




Desarrollo dirigido por pruebas (*Test-Driven Development*)

Raúl García García
Curso 2024/2025
Universidad San Pablo-CEU
Escuela Politécnica Superior
Campus de Montepríncipe

1




Motivation

- » One of the goals of the course is to introduce new development methodologies such as *eXtreme Programming (XP)* and *Test-Driven Development (TDD)*
 - TDD sits nicely in the XP “way of doing things”
 - › TDD can be used *without* practicing XP
 - Giving lectures is one such way of achieving that goal
 - › Practising them is *better*
 - Reduce the amount of re-testing that is required
 - › Especially when dealing with legacy applications
 - › Avoid introducing new bugs after *refactoring* existing code
- » TDD has “broader goals” that just insuring quality
 - Improve developers lives (coping, confidence)
 - Support design flexibility and change
 - Allow iterative development with working code early

18-ene.-25 Curso 2024/2025 página 2

2




What is TDD?

» “Before you write code, think about what it will do. Write a **test** that will use the methods you haven’t even written yet”

Extreme Programming Applied: Playing To Win
Ken Auer, Roy Miller (“The Purple Book”)

» A **test** is not something you “do”, it is something you “**write**” and run once, twice, three times, etc.

- It is a piece of code
 - › Testing is therefore **automated**
- Repeatedly **executed**, even after small changes
- As much about **design** as about **testing**
 - › Encourages design from a user’s point of view
 - › Encourages testing classes in isolation
 - › Produces **loosely-coupled, highly-cohesive** systems



18-ene.-25


Curso 2024/2025

página 3

Acto de design mas que de verificacion

Desde el punto de vista de otros objetos que se relacionan con el objeto con el que se quiere probar

3



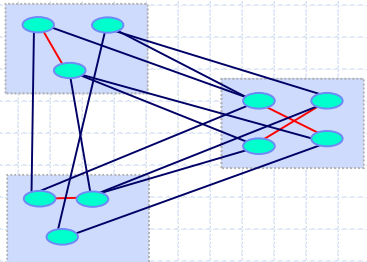
Sidenote: Cohesion and coupling

» Each module should be **highly cohesive**

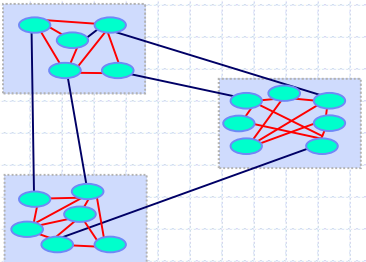
- module understandable as a **meaningful unit** (clarity)
- components of a module are **closely related** to one another

» Modules should exhibit **low coupling**

- modules have **low interactions** with others
- understandable **separately**



- high coupling, low cohesion



+ low coupling, high cohesion

18-ene.-25

Curso 2024/2025

página 4

fuera acoplamiento, dentro cohesion

4

What is TDD?

- » TDD is the process of thinking about a block of code from the perspective of the user of that code
- » TDD is a technique whereby you write your test cases before you write any implementation code
- » An iterative technique for developing software
 - Tests drive or dictate the code that is developed
 - › Write a test that shows what you want the code to do
 - › Letting your codebase grow organically as you create examples of what the code should do
 - Software is written from the outside in: tests provide a specification of “what” a piece of code actually does
 - › Some say that “tests are part of the documentation”
 - › Other say that “tests are part of the design”

18-ene.-25 Curso 2024/2025 página 5

5

TDD vs other kinds of testing

» <http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants>

Business Facing (top) / **Technology Facing** (bottom)

Support Development (left) / **Critique Product** (right)

Quadrants and Annotations:

- Customer Tests (Q2)**: Automated & Manual
- Exploratory Tests (Q3)**: Manual
- Programmer Tests (Q1)**: Automated (circled in red)
- Scalability Performance Usability Security (Q4)**: Tools

