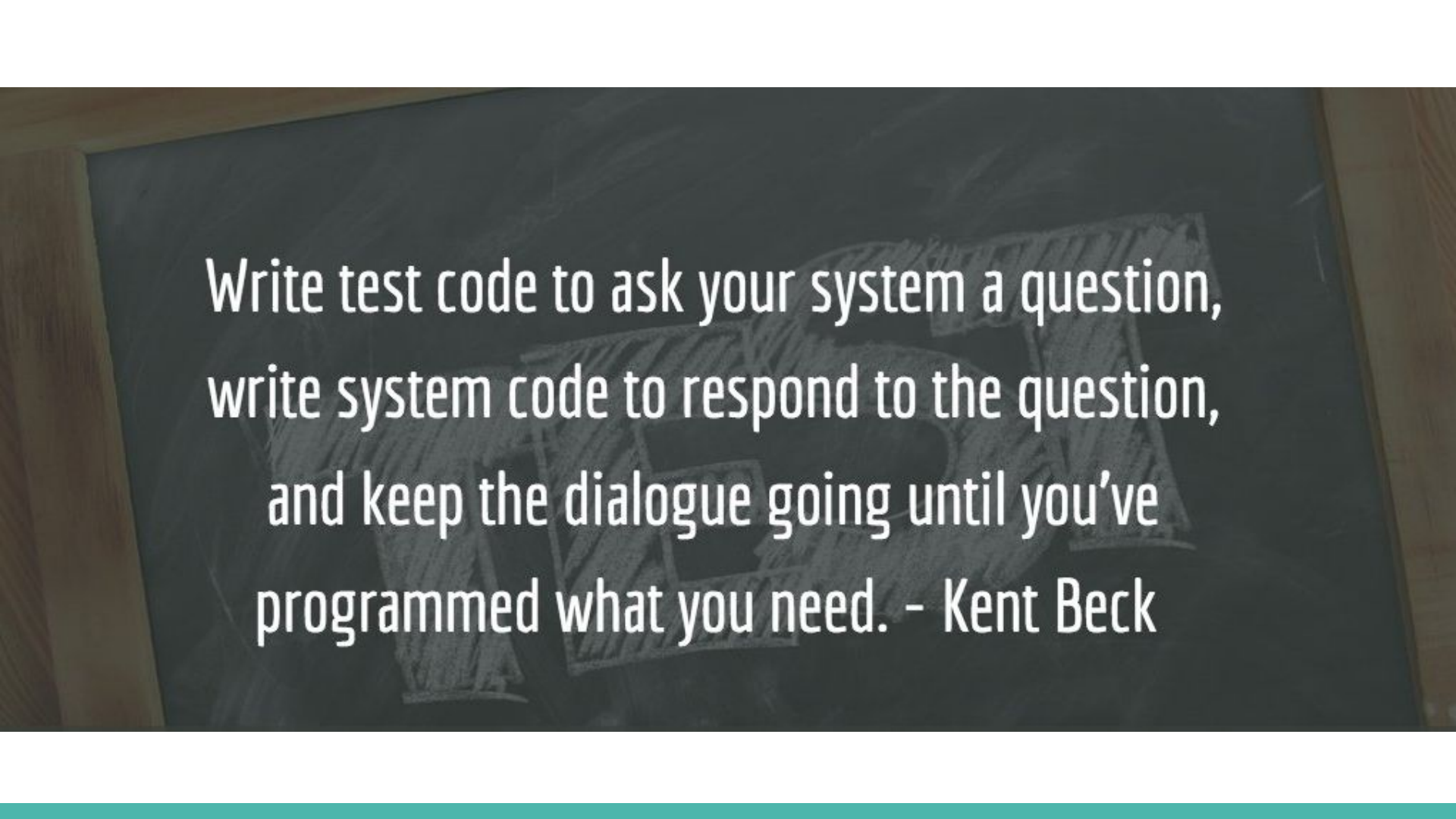
TDD **NO** es simplemente una forma de escribir tests

TDD **NO** es una técnica de diseño ni necesariamente mejora tu diseño



The mantra of Test-Driven Development (TDD) is “red, green, refactor.”



A chalkboard with a faint background drawing of a person in a suit. The text is written in white chalk on the dark surface of the board.

