10 Pines

Diseño a la Gorra - Episodio 06

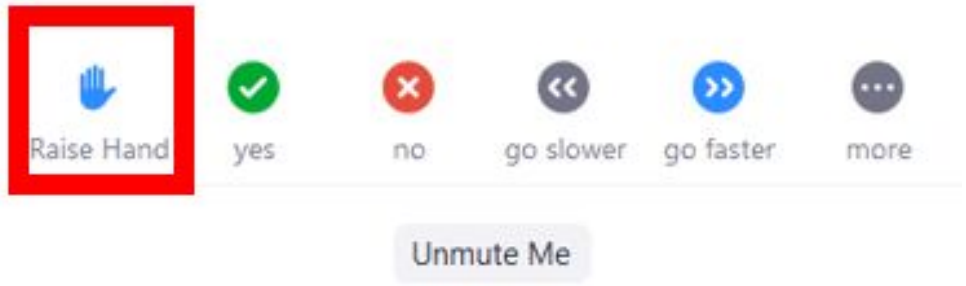
Hernán Wilkinson

hernan.wilkinson@10pines.com

@HernanWilkinson

https://alagorra.10pines.com

Estar muteados a menos que sea necesario



No voy a poder leer el chat

La comunicación visual es importante. Usarla a discreción

Diseño ¡a la gorra!

¡Bienvenidos!
Durante esta serie de Webinars exploraremos *qué* significa **Diseñar Software con Objetos** y *cómo* lo podemos hacer cada vez mejor.
Trataremos muchos temas que irán desde cuestiones filosóficas como qué significa Diseñar en nuestra profesión y dónde está expresado ese Diseño, pasando por consejos y heurísticas para diseñar "mejor" y terminado con ejemplos concretos de cómo aplicar esas heurísticas en la *vida real.*
Los webinars son "*language agnostic*", o sea que no dependen de un lenguaje de programación en particular, aunque los ejemplos que usaremos estarán hechos principalmente en **Java**, **JavaScript**, **Ruby**, **Python** y mi querido **Smalltalk** cuando amerite 😉 .

Te esperamos todos los Martes a las 19 Hrs GMT-3 a partir del Martes 11 de Agosto de 2020.
Para poder participar tenes que registrate acá.

Todo el código y presentaciones estarán disponibles para que lo puedan usar y consultar en cualquier momento acá.

¡Trae ganas de aprender y pasarla bien!

¿Por qué a la Gorra?
Al igual que cuando Diseñamos Software está bueno usar una **Metáfora** para entender qué estamos modelando, en este caso usamos una metáfora para explicar cómo *financiaremos*

Donaciones

mercado pago

$100 - Casi una 🍷 Pagar

$250 - Una buena 🍺 Pagar

$500 - Menos que 🍔 +🍟 Pagar

PayPal Donate

¿Querés donar un monto variado o en otra plataforma?

Dictado por:
Hernán Wilkinson

https://alagorra.10pines.com

Online

Diseño Avanzado de Software con Objetos I

Empieza el **13/10**

Del 13/10 al 22/10. Días: 6 días — [illegible] al 16, 20 y 22 de Octubre — [illegible] GMT-3.

AR$ **15.300-** (IVA incluído)

Hacer una consulta

Inscribirme ahora 15% Dto.

Dictado por:
Máximo Prieto

~~AR$ 18.000-~~
AR$ 15.300-
(IVA incluído)
~~U$D 300-~~
U$D255-
(impuestos incluídos)

Super Early Bird

~~Early Bird~~

¡¡Sorteo 2 Entradas!!

https://academia.10pines.com/courses/96-diseno-avanzado-de-software-con-objetos-i

Online

Test Driven Development Avanzado

Empieza el **02/11**

AR$ **12.750-** (IVA incluído)

Hacer una consulta

Inscribirme ahora 15% Dto.

Del 02/11 al 06/11. Días: Lunes a Viernes — 9:00 a 13:00 GMT-3.

