# Embrace your Legacy... Code

@tolkiana

👋

# Hello CodeMash!

## I'm Nelida

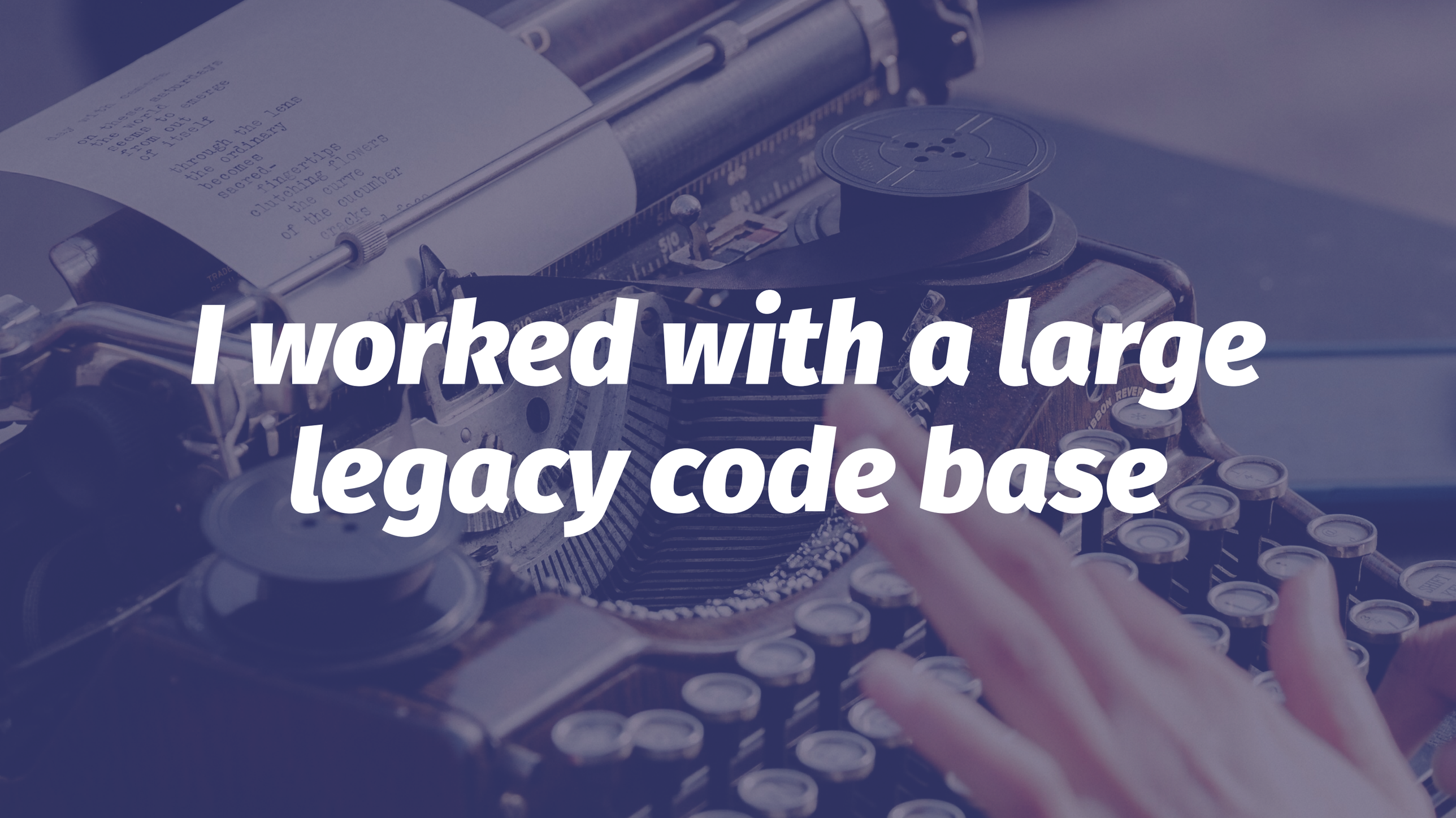
## @tolkiana

**Detroit Labs**

# Why do I care about Legacy code?

@tolkiana


Photo by David Klein on Unsplash




# I worked with a large legacy code base

# What is *Legacy Code*?

@tolkiana

Photo by Viktor Forgacs on Unsplash

***The code that you wrote yesterday is Legacy Code***

-- Thom, former coworker


Photo by Clem Onojeghuo on Unsplash

**Legacy code is not good or bad is just that, *The Heritage* that others have left for us to build upon**

# What legacy codebases taught me?

@tolkiana