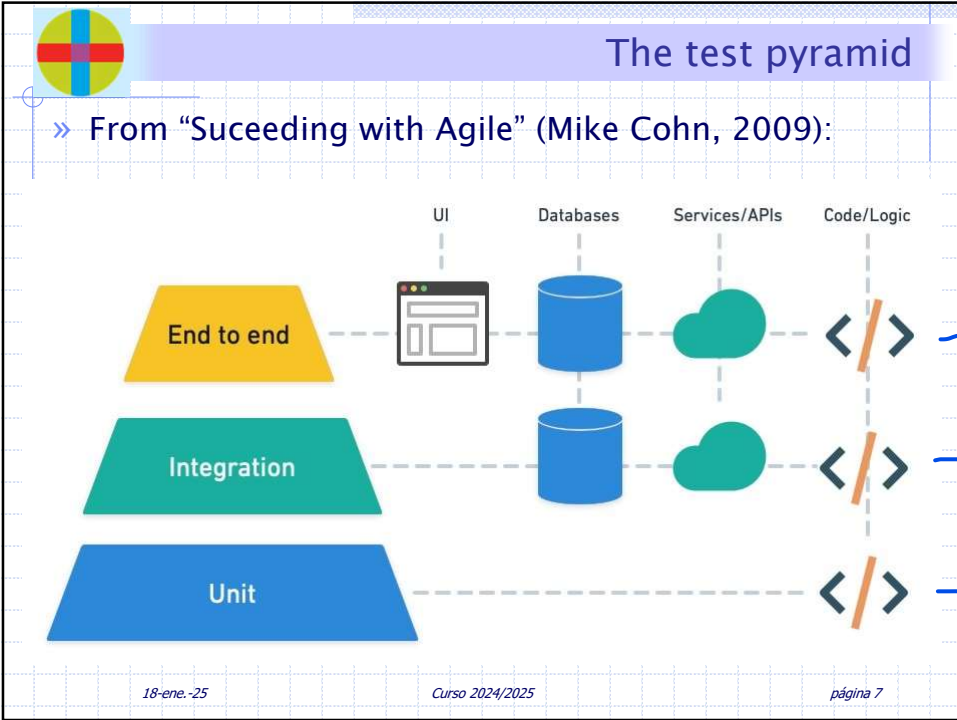
Handwritten notes on the right:

- Pruebas para ver como funcionan cosas que no conoces. Como de elegir lenguaje a model de db, u otras cosas. Se hace una prueba fuera de el programa para probar.
- Para hacer estas pruebas haya que tener un programa que funcione y que sea muy usado

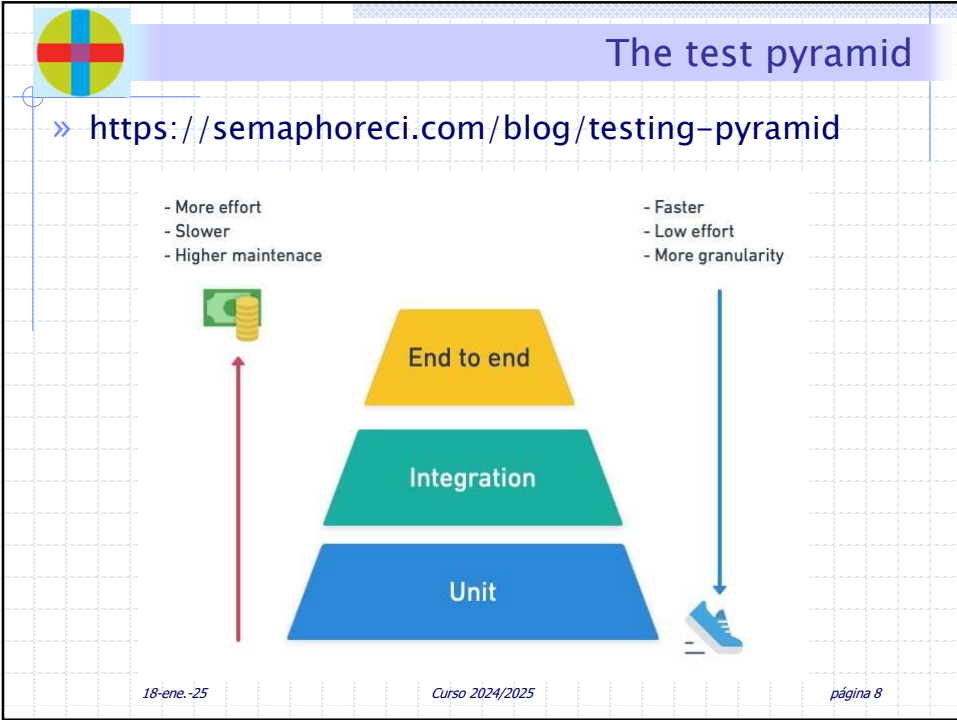
18-ene.-25 Curso 2024/2025 página 6

6


El TDD va sobre todo de Unit Tests, aunque tambien aplicable sobre los Integration Y End to End Tests



7



8




Why TDD?

- » Programmers **dislike testing**
 - They will test reasonably thoroughly the first time
 - The second time however, testing is usually less thorough
 - The third time, well...
- » Testing is considered a **boring task**
 - Testing might be the job of another department / person
- » TDD encourages programmers to maintain an **exhaustive set of repeatable tests**
 - With tool support, tests can be run selectively
 - The tests can be run after every single change
 - Must be learned and practiced: **if it feels natural at first, you're probably doing it wrong**
- » More **productive** than *debug-later* programming
 - Developers work in a **predictable** way of **developing** code
 - *It's an addiction rather than a discipline* – Kent Beck

18-ene.-25 Curso 2024/2025 página 9

9



Which is TDD?