Dictado por:
Hernán Wilkinson

~~AR$ 15.000-~~

AR$ **12.750-**

(IVA incluído)

~~U$D 250-~~

U$D215-

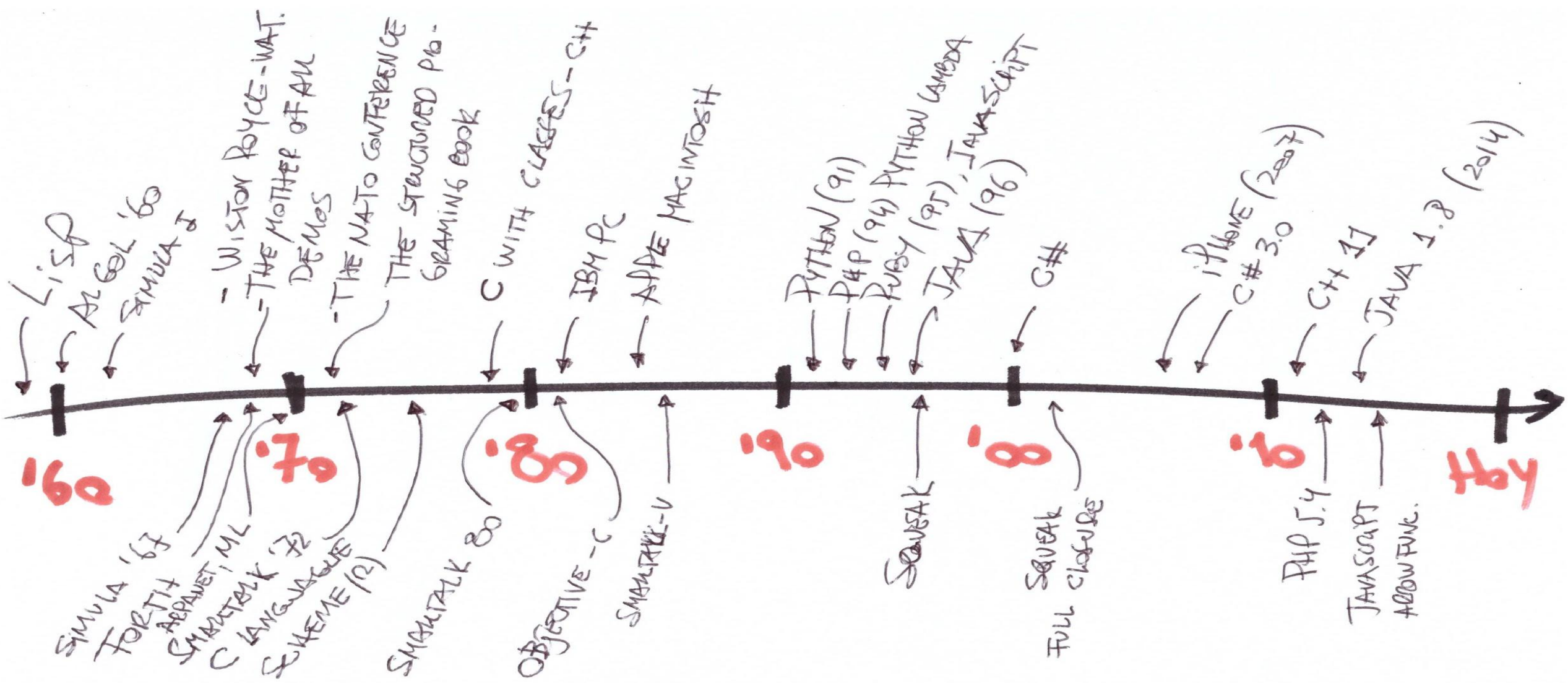
(impuestos incluídos)

Super Early Bird

~~Early Bird~~

https://academia.10pines.com/courses/95-test-driven-development-avanzado

¿Qué vimos en el último episodio?



La Historia de los Closures

¿Cómo se saca el Código Repetido?