# Patience

@tolkiana

# Persistence

@tolkiana

# Discipline

@tolkiana

# Curiosity

@tolkiana

# Empathy

@tolkiana

# Humility

@tolkiana

# Working with legacy code bases is fundamental for any software developer

Photo by Dane Deaner on Unsplash

# What do I do now?


Legacy code takes a lot of time, care and love
@tolkiana

# 1. Understand

✓ **Read documentation if any**

✓ Read documentation if any

✓ **Get to know your app as a user**

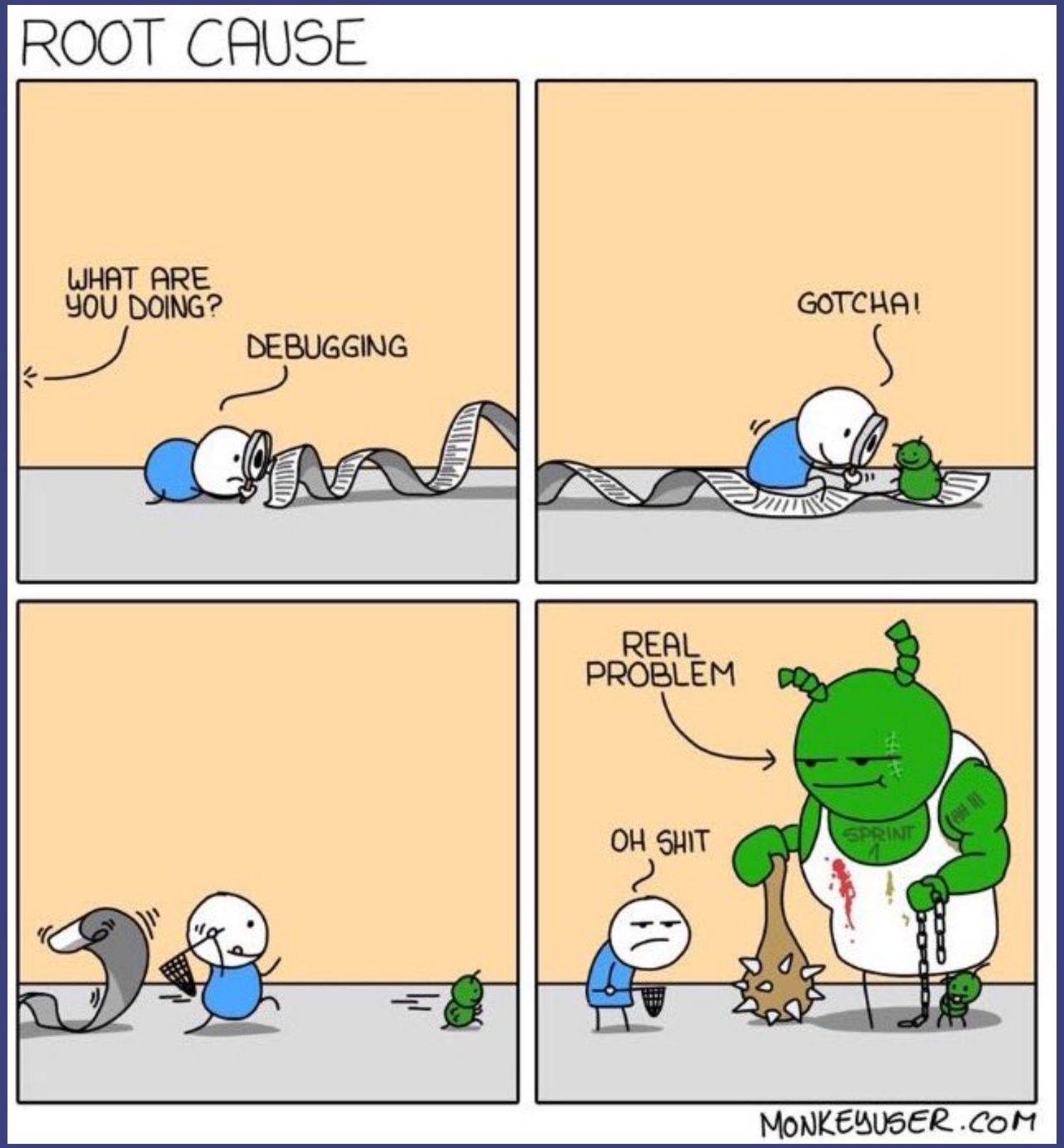
- ✓ Read documentation if any
- ✓ Get to know your app as a user
- ✓ **Use the UI as an entry point**