Write test code to ask your system a question,  
write system code to respond to the question,  
and keep the dialogue going until you've  
programmed what you need. - Kent Beck

# Qué aporta TDD (I)

- **FOCO** en la funcionalidad a desarrollar: escribir menos código
- **Feedback** rápido
- **Simplicidad**
- **Cadencia**, flow...

# Qué aporta TDD (II)

- **Descubrir** poco a poco cómo resolver el problema
- Crear mejores **tests** de tu código
- **Documentación** (muy probablemente mejor)
- Mejor **diseño** (si ya tienes los conocimientos)



**J. B. Rainsberger**

@jbrains

Test-first programming lets me program while drunk. As long as I can re-run the tests, I can remember what to do next.

12:10am · 31 Aug 2017 · YoruFukurou

1 REPLY 4 LIKES



Reply to @jbrains



**Tim Wright** @drtimwright

40s

Replying to @jbrains

I find it lets me recover into the flow faster when interrupted - for similar reasons.



# La regla dorada de TDD

Nunca escribas nueva funcionalidad sin un test que falle

# ¿Cuándo no haría TDD?

- **Spikes**
  - Muy **alta incertidumbre**, exploración de una posible solución: tras verificar, borrarlo y empezar con TDD
- Código MUY legacy (sin tests, pésima legibilidad, etc.)



# ¿Y todo esto, pa' qué?

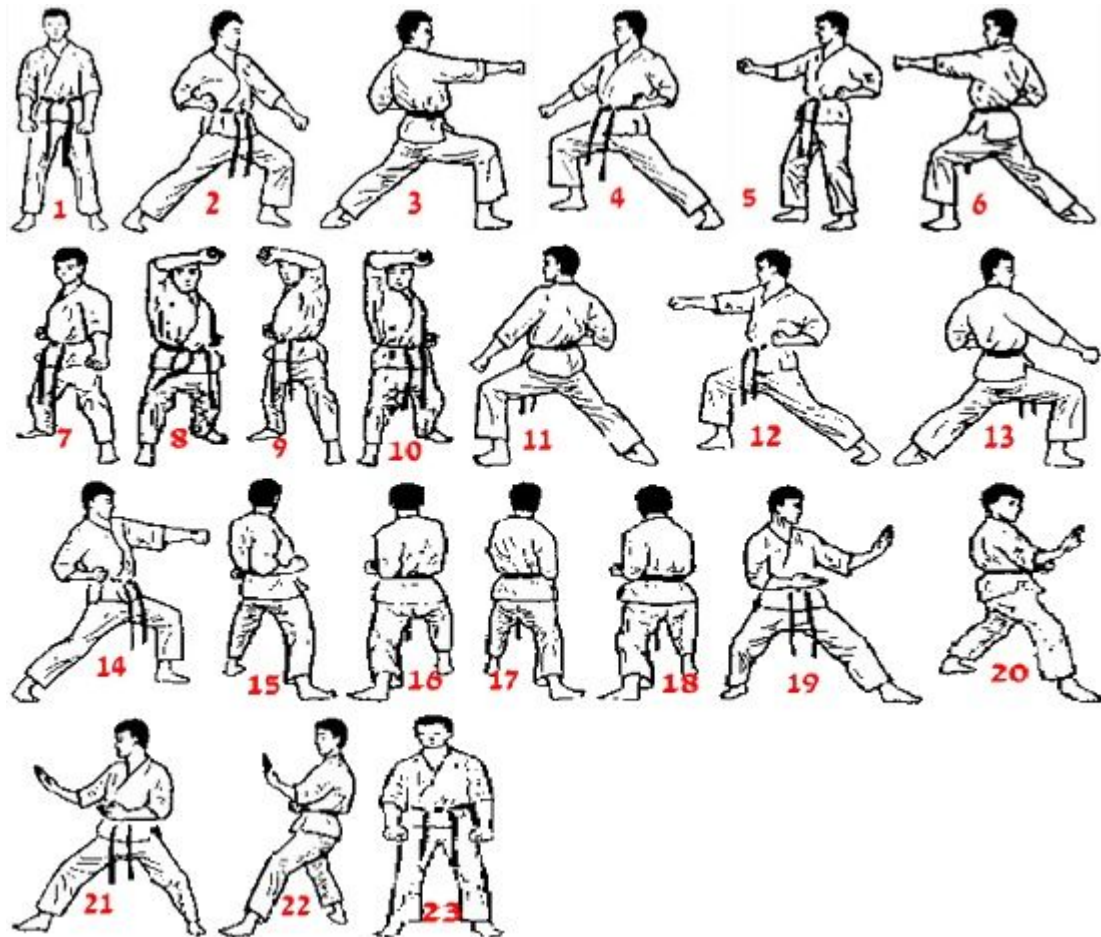
- Para vivir más **tranquilos**
- Para **disfrutar más** de tu trabajo (e incluso querer ir :-))
- ... y para servir a **negocio** (aka **aportar valor**): clientes más satisfechos, menor Time-To-Market, más beneficios, etc.
- Es NUESTRA responsabilidad (no “just following orders”)