1. Mover lo repetido a una nueva abstracción
2. Parametrizar lo que cambia
3. PONERLE NOMBRE
4. Reemplazar código repetido por nueva abstracción

Lisp

PROGRAMS WITH COMMON SENSE

John McCarthy
Computer Science Department
Stanford University
Stanford, CA 94305
jmc@cs.stanford.edu
http://www-formal.stanford.edu/jmc/

1959

Recursive Functions of Symbolic Expressions
and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge, Mass. *

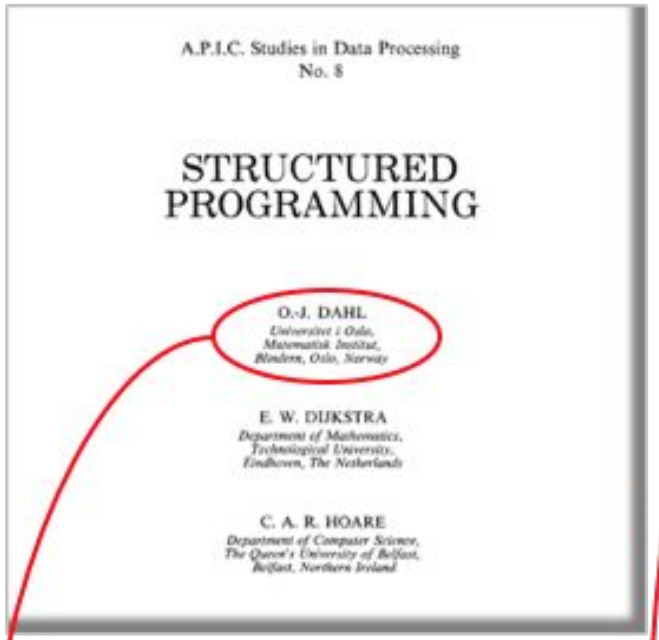
April 1960

Por qué la metodología cascada no funciona, por el mismo W. Royce

!!

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

"The Mother of All Demos"



III. Hierarchical Program Structures

OLE-JOHAN DAHL AND C. A. R. HOARE

1. INTRODUCTION

In this monograph we shall explore certain ways of program structuring and point out their relationship to concept modelling.

We shall make use of the programming language SIMULA 67 with particular emphasis on structuring mechanisms. SIMULA 67 is based on ALGOL 60 and contains a slightly restricted and modified version of ALGOL 60 as a subset. Additional language features are motivated and explained informally when introduced. The student should have a good knowledge of ALGOL 60 and preferably be acquainted with list processing techniques.

For a full exposition of the SIMULA language we refer to the "Simula 67 Common Base Language" [2]. Some of the linguistic mechanisms introduced in the monograph are currently outside the "Common Base"*.

The monograph is an extension and reworking of a series of lectures given by Dahl at the NATO Summer School on Programming, Marktoberdorf 1970. Some of the added material is based on programming examples that have occurred elsewhere [3, 4, 5].


System Transcript
11:41:21 am)
Snapshot at: (8
September 2020
11:42:29 am)
Snapshot at: (8
September 2020
11:42:43 am)
System Browser
Interface-Changes
System-Support
System-Changes
System-Compiler
System-Releasing
Files-Streams
Files-Abstract
Files-Xerox Alto
Files-Posix
Ejemplo
PosixFile
PosixFileD
PosixFileP
Instance
File subclass: #PosixFile
instanceVariableNames: 'fc
classVariableNames: ''
poolDictionaries: ''
category: 'Files-Posix'
Workspace
| factorial |
factorial ← [:n |
n = 1
ifTrue: [1]
ifFalse: [(factorial value: n - 1)* n]].
factorial value: 1.
1 odd
ifTrue: ['Es Impar']
ifFalse: ['Es par'] .
[10 factorial] value
[10 factorial] inspect
[:n | n factorial] value: 100
System Browser
Numeric-Magnitudes
Float
testing
System Workspace
Undeclared inspect.

