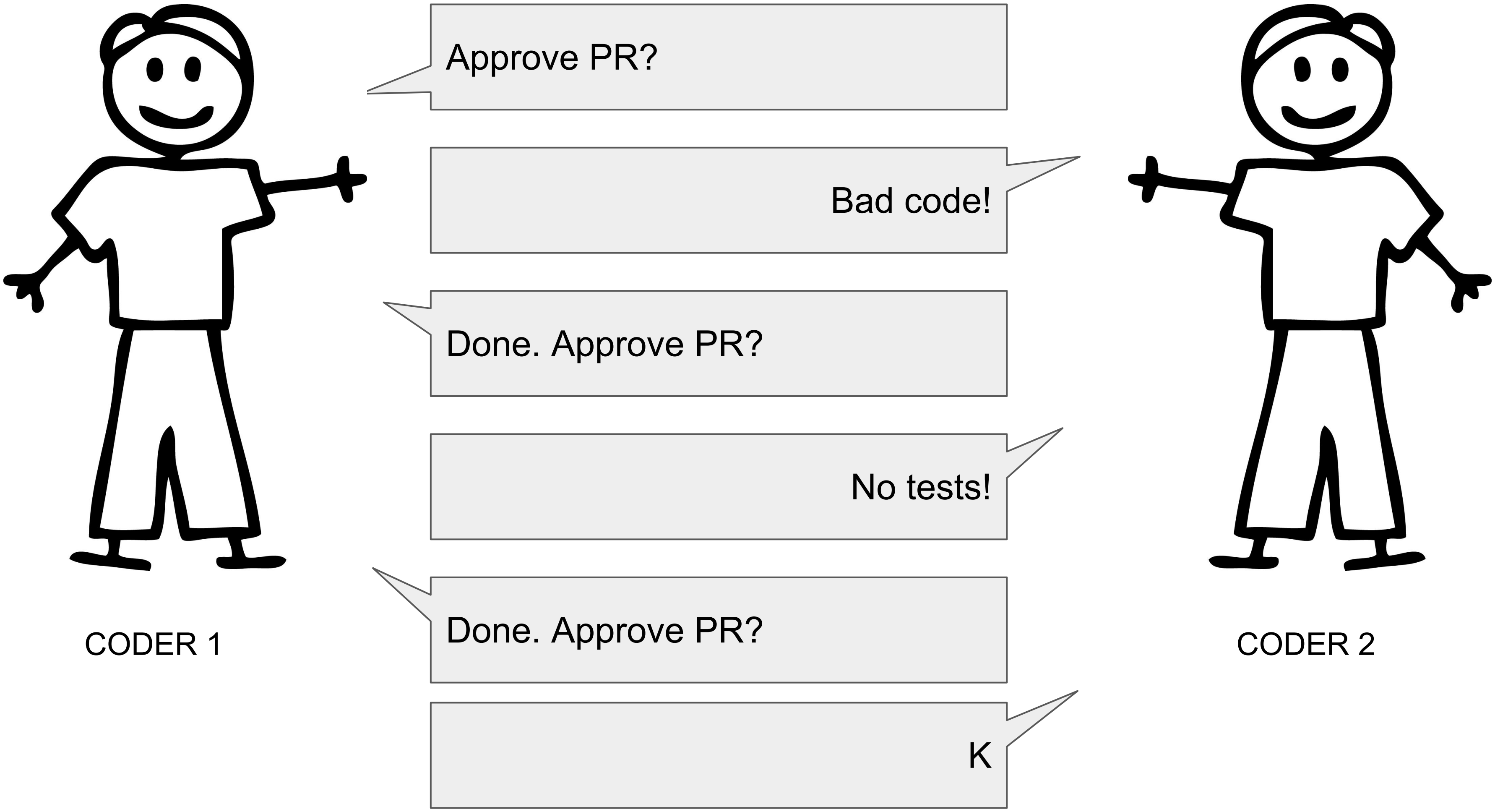
How coding for testability made me a better developer

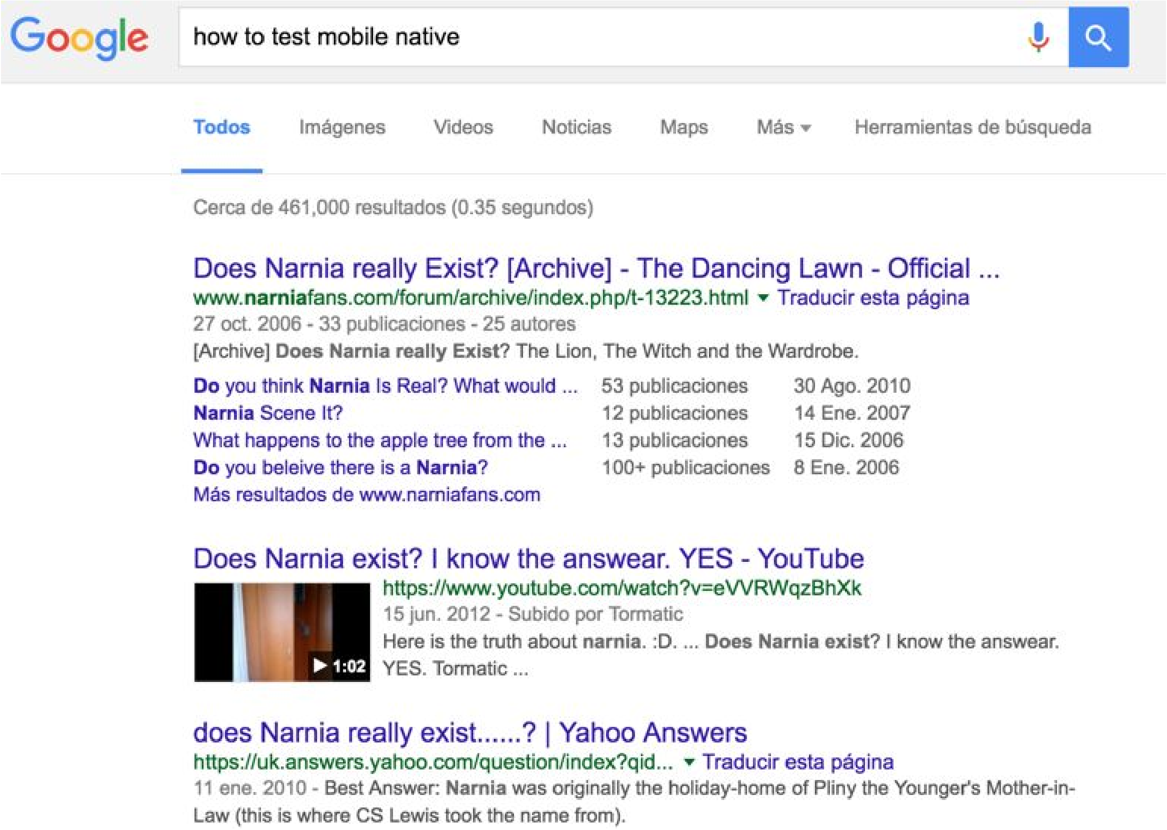
A brief story about web development well done



So I did some research

how to test mobile native

So I did some research



how to test mobile native

Cerca de 461,000 resultados (0.35 segundos)

Does Narnia really Exist? [Archive] - The Dancing Lawn - Official ...
www.**narnia**fans.com/forum/archive/index.php/t-13223.html ▾ Traducir esta página
27 oct. 2006 - 33 publicaciones - 25 autores
[Archive] **Does Narnia really Exist**? The Lion, The Witch and the Wardrobe.

Do you think **Narnia** Is Real? What would ...	53 publicaciones	30 Ago. 2010
Narnia Scene It?	12 publicaciones	14 Ene. 2007
What happens to the apple tree from the ...	13 publicaciones	15 Dic. 2006
Do you beleive there is a **Narnia**?	100+ publicaciones	8 Ene. 2006

Más resultados de www.narniafans.com

Does Narnia exist? I know the answear. YES - YouTube
https://www.youtube.com/watch?v=eVVRWqzBhXk
15 jun. 2012 - Subido por Tormatic
Here is the truth about **narnia**. :D. ... **Does Narnia exist?** I know the answear. YES. Tormatic ...

does Narnia really exist......? | Yahoo Answers
https://uk.answers.yahoo.com/question/index?qid... ▾ Traducir esta página
11 ene. 2010 - Best Answer: **Narnia** was originally the holiday-home of Pliny the Younger's Mother-in-Law (this is where CS Lewis took the name from).