- » Bob Martin: “The act of writing a unit test is more an **act of design** than of verification”
 - Disclaimer: **Developer** = **Programmer**
- » We're talking about **unit testing** – i.e. testing the internals of a class
 - There is some debate about what constitutes a **unit**
 - Here are some **common definitions** of a unit:
 - › The smallest chunk that can be compiled by itself
 - › A stand-alone procedure or function
 - › Something so small that it would be developed by a single person
 - **Black box** testing for objects
 - › Classes are testing in **isolation**
 - › Remember: the goal is to produce loosely coupled, highly cohesive architectures

18-ene.-25 Curso 2024/2025 página 10

10

Who should write the tests?

- » The **programmers** should write the tests
 - They can't wait for somebody else to write tests
- » TDD promotes "**small steps**", and lots of them
 - Small steps: the **real** shortest distance between two points
 - Use TDD to get from A to B in very small, verifiable steps
 - › You often end up in a better place



BDUF (Big Design Up-Front)
Software development method where a "big" design is created before coding and testing takes place.

Spike
A quick (minutes, hours) exploration by coding of an area in which the development team lacks confidence.

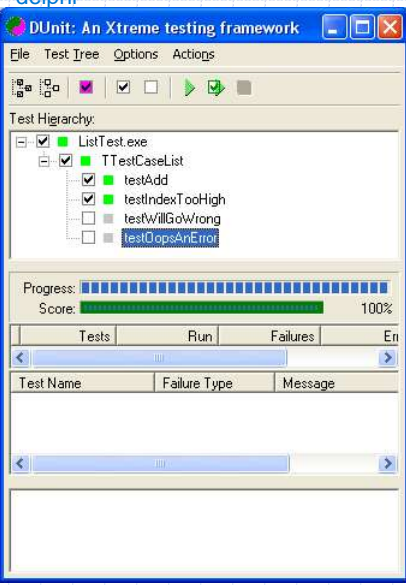
18-ene.-25 Curso 2024/2025 página 11

No siempre acabas en el punto inicial que pensabas (B) a veces acabas en un sitio cercano pero no exacto (S)

11

Unit Testing Frameworks

- » Test must be **automated**
 - If not, they won't be run
- » **xUnit** is a colloquial umbrella term
 - Most languages have a testing framework
 - **JUnit, CppUnit, PyUnit, XMLUnit**
 - › Simple tool (platform-specific implementations)
 - › Collects, organizes and automatically calls your test code
- » Graphical test runner
 - **Green bar** makes you feel good