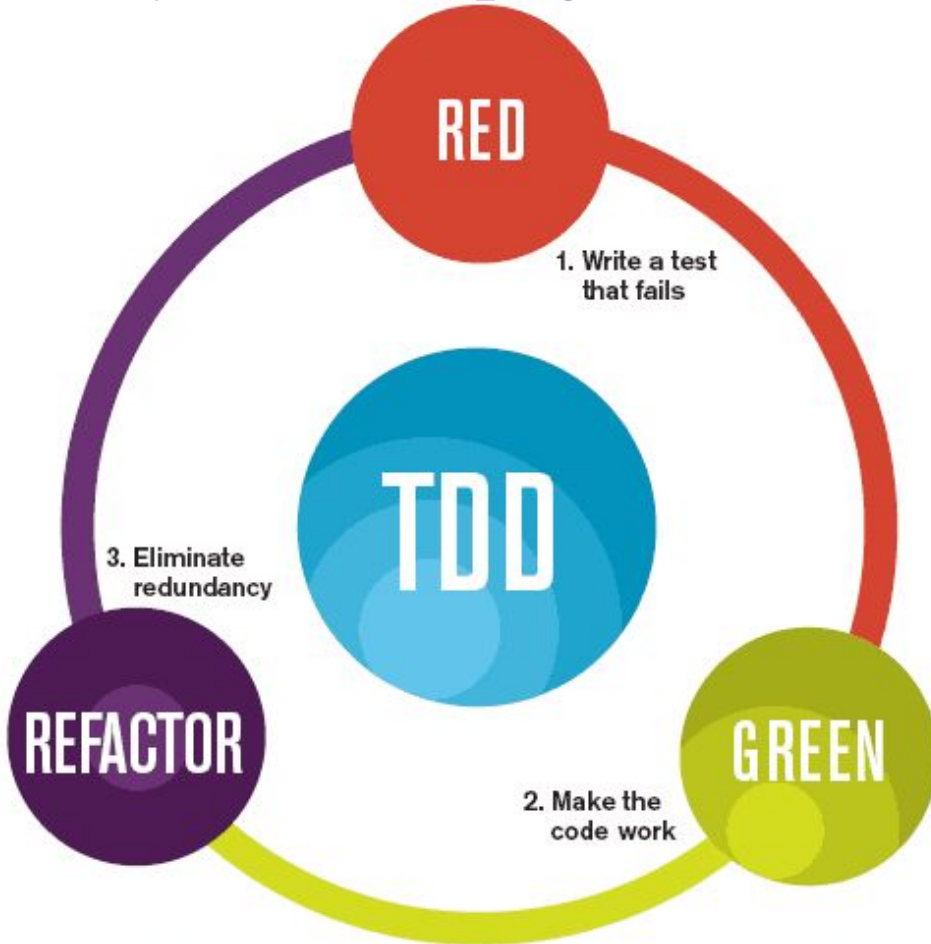
# Hands-on time!!

Pair programming

Pomodoros: ~25 min.

**Kata** FizzBuzz





The mantra of Test-Driven Development (TDD) is “red, green, refactor.”

# Kata FizzBuzz

Cómo vamos a hacerlo:

- **Pomodoro** (introducción + 2 iteraciones de 25 minutos cada una)
- **Pair programming** + Ping Pong
- Escribe lo mínimo para que funcione y pase el test

<http://www.solveet.com/exercises/Kata-FizzBuzz/11>

Escribe un programa que imprima los números del 1 al 100, pero aplicando las siguientes normas:

- Devuelve **Fizz** si el número es divisible por 3.
- Devuelve **Buzz** si el número es divisible por 5.
- Devuelve **FizzBuzz** si el número es divisible por 3 y por 5.