Agenda

- ➔ Introduction √
- ➔ About me
- ➔ Wasteful and helpful tests (6 min)
- ➔ A short **TDD** implementation (6 min)
- ➔ Where tests go in **MVVM** (6 min)
- ➔ **Application Events** (10 min)
- ➔ Recap (4 min)

About me

Last 3 years working in Restorando

Before that, rewrote AMC Theatres app.

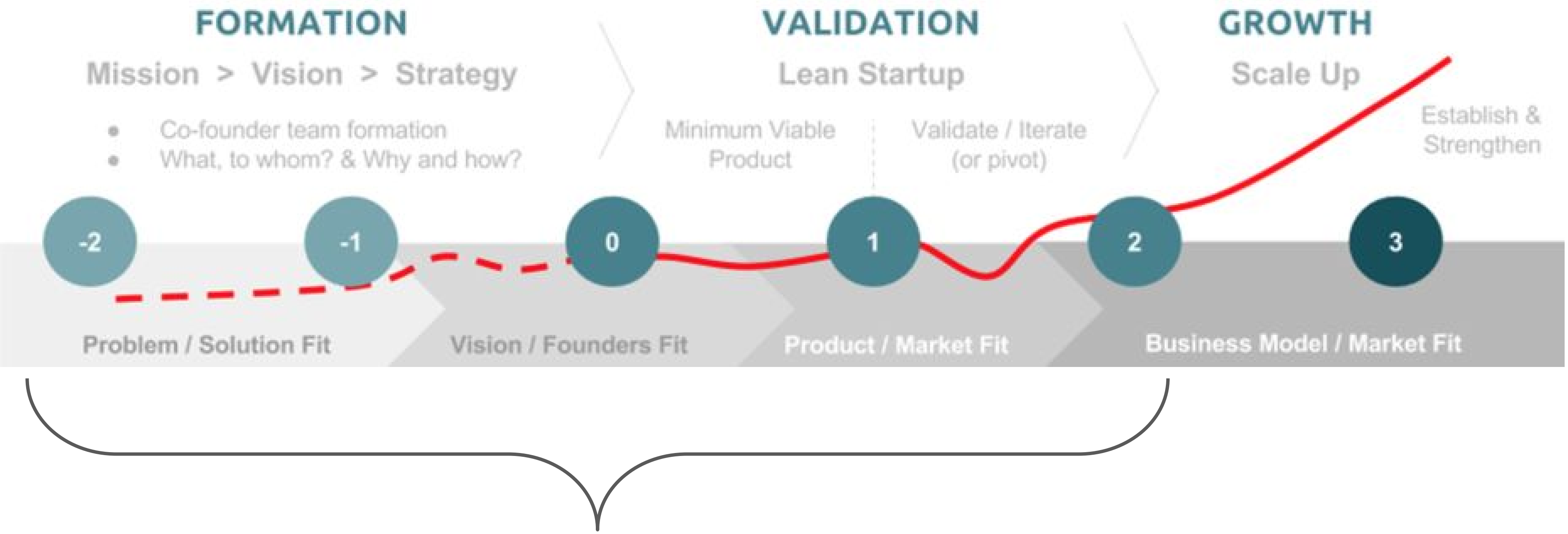
Before that, a gazillion MVPs in software factories

Teaching assistant in UTN FRBA

Amateur guitar player

Lucas Vidal
LucasVidal

Where this talk applies



Here

How did I started testing

- At my university one of my programming classes required to present a work with Unit Tests as an outcome.
- At Restorando, every web dev did it
- When I released my first Pod, my boss (luckily) forced me to have it properly tested as a company rule.
- WWDC and Google I/O started to speak openly about it.

Why I did not do it in the first place

Not enough time.

Just one developer per project.

Too many iterations.

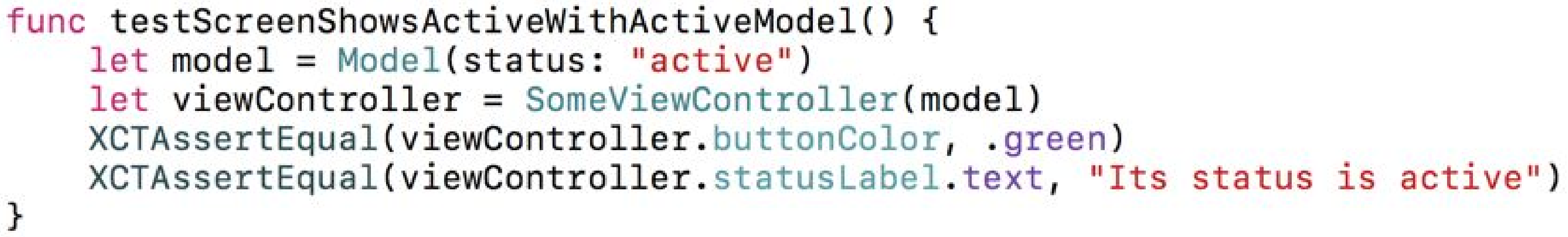
Native is way more complex to test due to all its features and concurrencies.

And our client app was just a presentation client.

Wasteful tests

Presentation client app have very low business logic

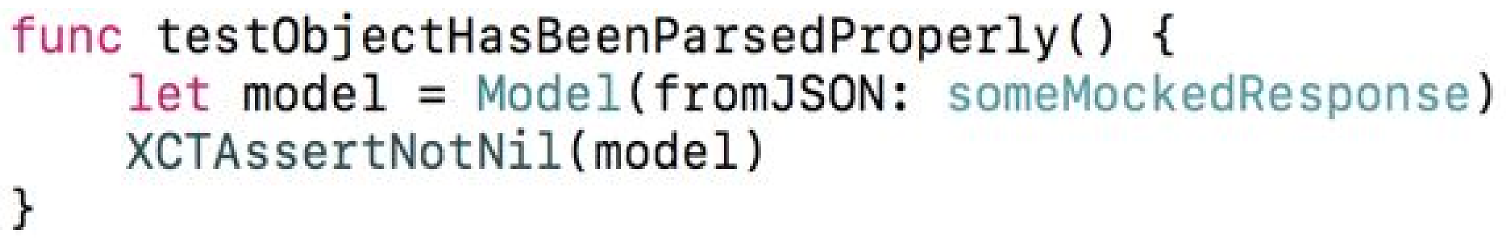
If I had done a test back then, it would have looked kind of like this...



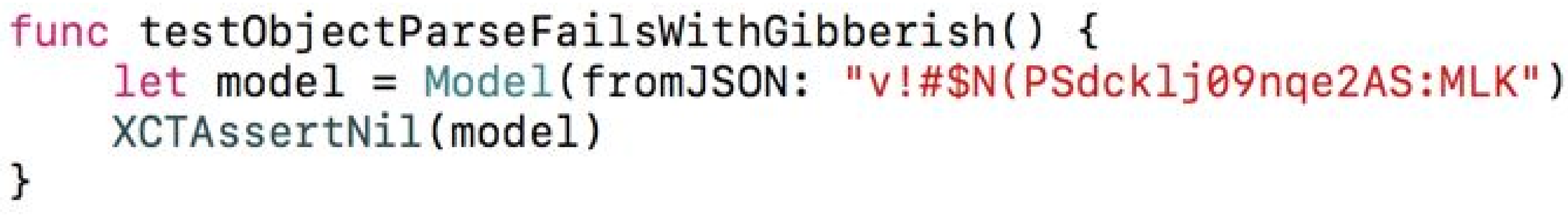
```swift
func testScreenShowsActiveWithActiveModel() {
    let model = Model(status: "active")
    let viewController = SomeViewController(model)
    XCTAssertEqual(viewController.buttonColor, .green)
    XCTAssertEqual(viewController.statusLabel.text, "Its status is active")
}
```

Presentation client app have very low business logic

Or like this...



```
func testObjectHasBeenParsedProperly() {
    let model = Model(fromJSON: someMockedResponse)
    XCTAssertNotNil(model)
}
```



```
func testObjectParseFailsWithGibberish() {
    let model = Model(fromJSON: "v!#$N(PSdcklj09nqe2AS:MLK")
    XCTAssertNil(model)
}
```

It’s a fact that

Teams will change

Rules will change

Code will be mostly rewritten once or twice a year

Metrics over unit tests

Take away #1

Some application layers change frequently.