Scope Dinámico vs. Scope Lexicográfico

AI Memo No. 353

March 10, 1976

LAMBDA

THE ULTIMATE IMPERATIVE

AI Memo 379

November 1976

LAMBDA

THE ULTIMATE DECLARATIVE

AI Memo 443

October 1977

DEBUNKING THE "EXPENSIVE PROCEDURE CALL" MYTH

or, PROCEDURE CALL IMPLEMENTATIONS CONSIDERED HARMFUL

or, LAMBDA: THE ULTIMATE GOTO

AI Memo No. 514

March 1979

Design of LISP-Based Processors
or, SCHEME: A Dielectric LISP
or, Finite Memories Considered Harmful
or, LAMBDA: The Ultimate Opcode

by

Full Closure

```
closureWithExplicitReturn
"
self new closureWithExplicitReturn
"

	| closure |

	closure := [ ↑10 ].

	↑closure value + 5
```

Las limitaciones de Java

```
public class Ejemplo {

    public void m1(){
        int counter;
        counter = 0;

        Runnable adder = ()-> counter = counter + 1;

    }
}
```

Los Lenguajes de Programación representan el estado de conocimiento de aquellos que los crearon

...

como cualquier modelo

Smalltalk

- Simplificación: Todo “bloque de código” es un Closure

closureAsBlockInIfTrue

```
1 odd
        ifTrue: [ 'es impar' ]
        ifFalse: [ 'es par' ]
```