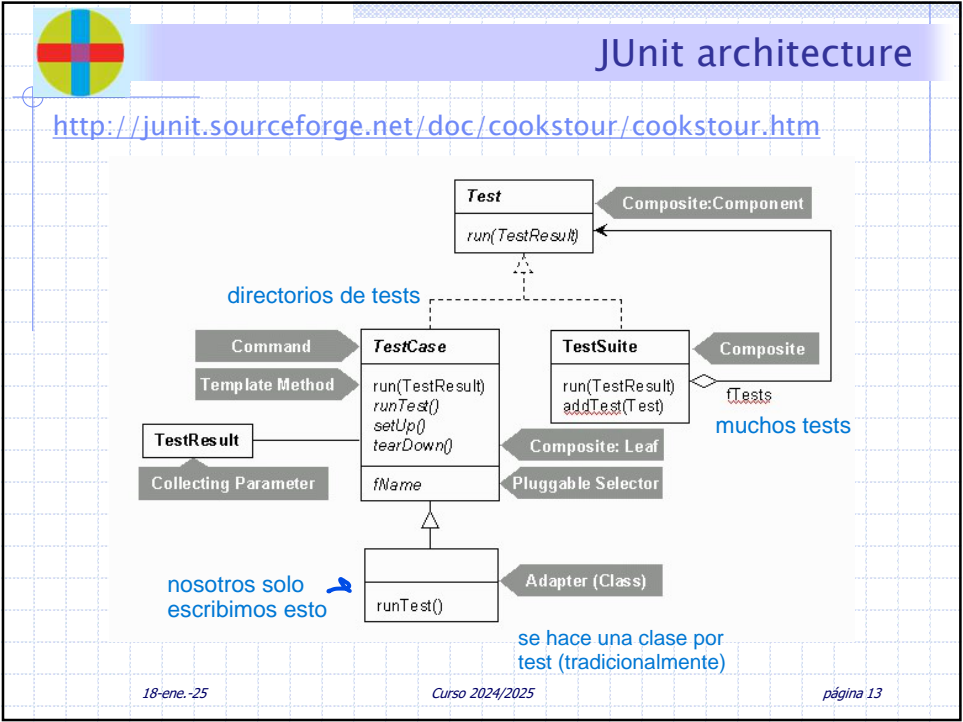
18-ene.-25 Curso 2024/2025 página 12

12

cursivo: abstracta

interfaz: clase abstracta pura, sin ningun metodo implemetado (abstracta nomal, tiene implemetados y sin implemetar)

No devuelte true or false, simplemente ejecuta y deja anotaciones que navega por las otras clases



No se puede crear un objeto de una clase abstracta

Con el composite no tienes que saber la jerarquia

La herramienta exec el composite

ni crea ni los destruye los tests
Mete al test en la lista de ftests

Devuelve

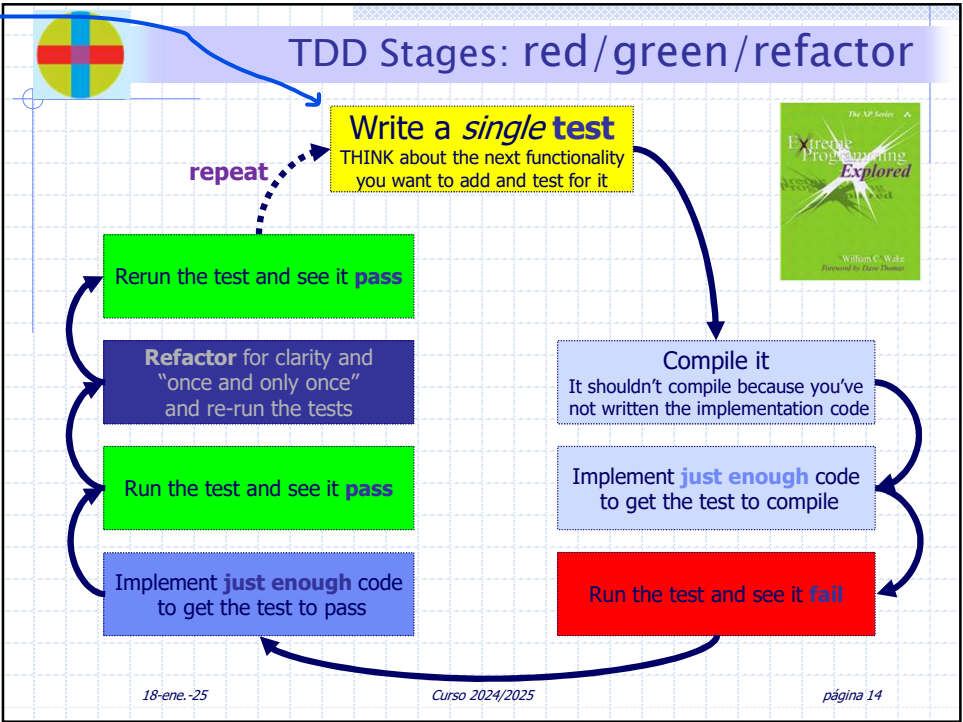
13

Cuando escribes un test, estas designing, por lo que estas pensando. De uno en uno, para un mejor design.

El refactor es para mejorar el codigo, no para meter funcionalidad. No añade nada, se hace para que sea mas facil pasar de caja roja a verde en ciclos futuros.

La caja verde es donde se para cuando funciona todo. No se para hasta que funciona todo.

Implementar lo sufieciente para que el test pase, hacer commit. Despues se ve si se quiere mejorar el codigo.



Cuanto mas se repite el ciclo, mas dificil sera el pasar de la casa roja a la verde.

Se quiere pasar el menos tiempo posible en la parte de implementar, hay que pasar mas tiempo en pensar.

Se intenta compilar, pero sin tener nada implementado, no debe compilar. Y despues se hace lo suficiente para que compile, no para que funcione.


14

Los ciclos deberian de ser muy cortos, teoricamente ~10 min

Test tiene que tener tanto las preguntas como las respuestas correctas. (elegir buenos nombres)

Ahora que funciona, elimina la basura y hazlo bien

Si lo haces demasiado bien y complicado desde el principio, te vas a empezar a complicar rapidamente, y se te acaba la vida.