Don’t test dummy things, nor things that will change more than what you can maintain.

Helpful tests

Tests are the best documentation

You can be **100% sure** that what the test says it happens, happens (or fails)

You have to be **really explicit about the rules** you are setting, any other should understand it in the code review

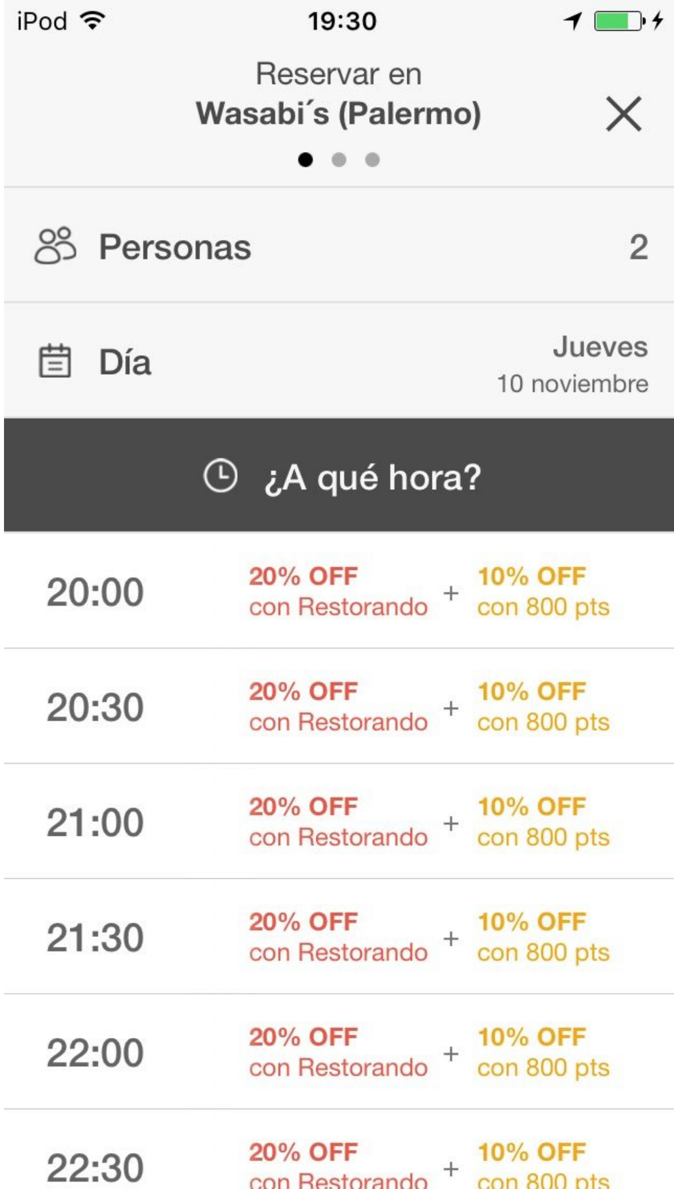
You are leaving for future self and newcomers **the reasons why something happens**.

There was a special place in my app

1. Show me all the availability broken down by time
2. For the same time there might different slots
3. For each slot, sort them by discount %
4. Also include extra discounts if logged in onto our loyalty program.
5. Just put the better discount along with the extra discount if logged in to our loyalty program.
6. Ok: this is the rule: Voucher, Best discount, Best extra discount.

6 a. ... But only two of them

6 b. … In that order



iPod 19:30

Reservar en
Wasabi´s (Palermo)

Personas 2

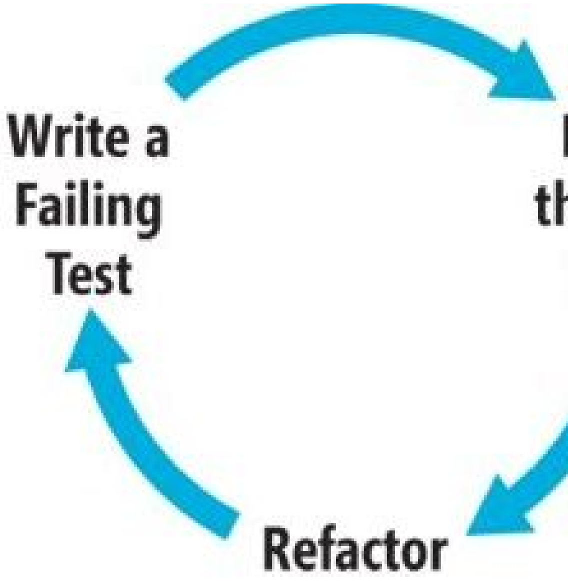
Día Jueves
10 noviembre

¿A qué hora?

20:00	20% OFF con Restorando	+	10% OFF con 800 pts
20:30	20% OFF con Restorando	+	10% OFF con 800 pts
21:00	20% OFF con Restorando	+	10% OFF con 800 pts
21:30	20% OFF con Restorando	+	10% OFF con 800 pts
22:00	20% OFF con Restorando	+	10% OFF con 800 pts
22:30	20% OFF con Restorando	+	10% OFF con 800 pts

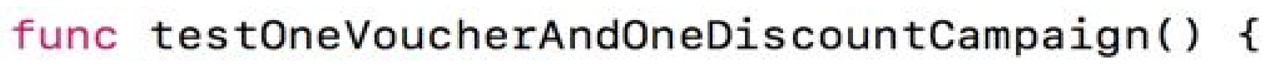
TDD

TDD 101



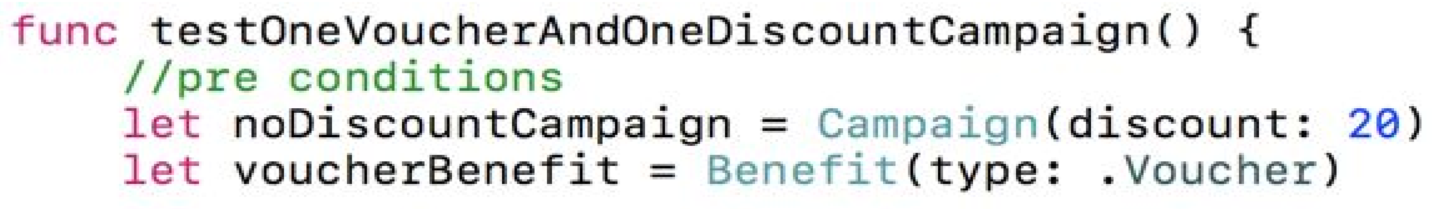
In practise, there's no need to have a clear distinction between these two

Example of a real TDD



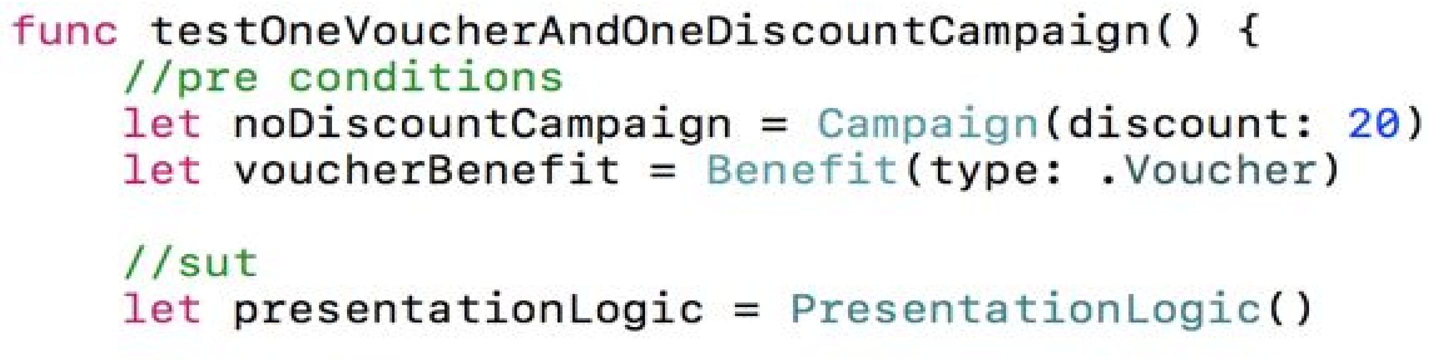
```
func testOneVoucherAndOneDiscountCampaign() {
```