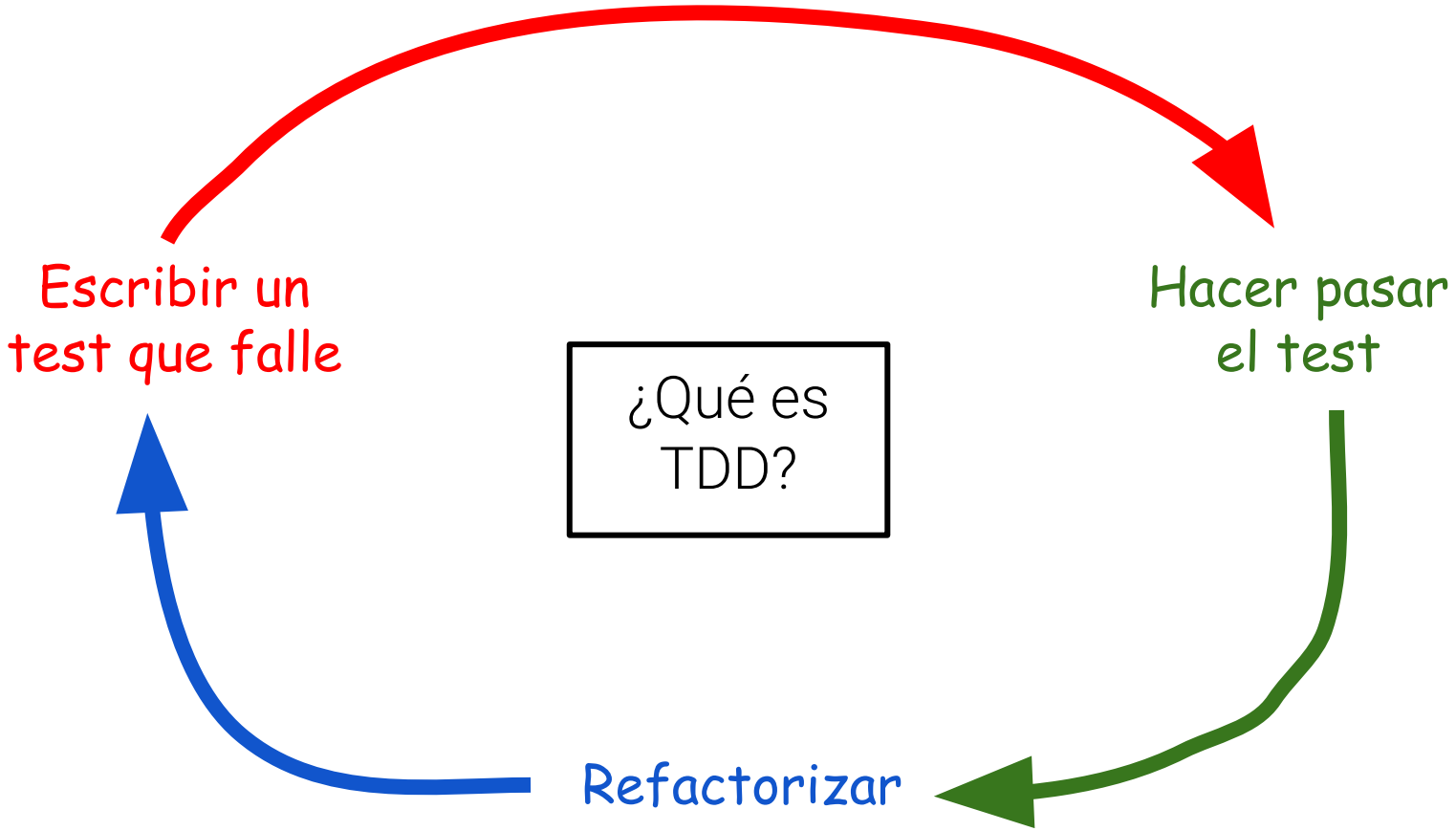
closureAsBlockInIfTrue

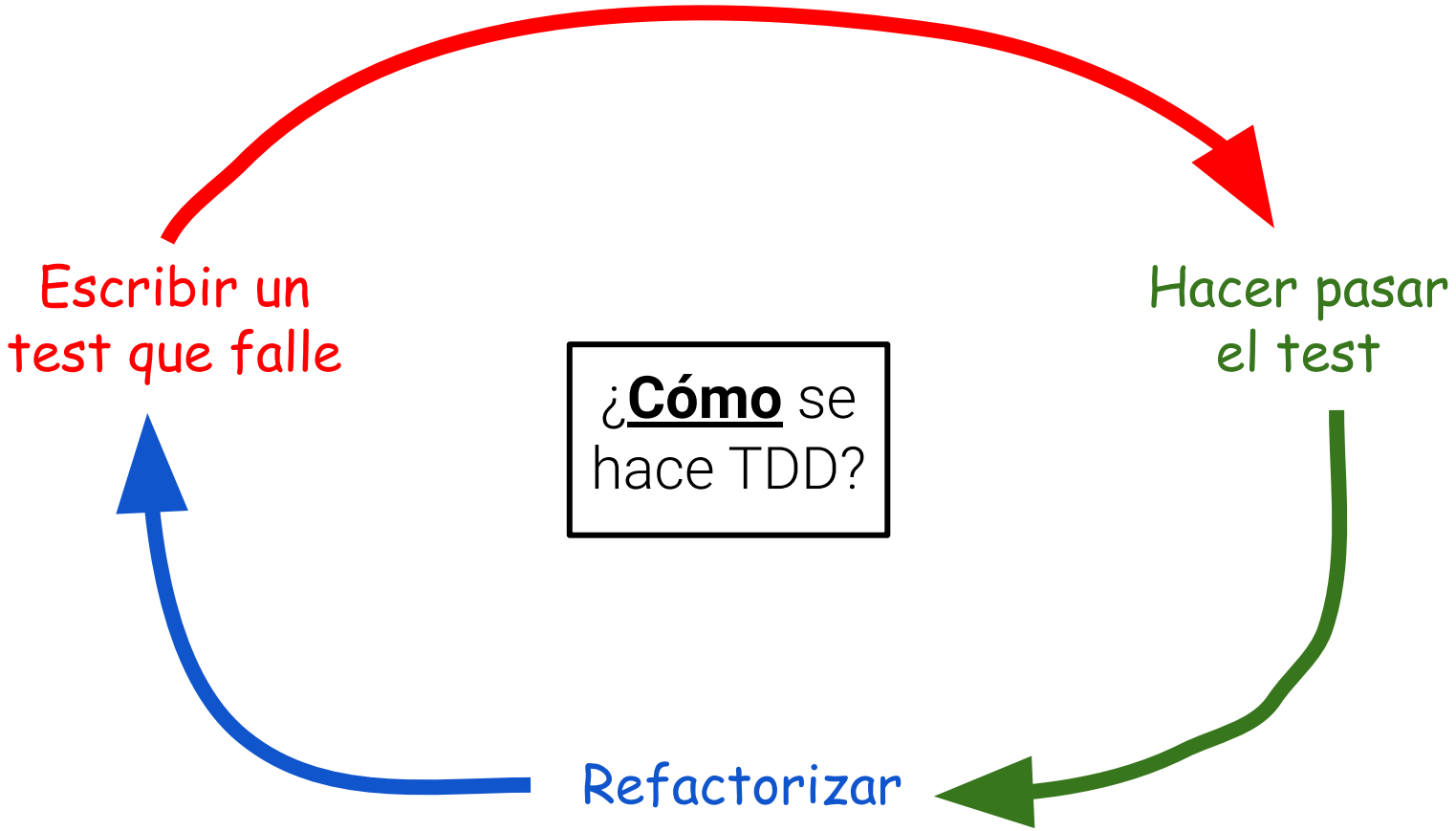
```
| trueBlock falseBlock |

trueBlock := [ 'es impar' ].
falseBlock := [ 'es par' ].

1 odd
        ifTrue: trueBlock
        ifFalse: falseBlock
```