A Test Life Cycle

1. **Write a test (red): the test fails**
 - The test must not work; must not even compile at first
 - Think about how you would like the feature to appear in code
 - Invent the API you wish you had
 - Include all the elements in the story that you imagine will be necessary to calculate the right answers
2. **Make it run (green): make the test pass**
 - Make the test work quickly, doing whatever sins be necessary
 - Quickly getting the test to pass dominates everything else
 - If a clean simple solution is obvious, type it in
 - If the clean, simple solution is obvious but will take a minute, make a note of it and get back to the main problem – making the test pass
 - Quick test success excuses all sins, but only for the moment
3. **Make it right (refactor): change the design**
 - Now that the system is ok, put the sins of the recent past out
 - Step back onto the straight and narrow path of software righteousness
 - Eliminate the duplication created in just getting the test to work


18-ene.-25
Curso 2024/2025
página 15

Que no pierdas tiempo, que llegues al green lo antes posible. Usando cualquier manera.

Y una mierda que funciona, es mejor que algo decente pero que no funciona. Lo que funciona es entregable, lo que no funciona no lo es.

La limpieza se hace despues de que nos hayan contratado.

15



TDD Process

- » **Features and benefits**
 - Once a test passes, it is re-run with every change
 - Broken tests are not tolerated
 - Side-effects defects are detected immediately
 - Assumptions are continually checked
 - How should running of tests affect one another?
 - Not at all – isolated tests
 - Automated tests provide a safety net that gives you the courage to refactor
- » **What do you test?**

everything that could possibly break – Ron Jeffries
- » **Don't test anything that *could not possibly break* (always a judgment call)**
 - Example: Simple accessors and mutators los getters y los setters

18-ene.-25
Curso 2024/2025
página 16

Los tests no estan relacionados, cada uno prueba una parte por separado y no se deben de afectar entre si.

Tests que se afectan entre ellos son tests de integracion, un tema diferente.

En teoria habria que hacer muchisimos tests para cubrir todas las funcionalidades. Caso 80-20, 20% de los tests que cubren el 80% de la funcionalidad

16



TDD Process

1. *Start small or not at all* (select one small piece of functionality that you know is needed and you understand)
2. Ask “what set of tests, when passed, will demonstrate the presence of code **we are confident fulfills this functionality correctly?**”
3. Make a **to-do list**, keep it next to the computer
 1. Lists tests that need to be written
 2. Reminds you of what needs to be done
 3. Keeps you focused
 4. When you finish an item, cross it off
 5. When you think of another test to write, add it to the list

Empieza los tests por donde mejor te sale.

18-ene.-25

Curso 2024/2025

página 17

17



Automated Unit Tests: Right

- » Unit tests show the programmer that **the code does what is expected to do**
 - Specifies what the **code must do** (**specification by example**)
 - Provides examples of how to use the code (documentation)
 - All test are run every few minutes, with every change
- » Right: Are the results **right**?
 - **Validate results**: use the acronym **BICEP**
 - Does the expected result match what the method does?
- » If you don't know what **right** would be, then how can you test? How do you know if your code works?
 - Perhaps requirements not known or stable
 - **Make a decision**. Your tests document what you decided. Reassess if it changes later

Da mucha info como los tipos, los nombres, los returns, ...

Si no sabes realmente como tiene que funcionar el test, hazlo de todas maneras, porque deja documentada la decision.

Como si quieres implementar la `sqrt(-1)` como una excepcion, o como numero imaginario i. Decide el disenno.


18-ene.-25

Curso 2024/2025

página 18

18

Dejar apuntado de que quieres hacer tests, ya sea dejandolos escritos los nombres o con un to-list



Right-BICEP


- » **Boundary** conditions: consider
 - Garbage input values
 - Badly formatted data
 - Empty or missing values (0, null, etc.)
 - Values out of reasonable range
 - Duplicates (if they're not allowed)
 - Unexpected orderings
- » Use this acronym, **CORRECT**, to remember:
 - **C**onformance Funcionalidad
 - **O**rdering
 - **R**ange
 - **R**eference: Does the code reference anything external outside of its control?
 - **E**xistence
 - **C**ardinality
 - **T**ime (absolute and relative time): Are things happening in order? On time? In time?

Deberíamos de escribir test de frontera, considerando valor extrannos.

Tiempo relativo es mas imporetante, casos como que no hace falta que solo te puedas matricular a una unica hora.

18-ene.-25 Curso 2024/2025 página 19

19



Right-BICEP

- » Check **Inverse** relationships
 - If your method does something that has an inverse, then apply the inverse
 - › Square and square-root
 - › Insertion then deletion.
 - Beware errors that are common to both your operations
 - › Seek alternative means of applying inverse if possible
- » **Cross-check** using other means
 - Can you do something more than one way?
 - › Your way, and then the other way. Match?
 - Are there overall consistency factors you can check?
 - › Overall agreement

Si tienes un codigo que hace cuadradas y raices cuadradas, pues haz un test que haga la cuadrada de la raiz cuadrada.