Example of a real TDD



```
func testOneVoucherAndOneDiscountCampaign() {
    //pre conditions
    let noDiscountCampaign = Campaign(discount: 20)
    let voucherBenefit = Benefit(type: .Voucher)
```

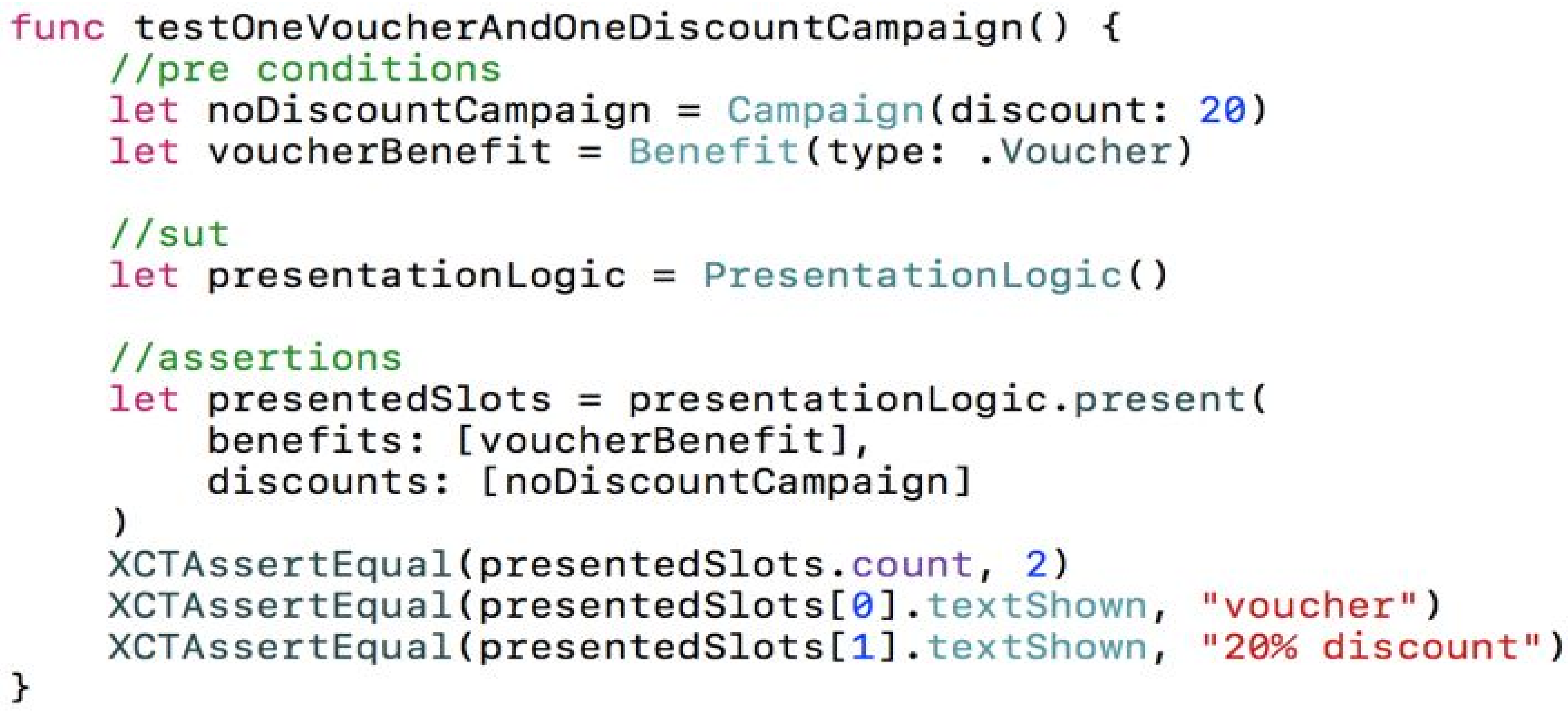
Example of a real TDD



```swift
func testOneVoucherAndOneDiscountCampaign() {
    //pre conditions
    let noDiscountCampaign = Campaign(discount: 20)
    let voucherBenefit = Benefit(type: .Voucher)

    //sut
    let presentationLogic = PresentationLogic()
```

Example of a real TDD



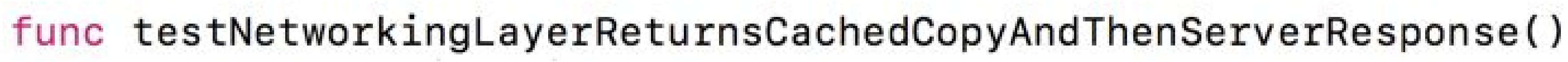
```
func testOneVoucherAndOneDiscountCampaign() {
    //pre conditions
    let noDiscountCampaign = Campaign(discount: 20)
    let voucherBenefit = Benefit(type: .Voucher)

    //sut
    let presentationLogic = PresentationLogic()

    //assertions
    let presentedSlots = presentationLogic.present(
        benefits: [voucherBenefit],
        discounts: [noDiscountCampaign]
    )
    XCTAssertEqual(presentedSlots.count, 2)
    XCTAssertEqual(presentedSlots[0].textShown, "voucher")
    XCTAssertEqual(presentedSlots[1].textShown, "20% discount")
}
```

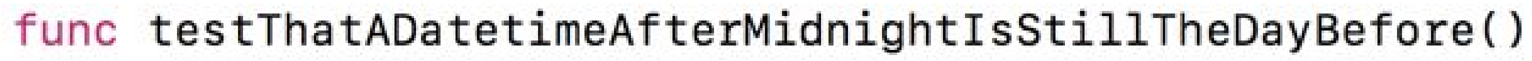
Other TDD use cases

Networking layer



```
func testNetworkingLayerReturnsCachedCopyAndThenServerResponse()
```

Weird date operations



```
func testThatADatetimeAfterMidnightIsStillTheDayBefore()
```

TDD is Dead!

- DHH
- Martin Fowler
- Kent Beck

MUST View!

http://martinfowler.com/articles/is-tdd-dead/

Cases of bad TDD

Test-induced design damage

→ It can lead to approaches such as hexagonal rails, that is design damage due to the complexity of excessive indirection.

You can do too much testing!

→ There is a problem with teams valuing tests more than they value the functional code

Take away #2

Write test so that you can understand complex tasks better, while leaving documented code.

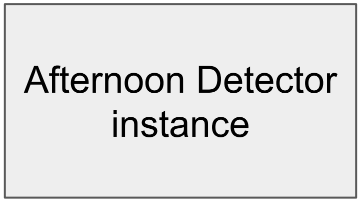
Don’t TDD everything, don’t test everything. It can be counterproductive and lead to excessively complex designs.

Dependencies

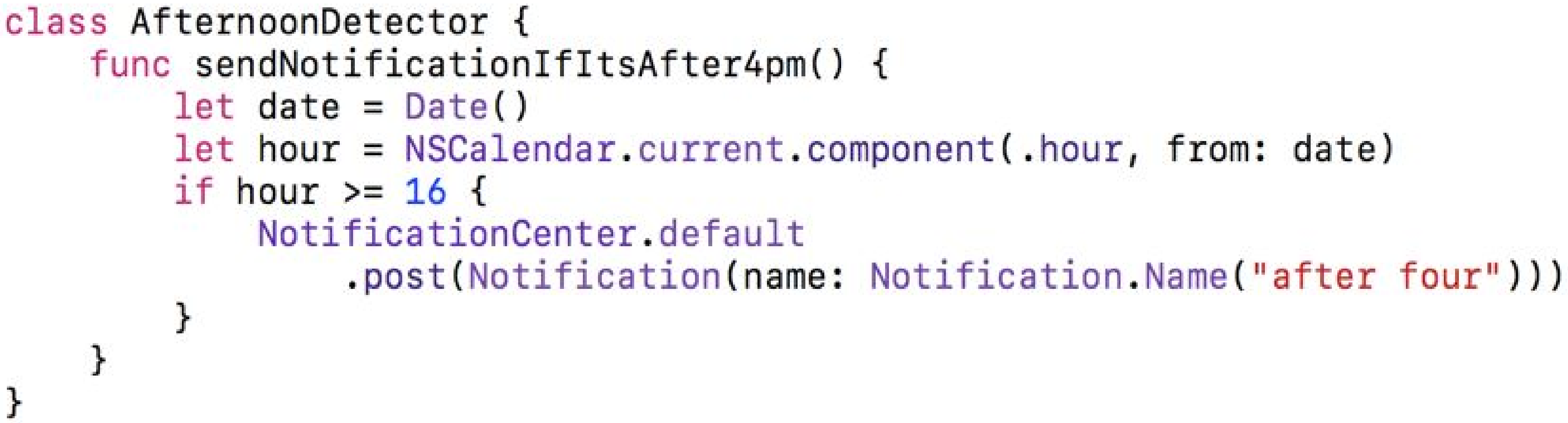
How coupled your code can be

(and you didn’t even noticed)