Salida de ejemplo:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
... etc hasta el 100
```

<http://www.solveet.com/exercises/Kata-FizzBuzz/11>

# Cómo realizar la práctica

Dos opciones:

- <https://codepen.io/islomar/details/PKLbzx/>
  - Pinchar en “Fork”
  - No es necesario crearse cuenta, pinchar en “Save as anonymous”
- <https://github.com/islomar/adalab-intro-testing-tdd>

# Round 1... code!!



# Bonus track

- Devolver **“Woof”** si un número es divisible por 7
- Devolver **“Fizz”** si es divisible por 3 **o si incluye** un 3 en el número
- Devolver **“Buzz”** si es divisible por 5 **o si incluye** un 5 en el número
- No usar **else**
- Un **único nivel de indentación** por método
- Sin usar **if()**



## Round 2... code!!



# Resumiendo, que es gerundio

- **Testing**: imprescindible... casi siempre
- **TDD**: técnica de desarrollo, más que recomendable
- Prueba y **juzga por ti misma**
- Rodéate de buena gente con experiencia (y busca una **mentora/mentor**)
- A nivel profesional, **el software es un medio para un fin**: ¡nunca lo olvides!



# Bibliografía

- [TDD by example](#) (Kent Beck)
- [Growing Object-Oriented guided by tests](#) (Steve Freeman, Nat Pryce)
- [Diseño ágil con TDD](#) ([Carlos Blé](#))
- [Specification by example](#) (Gojko Adzic)
- [Refactoring](#) (Martin Fowler)
- [Clean Code](#) (Uncle Bob)
- [XP Explained](#) (Kent Beck)

# Blogs

<http://blog.adrianbolboaca.ro/>

<http://blog.jbrains.ca/>

<https://codurance.com/publications/>

<https://www.codesai.com/publications/>

# Screencasts

- Screencasts de Sandro Mancuso:
  - Algorithms: <https://www.youtube.com/watch?v=iZjgj1S0FCY>
  - Outside-In TDD: <https://www.youtube.com/watch?v=XHnuMjah6ps>
- Carlos Blé
  - [Implementando algoritmos con TDD](#)
  - [Kata sobre juego de cartas](#)
- Twitch:
  - Guillermo Gutiérrez: <https://www.twitch.tv/ggalmazor>
  - <https://www.twitch.tv/wedotdd/videos/all>

# Charlas

- Joaquín Engelman (@kinissoftware)
  - [Adicto al verde](#)
  - [Dando amor a los tests](#)
- [The limited red society](#)
- [The three laws of TDD](#)
- [Test-Bridle Development](#) (@flipper83), SCPNA

# Artículos y webs de interés

- <https://martinfowler.com/articles/microservice-testing/>
- <http://www.agiledata.org/essays/tdd.html>
- <https://medium.com/@ramtop/what-im-talking-about-when-i-talk-about-tdd-546a383468be>
- <https://medium.com/powtoon-engineering/a-complete-guide-to-testing-javascript-in-2017-a217b4cd5a2a>
- Ideas para katas: <https://github.com/12meses12katas>



# Tutoriales

- [Gentle introduction to JavaScript TDD](#)
- <https://github.com/dwyl/learn-tdd>
- Katas para aprender ES6 con tests: <http://es6katas.org/>

# Cursos

<https://www.codesai.com/curso-de-tdd/>

<http://www.jbrains.ca/training/the-worlds-best-introduction-to-test-driven-development/>

<http://www.codemanship.co.uk/tdd.html>

Pluralsight: <https://www.pluralsight.com/search?q=TDD>

# Comunidades y Meetups

<https://www.meetup.com/es-ES/madswcraft/>

<https://www.meetup.com/es-ES/madriagil/>

<https://www.meetup.com/es-ES/MADQA-Grupo-Meetup-de-QA-y-TESTING-de-SOFTWARE-en-Madrid/>

# *¡Gracias!*

¿Comentarios, dudas, preguntas?

¡¡Feedback, por favor!!

How's it going?



@islomar



islomar@gmail.com

The image features a series of concentric circles. The outermost circle is a dark red. Inside it is a lighter red circle, followed by a medium red circle, and then a bright red circle. At the very center is a solid dark blue circle. Overlaid on these circles is the text "That's all Folks!" in a white, elegant cursive script. The text is positioned diagonally, starting from the lower left and ending at the upper right, with the dark blue center circle acting as a backdrop for the word "Folks".

*That's all Folks!*