TDD


Escribir un
test que falle
Hacer pasar
el test
¿Qué es
TDD?
Refactorizar


Escribir un test que falle
Hacer pasar el test
¿Cómo se hace TDD?
Refactorizar

Uncle Bob Martin
@unclebobmartin

OO is not a mindset, nor a philosophy, nor a technique for modeling "the real world". It is nothing more, nor less, than data structures manipulated by associated functions that are called indirectly through vectors. Deal with it.

Uncle Bob Martin @unclebobmartin · Sep 8

OO is not a mindset, nor a philosophy, nor a technique for modeling "the real world". It is nothing more, nor less, than data structures manipulated by associated functions that are called indirectly through vectors. Deal with it.

61 169 1K

Hernan Wilkinson
@HernanWilkinson

Replying to @unclebobmartin

This is the same as saying that poetry is nothing more than a bunch of letters with a certain rhyme and writing is nothing more than a bunch of lines drawn in a certain order and so on. This type of reductionism ignores that the hole is more than the sum of its parts and ... 1/n

Hernan Wilkinson @HernanWilkinson · Sep 13

This is the same as saying that poetry is nothing more than a bunch of letters with a certain rhyme and writing is nothing more than a bunch of lines drawn in a certain order and so on. This type of reductionism ignores that the hole is more than the sum of its parts and ... 1/n

2 6 31

Hernan Wilkinson
@HernanWilkinson

forgets the meaning created by new layers of abstraction, keeping everything at the implementation level... so yes, at the end they are all bits and bytes, but we can do much more with them when we give them meaning
(I meant whole not hole in the previous tweet)

Hernan Wilkinson @HernanWilkinson · Sep 13

This is the same as saying that poetry is nothing more than a bunch of letters with a certain rhyme and writing is nothing more than a bunch of lines drawn in a certain order and so on. This type of reductionism ignores that the hole is more than the sum of its parts and ... 1/n

2 6 31

Uncle Bob Martin
@unclebobmartin

Replying to @HernanWilkinson

There is certainly a beauty in well written code. I might even agree that it is a kind of poetry. Proper use of OO is necessary to the structure of that poetry. But that does not change what OO is.

TDD es mucho más que tests que se escriben primero y que hay que hacer pasar después

¿Qué es TDD?

- Técnica de desarrollo basada en características del Aprendizaje
  - Iterativa e Incremental
  - Basada en Feedback Inmediato
- Side-effect:
  - Recuerda todo lo aprendido
  - Y permite asegurarnos de no haber "desaprendido"
- Incluye análisis, diseño, programación y testing

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

10 Pines

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más s[illegible]se nos ocurra
   - Debe fallar al correrlo

¿POR QUÉ?