- ✓ Read documentation if any
- ✓ Get to know your app as a user
- ✓ Use the UI as an entry point
- ✓ **Pair**

***Working with legacy code is like surgery and doctors don't operate alone.***

-- Michael C. Feathers

- ✓ Read documentation if any
- ✓ Get to know your app as a user
- ✓ Use the UI as an entry point
- ✓ Pair
- ✓ **Work on Bugs**


ROOT CAUSE
WHAT ARE YOU DOING?
DEBUGGING
GOTCHA!
REAL PROBLEM
OH SHIT
SPRINT
MONKEYUSER.COM

# 2. Document

✓ **Use it as a learning tool**

✓ Use it as a learning tool

✓ **Use it to share knowledge**

***When teams aren't aware of their architecture, it tends to degrade... The brutal truth is that architecture is too important to be left exclusively to a few people***

-- Michael C. Feathers

- ✓ Use it as a learning tool
- ✓ Use it to share knowledge
- ✓ **Use it to refresh your memory**

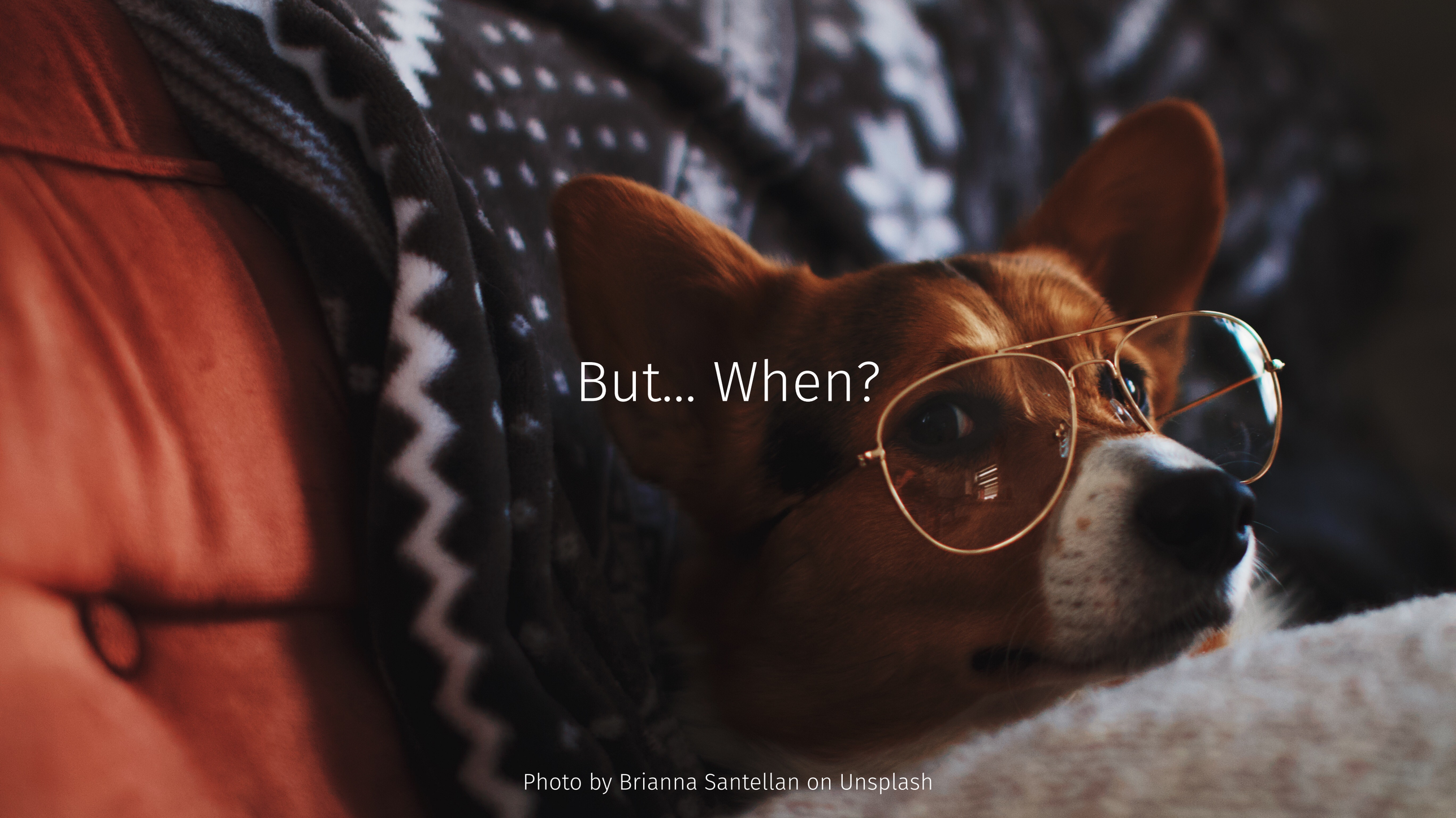



Photo by Brianna Santellan on Unsplash

# 3. Define a direction

**Make your team the priority, without a healthy team, there's no code base that can be improved**

# 4. Test or improve current tests

# Common testing traps

# Common testing traps

- ✓ Letting unit tests to grow into mini integration tests

# Common testing traps

- ✓ Letting unit test to grow into mini integration tests
- ✓ **Wanting to test every single thing**