Pero cuidado que hay inversas de inversas que pueden no dar lo mismo, puede depender del design que se haya elegido.


Y si tengo dos maneras distintas de hacer la misma cosa? Probar ambas maneras y probar igualdad entre ellos.

Cuidado de que los tipos sean equivalentes, a lo mejor la aproximacion puede ser diferente.

Se suelen dar cuota de error

18-ene.-25 Curso 2024/2025 página 20

20



Right-BICEP


- » Force **Error** conditions
 - Some are easy: invalid parameters, out of range values, ...
 - Others not so easy: exceptions, ...
 - Failures outside your code:
 - > Out of memory, disk full, network down, etc.
 - > Can simulate such failures
 - Example: use Mock Objects
- » Performance
 - Perhaps absolute performance, or
 - Perhaps how performance changes as input grows
 - Perhaps separate test suite in JUnit
 - But... Other, perhaps better ways to do this
 - > JUnitPerf: <https://github.com/clarkware/junitperf>

18-ene.-25 Curso 2024/2025 página 21

Un test unitario repetido muchas veces

Si tienes un test para por ejemplo hacer una factura completa, se le puede pasar a un programa como este para que lo repita muchas veces.

21



Sidenote: Mock Objects


- » Fake objects that **replace** real ones
- » Why use them?
 - Avoid external dependencies
 - Reduce coupling
 - Keep tests fast
 - Test object interactions
 - Promote interface based design
 - Ensure tests are durable
- » Example: Sending email?
- » Web site: <http://www.mockobjects.com>

18-ene.-25 Curso 2024/2025 página 22

Cuidado que si tienes que usar demasiados mocks en tu código puede indicar muchas dependencias en tu código.

22

Leer los errores que da el compilador como si fuera una to-do list de lo que tienes que hacer para implementar y que compile ese test. (el primeto suele ser el importante)



Continuing ...

- » Use the **compiler**
 - Let it tell you about errors and omissions
 - Read carefully its warning and error messages
- » **One assertion (check/assert) per test**
 - Subject of furious debate on Yahoo's TDD group
- » Use this **four techniques**
 1. Design: **Do the Simplest Thing**
 2. **Fake It ('Til You Make It)**
 3. **Triangulate**
 4. **Obvious Implementation**


Solo un assert por test para saber porque ha fallado, si prueba varias cosas no sabras porque ha fallado

18-ene.-25 Curso 2024/2025 página 23

23

Analizar bien los requisitos, no hacer lo que no se nos pide.

Empezar con las cosas faciles y las que sabes hacer. Ya se ira progresando a lo mas dificiles



Design: Do the Simplest Thing


- » **Do the simplest thing that works**
 - **KISS Principle** – Keep It Simple and Stupid
 - › Consider the simplest thing that could possibly work
 - › Simplicity is the ultimate sophistication – Leonardo Da Vinci
 - › Perfection is reached not when there is nothing left to add, but when there is nothing left to take away – St. Exupéry
- » **When Coding: YAGNI** – You Aint't Gonna Need It
 - Build the simplest possible code that will pass the tests
 - › Ron Jeffries: "Always implement things when you **actually** need them, never when you just **foresee** that you need them"
 - **DRY Principle** – Don't Repeat Yourself
 - › Refactor the code to have the simplest design possible
- » **Bad smells**
 - Duplication: eliminate it → **OAOO** – Once And Once Only
 - SetUp / TearDown: duplicated code reveals itself as common 'initialization' code and/or as 'cleanup' code

18-ene.-25 Curso 2024/2025 página 24

24

Si solo tienes un test, es para hacer el diseño. De primeras pones que devuelva lo que haga falta para que pase el test.

El día que te preocupe implementarlo bien, ya lo haras.



Fake It ('Til You Make It)

» What is your first implementation once you have a broken test?

- Return a constant
- Once you have a test running, gradually transform the constant into an expression using variables

» Example from **PyUnit** implementation

- First version

```
return "1 run, 0 failed"
```

- Second version

```
return "%d run, 0 failed" % self.runCount
```

- Third version

```
return "%d run, %d failed" % (self.runCount, self.failureCnt)
```

18-ene.-25


Curso 2024/2025

Beck

página 25

25

Cuando ya tienes un test escrito que funciona, usalo de nuevo para triangular.



Triangulate

» How do you most conservatively drive abstraction with tests?