How coupled your code can be

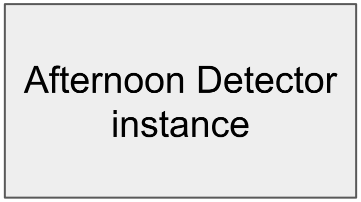


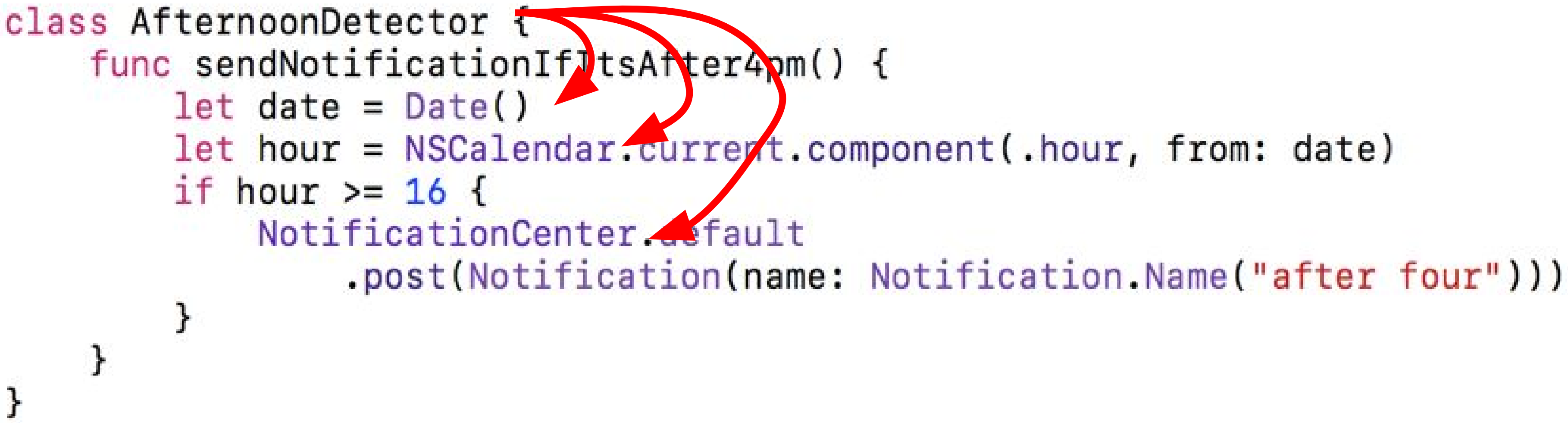
Afternoon Detector instance



```
class AfternoonDetector {
    func sendNotificationIfItsAfter4pm() {
        let date = Date()
        let hour = NSCalendar.current.component(.hour, from: date)
        if hour >= 16 {
            NotificationCenter.default
                .post(Notification(name: Notification.Name("after four")))
        }
    }
}
```

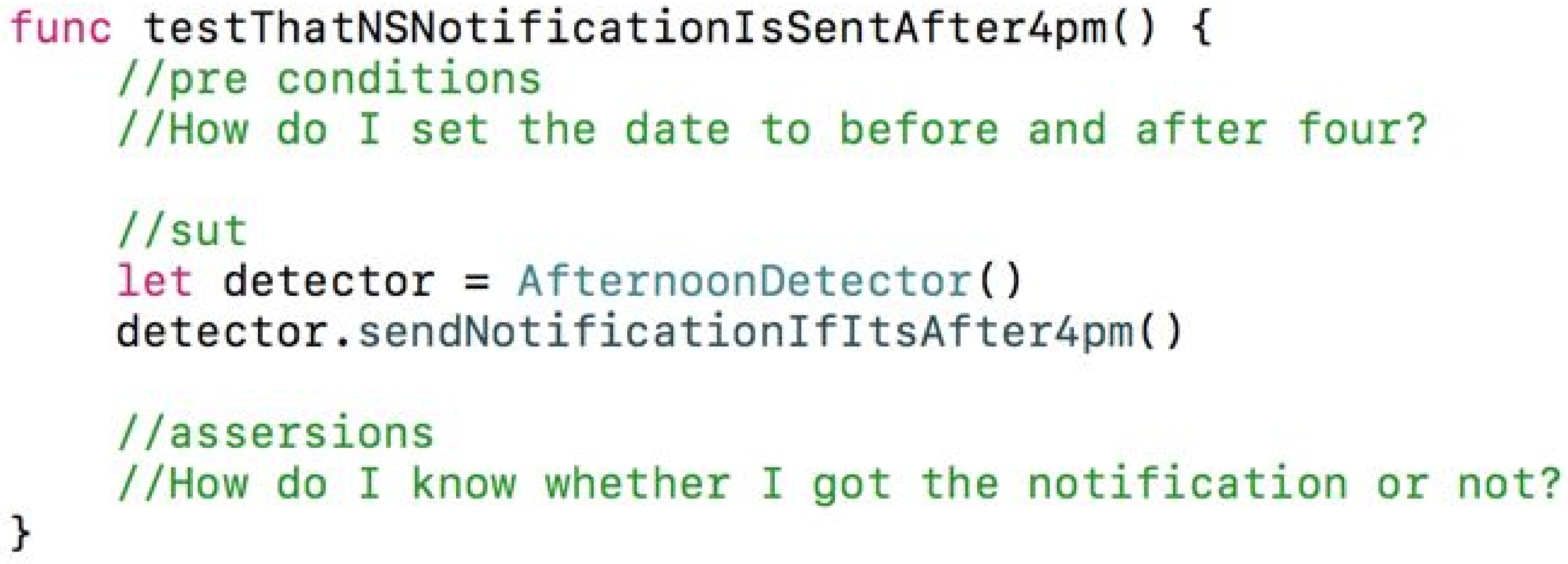
How coupled your code can be





```
class AfternoonDetector {
    func sendNotificationIfItsAfter4pm() {
        let date = Date()
        let hour = NSCalendar.current.component(.hour, from: date)
        if hour >= 16 {
            NotificationCenter.default
                .post(Notification(name: Notification.Name("after four")))
        }
    }
}
```

How would you test it?



```
func testThatNSNotificationIsSentAfter4pm() {
    //pre conditions
    //How do I set the date to before and after four?

    //sut
    let detector = AfternoonDetector()
    detector.sendNotificationIfItsAfter4pm()

    //assersions
    //How do I know whether I got the notification or not?
}
```

You need to let others know
who you depend from

Dependencies

IoC: inversion of control *(Or Hollywood Principle: Don't call us, we'll call you)*

In one line:

I’ll **provide** to you what **you say you need**

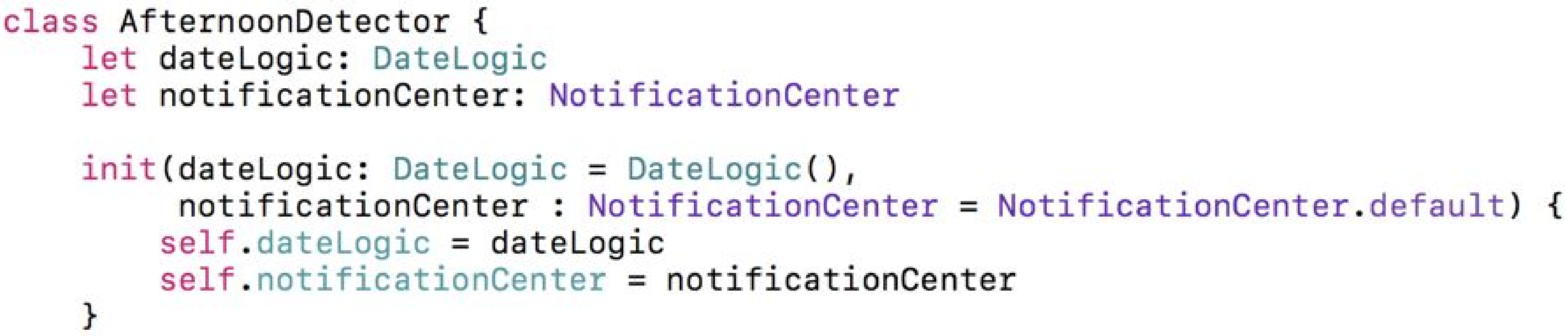
so that you can work **as I expect**

Dependencies

Clarify which are the layers that the system under test will be and which will not

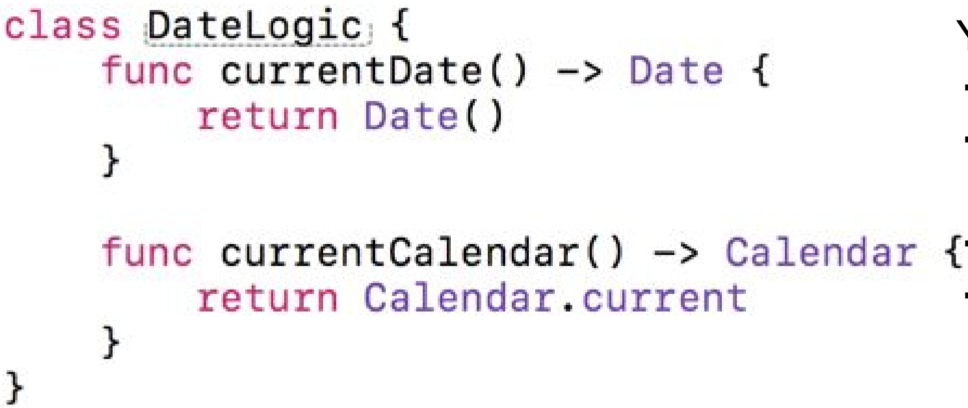
- Networking
- Parse
- APIs and third party libraries
- Business logic
- Events

Inject all that we expect not to fail for any given test.



```
class AfternoonDetector {
    let dateLogic: DateLogic
    let notificationCenter: NotificationCenter

    init(dateLogic: DateLogic = DateLogic(),
         notificationCenter : NotificationCenter = NotificationCenter.default) {
        self.dateLogic = dateLogic
        self.notificationCenter = notificationCenter
    }
```



```
class DateLogic {
    func currentDate() -> Date {
        return Date()
    }

    func currentCalendar() -> Calendar {
        return Calendar.current
    }
}
```

You can:

- ➔ Subclass
- ➔ Provide a mocked instance sharing a protocol
- ➔ Perform assertions on it
- ➔ ... or pretty much do whatever you want with this dependency now.

It’s yours

You don’t need a dependency injection framework.

But here’s the one I use.

Swinject

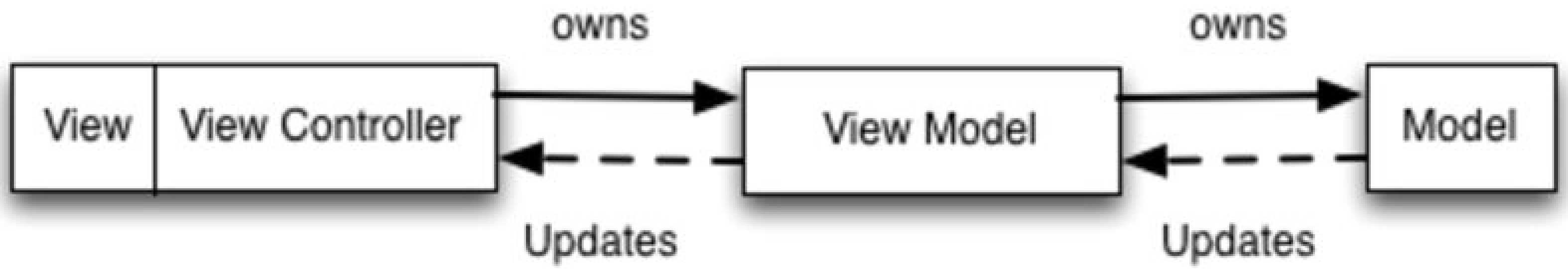
https://github.com/Swinject/Swinject

Take away #3

Be explicit on what a class uses so everyone can understand what happens under the hood.

By extracting your dependencies you can create tests that **only test** what you want to test.

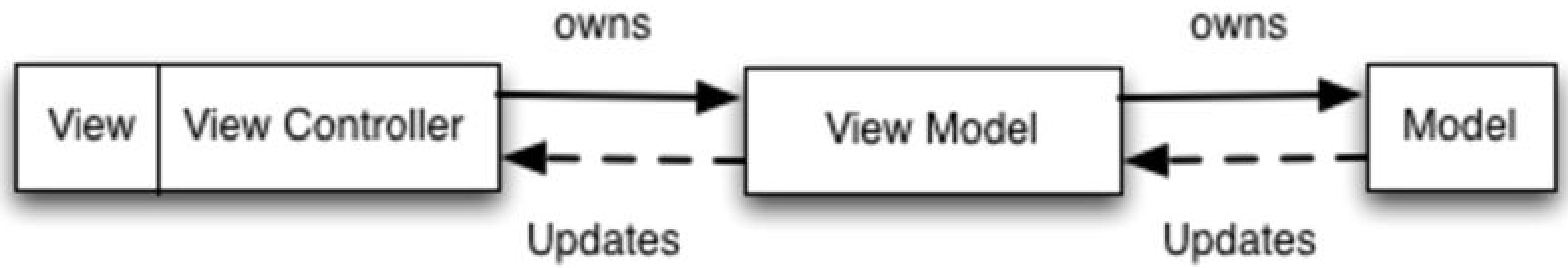
MVVM in 5 seconds



Data Binding is HIGHLY recommendable
You should learn reactive programming!!!

https://github.com/ReactiveX/RxSwift

MVVM as an easier way to handle the SUT



Don’t waste your time testing this!!!

Presentation Logic

Business logic
Model, Utils &
Helpers, etc.

Why MVVM?

Model

Data container
Some business rules

ViewController

Retrieve data from server
Transform response into business objects
Presentation rules
Business rules
…
...

View

UI Hierarchy

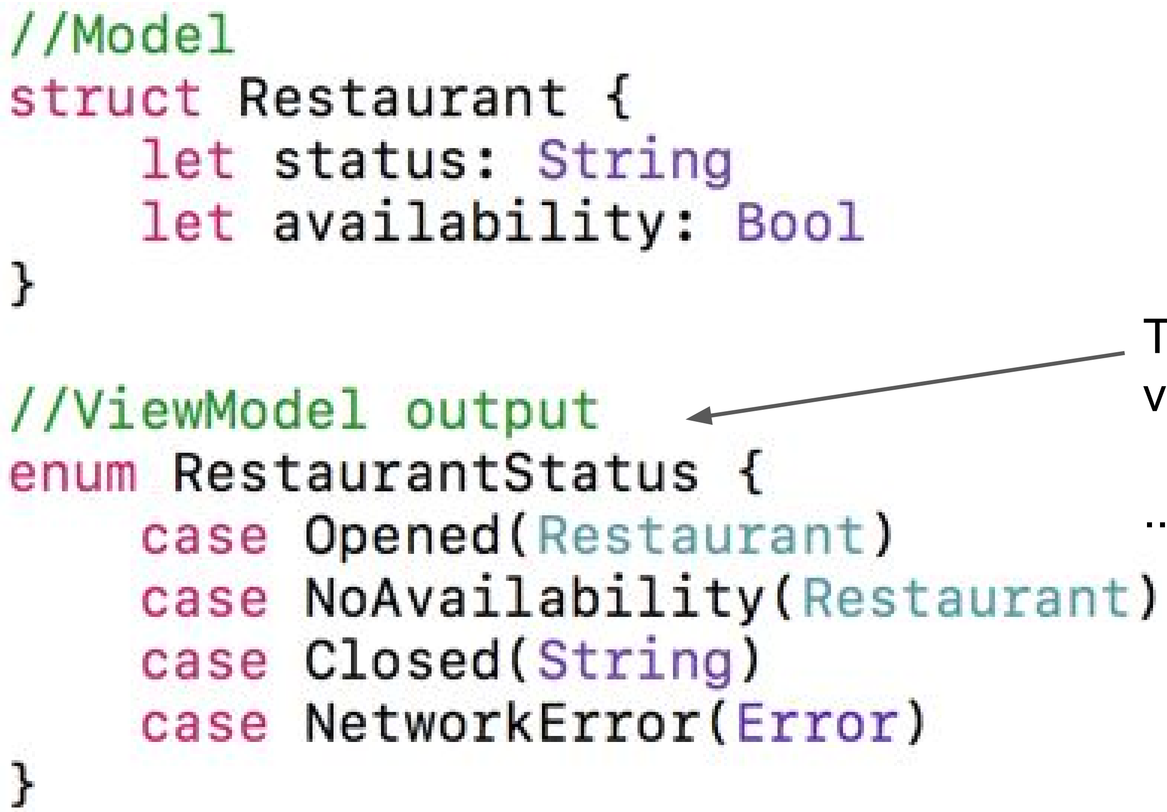
In order to test this, you'll have to supply a view object, a server response, and lots of dependencies