2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

¿POR QUÉ?

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta "todos los tests" pasen

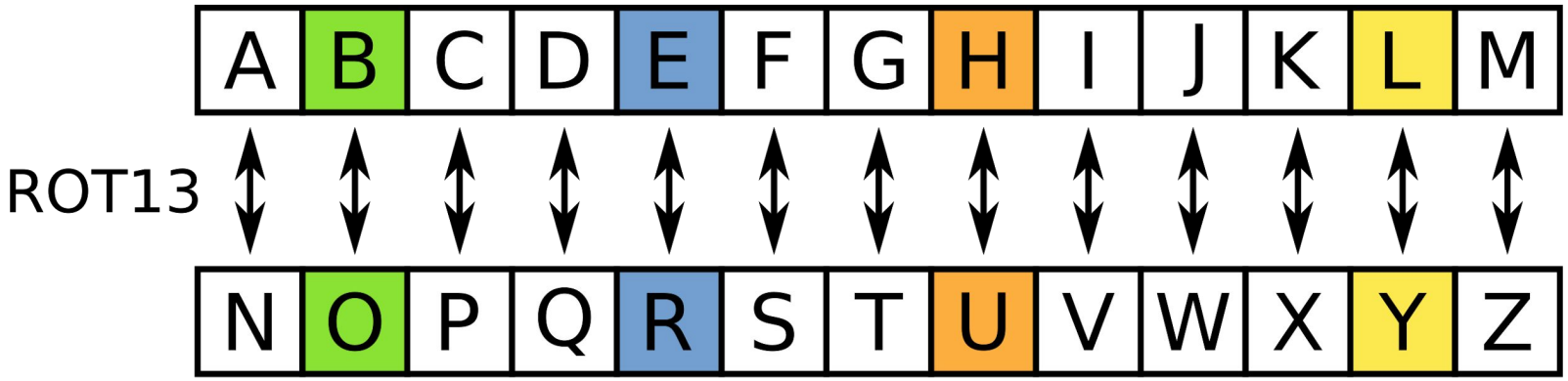
¿POR QUÉ?

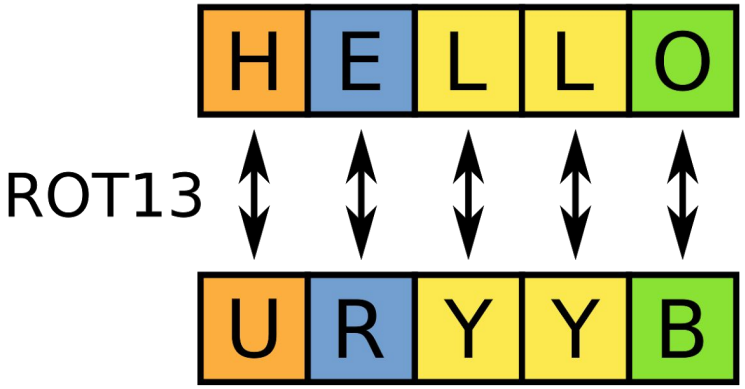
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

10 Pines

Ejemplo

Rot 13





ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Algunas conclusiones sobre TDD

El tiempo que tardo en cada paso de TDD, es un indicio de qué tan bien estoy realizando la técnica

Los test unitarios están “acoplados” al diseño

Puedo verificar qué tan bien hice los test usando mutation testing

(por lo menos de manera manual)

TDD hace que los programadores sean
los primeros usuarios
del sistema que desarrollan