» Abstract only when you have two or more examples

```
import unittest

class TestClass(unittest.TestCase):
    def test_plus(self):
        self.assertEqual(4, plus(3,1))

def plus(augend, addend):
    return 4
```

Test 1

Code

Test 2

```
import unittest

class TestClass(unittest.TestCase):
    def test_plus_1(self):
        self.assertEqual(4, plus(3,1))
    def test_plus_2(self):
        self.assertEqual(7, plus(3,4))
```

18-ene.-25

Curso 2024/2025

página 26

26



Obvious Implementation

- » How do you implement simple operations?
 - Just implement them
- » *Fake It* and *Triangulation* are for taking tiny steps
- » If you know what to type, and you can do it quickly, then do it
- » Keep track of how often you are surprised by red bars using *Obvious Implementation*

Solving “clean code” at the same time that you solve “that works” can be too much to do at once. As soon as it is, go back to solving “that works”, and then “clean code” at your leisure.

Kent Beck

18-ene.-25

Curso 2024/2025

página 27

27

No te pongas a hacer código limpio hasta que el código/implementación funcione. Ya cuando tengas tiempo lo limpias.

No te pongas a dar taconazos hasta que sabes que has ganado el partido



After the first cycle you have...

1. Made a list of tests we knew we needed to have working
2. Told a story with a snippet of code about how we wanted to view one operation
3. Made the test compile with stubs
4. Made the test run by committing horrible sins
5. Gradually generalized the working code, replacing constants with variables
6. Added items to our to-do list rather than addressing them all at once

18-ene.-25


Curso 2024/2025

página 28

28

Al final se te quedan una lista de tests que tenemos empezados.

No te preocupes de 'hacerlo bien' o 'bonito'



Refactoring

- » Refactoring cannot change the semantics of the program under any circumstance
- » This is called *Observation equivalence*
 - All test that pass before refactoring must pass after it
 - Places burden on you to have enough tests to detect unintended behavioral changes due to refactoring
- » Levels of scale
 - Two **loop structures** are similar
 - › By making them identical, you can merge them
 - Two **branches of a conditional** are similar
 - › By making them identical, you can eliminate the conditional
 - Two **methods** are similar
 - › By making them identical, you can eliminate one
 - Two **classes** are similar
 - › By making them identical, you can eliminate one

18-ene.-25 Curso 2024/2025 página 29


Un refactor no puede cambiar la funcionalidad del programa. (Comportamiento observable)

No cambia el comportamiento observable si siguen pasando todo los tests.
Tienen que haber tests para poder hacer refactor.

Cuando se hacen los refactor

Cuando hay duplicacion

29



Summary

- » TDD does not replace traditional testing
 - It defines a proven way that ensures effective unit testing
 - Tests are working examples of how to invoke a code
 - › Essentially provides a working specification for the code
 - No code should go into production unless it has associated tests
 - › Catch bugs before they are shipped to your customer
 - No code without tests
 - Tests determine, or dictate, the code
- » TDD isn't new
 - To quote Kent Beck: "...you type the expected output tape from a real input tape, then code until the actual results matched the expected result..."
- » TDD means less time spent in the debugger


18-ene.-25 Curso 2024/2025 página 30

No remplaza las pruebas a mano, hay tipos de tests en la que tienes que hacer casos practicos mas complejos manualmente. Como tests para los clientes.

Al final se usa mucho menos el debugger, ya te van guiando los errores de los tests

30

Te quita el miedo a hacer cambios. Puedes programar mas tranquilamente.



Summary


- » TDD **negates fear**
 - Fear makes developers communicate less
 - Fear makes developers avoid repeatedly testing code
 - › Afraid of negative feedback
- » TDD creates a **set of “programmer tests”**
 - Automated tests that are written by the programmer
 - Exhaustive: can be run over and over again
- » TDD allows us to **refactor**, or change the implementation of a class, without the fear of breaking it
 - TDD and refactoring go hand-in-hand
- » With care, [some] **User Acceptance Tests** can codified and run as part of the TDD process

18-ene.-25 Curso 2024/2025 página 31

Se podría llegar a usar como test de clientes con cierto cuidado y algunos cambios

31

Los tests dan una idea del proceso de desarrollo y explica el código. Puedes ponerte a leer los test para saber lo que hace el programa.