Why MVVM?

Model	**ViewModel**	**ViewController**	**View**
Data container Some business rules	Networking Business Logic Presentation logic	The view controller manipulates the view for a limited set of representational statuses	UI Hierarchy

Given a model or a network response, it produces an **enum** that represents a view state, which the view controller must handle

Quick example



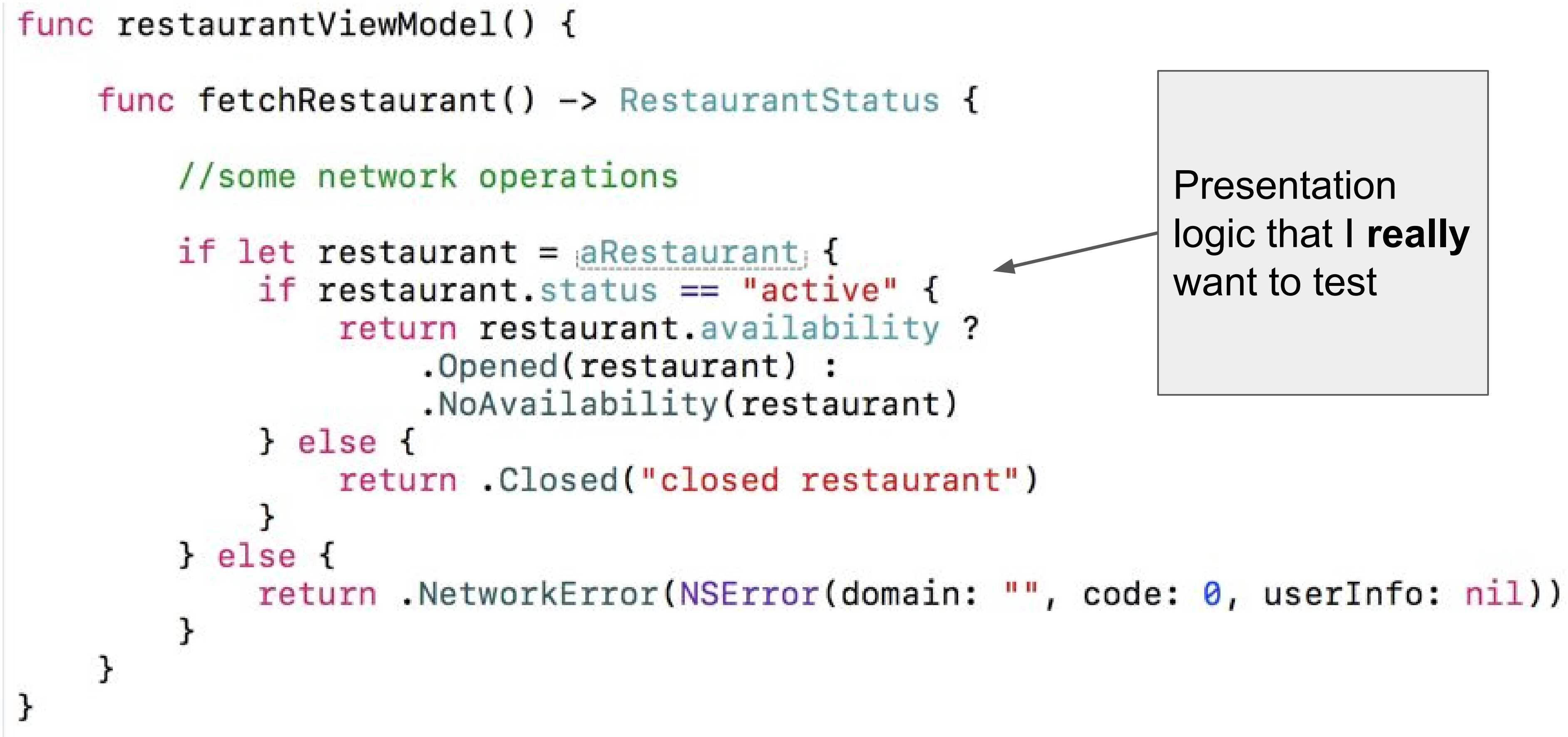
```
//Model
struct Restaurant {
    let status: String
    let availability: Bool
}

//ViewModel output
enum RestaurantStatus {
    case Opened(Restaurant)
    case NoAvailability(Restaurant)
    case Closed(String)
    case NetworkError(Error)
}
```

These are all the states a view can show

... but they’re abstracted.

Quick example



```
func restaurantViewModel() {

    func fetchRestaurant() -> RestaurantStatus {

        //some network operations

        if let restaurant = aRestaurant {
            if restaurant.status == "active" {
                return restaurant.availability ?
                    .Opened(restaurant) :
                    .NoAvailability(restaurant)
            } else {
                return .Closed("closed restaurant")
            }
        } else {
            return .NetworkError(NSError(domain: "", code: 0, userInfo: nil))
        }
    }
}
```

Presentation logic that I **really** want to test

Take away #4

The most important rules should be independent of the views were it is being presented.

You do not want to test those views, since they will easily change. You want to test **the logic** behind it (A.K.A. ViewModel)