Cuando nos sentimos así

Cuando nos sentimos así

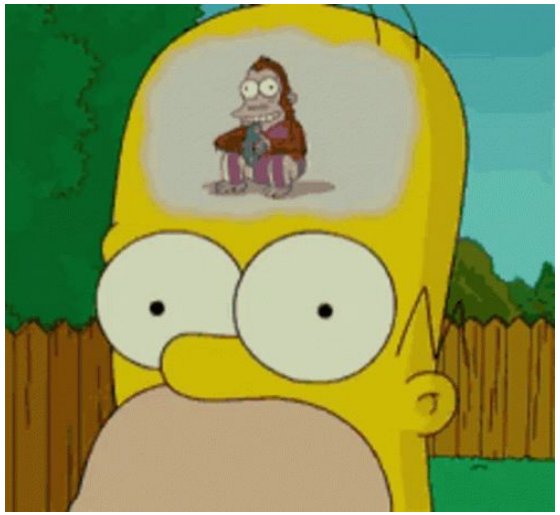
La culpa no es de TDD…

¡Estamos “sufriendo” nuestros diseños!

Un programador no es buen programador si no sabe testear

TDD jerarquizó el testing

TDD hace explícito todos los "tests" que corremos en nuestra cabeza



TDD nos da seguridad al momento de refactorizar

→

Evita que el sistema se convierta en un "sistema legacy"

TDD no implica buen diseño

Algunas conclusiones de Diseño

Es importantísimo poder extender clases para no caer en *“soluciones estructuradas”*

Extender clases nos permite que sus instancias cumplan distintos roles según el contexto

¡Hay mucho más por discutir sobre TDD y Diseño!

Preguntas

Semana que viene no hay episodio 😢😢😢

Online

Diseño Avanzado de Software con Objetos I

Empieza el **13/10**

Del 13/10 al 22/10. Días: 6 días — [illegible] al 16, 20 y 22 de Octubre — [illegible] GMT-3.

AR$ **15.300-** (IVA incluído)

Hacer una consulta

Inscribirme ahora 15% Dto.

Dictado por:
Máximo Prieto

~~AR$ 18.000-~~
AR$ 15.300-
(IVA incluído)
~~U$D 300-~~
U$D255-
(impuestos incluídos)

Super Early Bird

~~Early Bird~~

¡¡Sorteo 2 Entradas!!

https://academia.10pines.com/courses/96-diseno-avanzado-de-software-con-objetos-i

Diseño ¡a la gorra!

¡Bienvenidos!
Durante esta serie de Webinars exploraremos *qué* significa **Diseñar Software con Objetos** y *cómo* lo podemos hacer cada vez mejor.
Trataremos muchos temas que irán desde cuestiones filosóficas como qué significa Diseñar en nuestra profesión y dónde está expresado ese Diseño, pasando por consejos y heurísticas para diseñar "mejor" y terminado con ejemplos concretos de cómo aplicar esas heurísticas en la *vida real.*
Los webinars son "*language agnostic*", o sea que no dependen de un lenguaje de programación en particular, aunque los ejemplos que usaremos estarán hechos principalmente en **Java**, **JavaScript**, **Ruby**, **Python** y mi querido **Smalltalk** cuando amerite 😉 .

Te esperamos todos los Martes a las 19 Hrs GMT-3 a partir del Martes 11 de Agosto de 2020.
Para poder participar tenes que registrate acá.

Todo el código y presentaciones estarán disponibles para que lo puedan usar y consultar en cualquier momento acá.

¡Trae ganas de aprender y pasarla bien!

¿Por qué a la Gorra?
Al igual que cuando Diseñamos Software está bueno usar una **Metáfora** para entender qué estamos modelando, en este caso usamos una metáfora para explicar cómo *financiaremos*

Donaciones

mercado pago

$100 - Casi una 🍷 Pagar

$250 - Una buena 🍺 Pagar

$500 - Menos que 🍔 +🍟 Pagar

PayPal Donate

¿Querés donar un monto variado o en otra plataforma?

Dictado por:
Hernán Wilkinson

https://alagorra.10pines.com

Muchas gracias

10 Pines

Creative Software Development

10pines.com

info@10pines.com

+54 (011) 6091-3125 / 4893-2057

Av. Leandro N. Alem 896 6° - Bs. As. - Argentina

@10pines