Benefits of TDD

- » How can it **help my project?**
 - You will produce better code and designs
 - Your code will be easier to maintain when others' read it days/months/years from now
 - You will have more confidence when you code
 - It will make your life easier
- » How can I **convince my...**
 - ...**coworkers?**
 - › Just do it. They will see how well it works.
 - ...**team?**
 - › Just do it. When someone asks for help, help them by writing a test.
 - ...**boss?**
 - › Start with a small project (but not too small)
 - › Measure the results
 - › Describe the experience

18-ene.-25 Curso 2024/2025 página 32

32




Notable Quotes

- "TDD is clean code that works"* – Ron Jeffries
- "The best way that I know to write code is to shape it from the beginning with tests"* – Ron Jeffries
- "The act of writing a unit test is more an act of design than of verification"* – Robert C. Martin
- "The code is the design"* – Martin Fowler
- "Test-Driven Development is a powerful way to produce well designed code with fewer defects"* – Martin Fowler

18-ene.-25 Curso 2024/2025 página 33

33




Notable Quotes

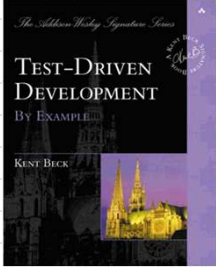
- "Legacy code is code without tests"* – Alan Francis
- "Fewer defects, less debugging, more confidence, better design, and higher productivity in my programming practice"* – Kent Beck
- "Writing test cases before the code takes the same amount of time and effort as writing the test cases after the code, but it shortens the defect-detection-debug cycle"* – Steve McConnell
- "I'm trying to preserve the right for a programmer to think while he's typing. If you feel that it's not going well, you can stop and say 'What did I get wrong? Let me correct it'"* – Ward Cunningham

18-ene.-25 Curso 2024/2025 página 34

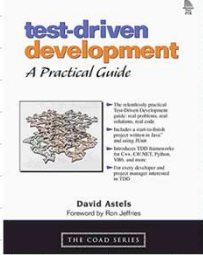
34



Resources (Books)



» Test-Driven Development: By Example
Kent Beck
Addison-Wesley, 2003
ISBN 0-321-14653-0




» Test-Driven Development: A Practical Guide
Dave Astels
Prentice-Hall/Pearson Education, 2003
ISBN 0-13-101649-0
Reviewed BUG developers' magazine,
Nov/Dec 2003

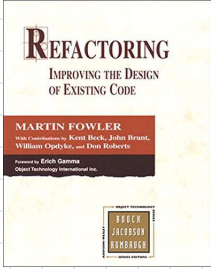
18-ene.-25

Curso 2024/2025

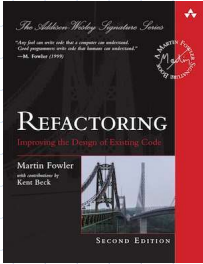
página 35



Resources (Books)



» Refactoring: Improving the Design of Existing Code
Martin Fowler
Addison-Wesley, 1999
ISBN 0-201-48567-2




» Refactoring: Improving the Design of Existing Code (Second Edition)
Martin Fowler & Kent Beck
Addison-Wesley, 2018
ISBN 0-134-75759-9

18-ene.-25

Curso 2024/2025

página 36




Other books

- » Percival, Harry “*Test-Driven Development with Python*” O’Reilly 2014
- » Freeman, Steve & Pryce, Nat. “*Growing Object-Oriented Software, Guided by Tests*” Addison-Wesley 2009
- » McConnell, Steve “*Code Complete 2. A Practical Handbook of Software Construction*” Microsoft Press 2004
- » Binder, R.V. “*Testing Object-Oriented Systems*” Addison-Wesley 1999

18-ene.-25 Curso 2024/2025 página 37

37



Articles

- » Beck, Kent and Gamma, Erich “*Test Infected: Programmers Love Writing Tests*”
<http://members.pingnet.ch/gamma/junit.htm>
- » Martin, Robert C. and Koss, Robert S. “*Test Driven Development of bowling scoring code*”
<http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata>
- » Janzen, David (Simex LLC) and Saiedian, Hossein (University of Kansas) “*Test-Driven Development: Concepts, Taxonomy, and Future Direction*”, IEEE Computer Magazine (Sept, 2005)
<https://www.computer.org/csdl/mags/co/2005/09/r9043-abs.html>
<https://works.bepress.com/djanzen/6/>

18-ene.-25 Curso 2024/2025 página 38

38




Resources (web sites)

- » xUnit implementations
 - https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
 - <http://www.junit.org>
- » Relevant web sites
 - <http://www.testdriven.com>
 - <http://www.refactoring.com>
 - <http://opensourceTesting.org>
- » Mock Objects
 - <http://www.mockobjects.com>
 - <https://www.agilealliance.org/glossary/mocks>
- » testdrivendevelopment group (Yahoo)
 - <https://groups.yahoo.com/neo/groups/testdrivendevelopment>




18-ene.-25 Curso 2024/2025 página 39

39



¿Preguntas?



RED
GREEN
REFACTOR

40