Take away #4

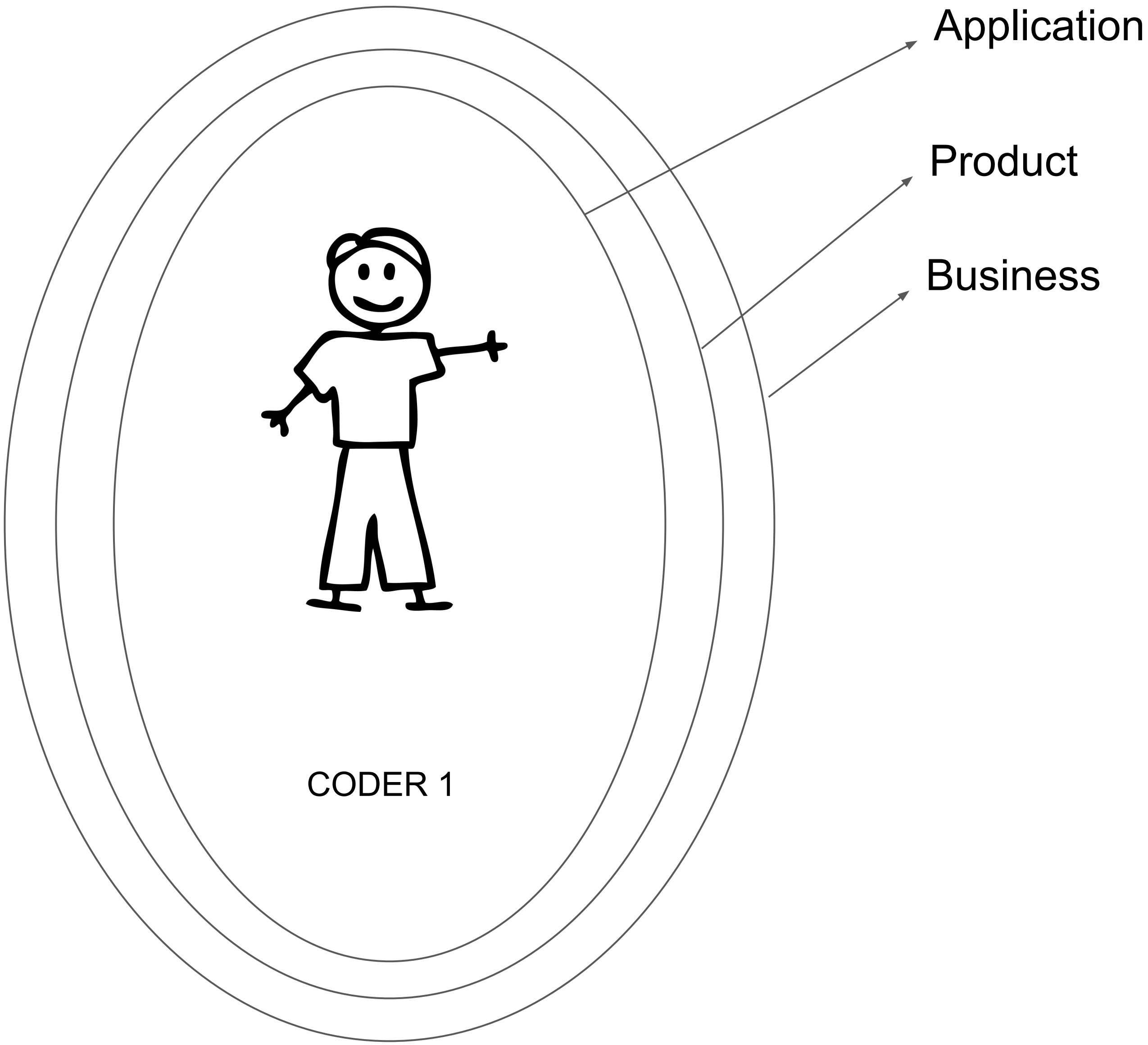
(Sidenote)

In bigger applications I do want to test the UI, but the purpose is to guarantee reliability, not to assert on the application’s behavior.

- Metrics might not be enough
- Uptime is critical
- Way too many developers!

Application Events

Know what matters



A mature developer is the one that can understand how a line of code affects the product, and ultimate the business

To understand business
you need to understand user behavior

No one will ever tell you that an event has stopped being tracked…

until it’s too late

No one

No one

No one

No one

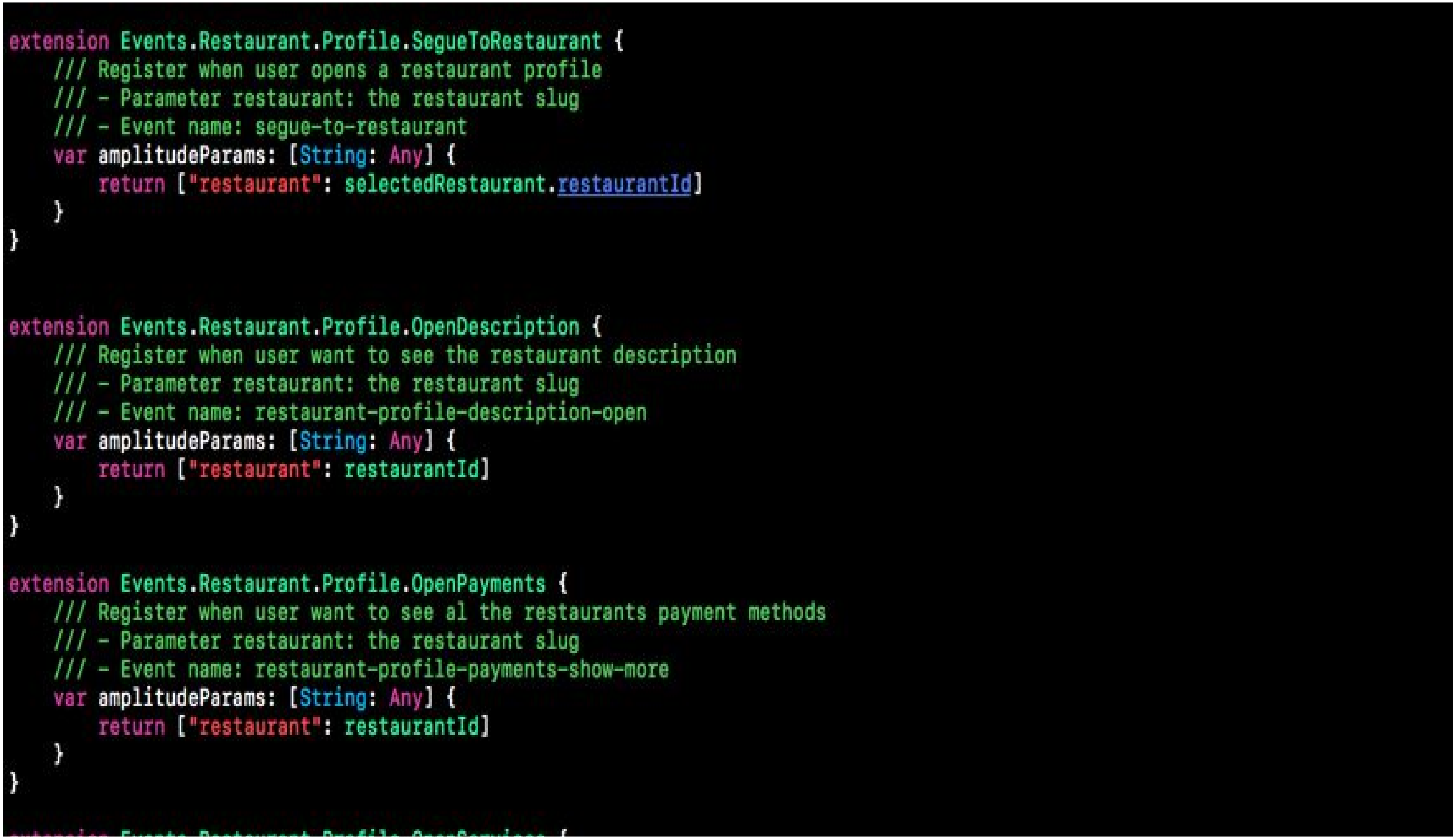
No one

No one

Take away #5

Have 100% coverage of all the events and where are they being triggered.

Upside: Events taxonomy documented!



```
extension Events.Restaurant.Profile.SegueToRestaurant {
    /// Register when user opens a restaurant profile
    /// - Parameter restaurant: the restaurant slug
    /// - Event name: segue-to-restaurant
    var amplitudeParams: [String: Any] {
        return ["restaurant": selectedRestaurant.restaurantId]
    }
}


extension Events.Restaurant.Profile.OpenDescription {
    /// Register when user want to see the restaurant description
    /// - Parameter restaurant: the restaurant slug
    /// - Event name: restaurant-profile-description-open
    var amplitudeParams: [String: Any] {
        return ["restaurant": restaurantId]
    }
}

extension Events.Restaurant.Profile.OpenPayments {
    /// Register when user want to see al the restaurants payment methods
    /// - Parameter restaurant: the restaurant slug
    /// - Event name: restaurant-profile-payments-show-more
    var amplitudeParams: [String: Any] {
        return ["restaurant": restaurantId]
    }
}

extension Events.Restaurant.Profile.OpenServices {
```

Recap

Take away #1
Don’t test what its dummy or change too often

Take away #2
Use a TDD approach to tackle rules that you want to be respected in the future

Take away #3
Learn to decouple your classes so you can use them independently (i.e. in tets)

Take away #4
Use MVVM or similar architecture to decouple presentation rules from views

Take away #5
Have your application events documented in code and fully covered by tests!

Thanks!

Questions?

Thanks again! :)

Github: LucasVidal
Twitter: @lucasvidalutn
Instagram: nope
Snapchat: nope
Email / Hangouts: lucasvidalutn@gmail.com

Restorando

Talk playground gist: http://bit.do/what-to-test