***Tests don’t have moral authority, they are there to document code***

-- Michael C. Feathers

# 5. Break dependencies

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = Order.shared.productsInCart()
        display(products)

        let mySetting = UserDefaults.standard.string(forKey: "mySetting")
        customize(mySetting)

        URLSession.shared.dataTask(with: URL(string: "http://service.com")!) { (data, response, error) in
            // do something
        }
    }

}
```

```
// Stubbing to tests in Objective-C

[order stub:@selector(productsInCart) andReturn:@["🍗","🥩","🥖"]];
[myController stub:@selector(order) andReturn:order];

[myController stub:@selector(urlSession) andReturn:mockSession];
[myController stub:@selector(userDefaults) andReturn:mockUserDefaults];
```

```
// Stubbing to tests in Objective-C

[order stub:@selector(productsInCart) andReturn:@["🍗","🥩","🥖"]];
[myController stub:@selector(order) andReturn:order];

[myController stub:@selector(urlSession) andReturn:mockSession];
[myController stub:@selector(userDefaults) andReturn:mockUserDefaults];
```

```
// Stubbing to tests in Objective-C

[order stub:@selector(productsInCart) andReturn:@["🍗","🥩","🥖"]];
[myController stub:@selector(order) andReturn:order];

[myController stub:@selector(urlSession) andReturn:mockSession];
[myController stub:@selector(userDefaults) andReturn:mockUserDefaults];
```

```
class Order {

    static let shared = OrderManager()
    init(){}

    func productsInCart() -> [String] {
        var products: [String] = []

        // Some complex and very scary code

        return products
    }

    func addProduct(_ product: String) {
        // Adding product
    }

    func removeProduct(_ product: String) {
        // Remove product
    }
}
```

```swift
protocol OrderProtocol {

    func productsInCart() -> [String]
}

class Order: OrderProtocol {

    func productsInCart() -> [String] {
        var products: [String] = []

        // Some complex and very scary code

        return products
    }
}

class FakeOrder: OrderProtocol {

    func productsInCart() -> [String] {
        return ["🥑", "🥕", "🧀"]
    }

}
```

```swift
protocol OrderProtocol {

    func productsInCart() -> [String]
}

class Order: OrderProtocol {

    func productsInCart() -> [String] {
        var products: [String] = []

        // Some complex and very scary code

        return products
    }
}

class FakeOrder: OrderProtocol {

    func productsInCart() -> [String] {
        return ["🥑", "🥕", "🧀"]
    }
}
```

```swift
protocol OrderProtocol {

    func productsInCart() -> [String]
}


class Order: OrderProtocol {

    func productsInCart() -> [String] {
        var products: [String] = []

        // Some complex and very scary code

        return products
    }
}

class FakeOrder: OrderProtocol {

    func productsInCart() -> [String] {
        return ["🥑", "🥕", "🧀"]
    }
}
```

```
class MyViewController: UIViewController {

    private let defaults: UserDefaults
    private let urlSession: URLSession
    private let order: OrderProtocol

    init(order: OrderProtocol, defaults: UserDefaults, urlSession: URLSession) {
        self.defaults = defaults
        self.urlSession = urlSession
        self.order = order
        super.init(nibName: nil, bundle: nil)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = order.productsInCart()
        display(products)

        let mySetting = defaults.string(forKey: "mySetting")
        customize(mySetting)

        urlSession.dataTask(with: URL(string: "http://myService.com")!) { (data, response, error) in
            // do something
        }
    }
}
```

```swift
class MyViewController: UIViewController {

    private let defaults: UserDefaults
    private let urlSession: URLSession
    private let order: OrderProtocol

    init(order: OrderProtocol, defaults: UserDefaults, urlSession: URLSession) {
        self.defaults = defaults
        self.urlSession = urlSession
        self.order = order
        super.init(nibName: nil, bundle: nil)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        let products = order.productsInCart()
        display(products)

        let mySetting = defaults.string(forKey: "mySetting")
        customize(mySetting)

        urlSession.dataTask(with: URL(string: "http://myService.com")!) { (data, response, error) in
            // do something
        }
    }
}
```

```
class MyViewController: UIViewController {

    init(order: OrderProtocol = Order.shared, defaults: UserDefaults = .standard, urlSession: URLSession = .shared) {
        self.defaults = defaults
        self.urlSession = urlSession
        self.order = order
        super.init(nibName: nil, bundle: nil)
    }
}

// In production
let controller = MyViewController()

// In Tests
let testController = MyViewController(orderManager: FakeOrder(), defaults: FakeDefaults(), urlSession: FakeSession())
```

```
class MyViewController: UIViewController {

    init(order: OrderProtocol = Order.shared, defaults: UserDefaults = .standard, urlSession: URLSession = .shared) {
        self.defaults = defaults
        self.urlSession = urlSession
        self.order = order
        super.init(nibName: nil, bundle: nil)
    }
}

// In production
let controller = MyViewController()

// In Tests
let testController = MyViewController(orderManager: FakeOrder(), defaults: FakeDefaults(), urlSession: FakeSession())
```

```
class MyViewController: UIViewController {

    init(order: OrderProtocol = Order.shared, defaults: UserDefaults = .standard, urlSession: URLSession = .shared) {
        self.defaults = defaults
        self.urlSession = urlSession
        self.order = order
        super.init(nibName: nil, bundle: nil)
    }
}

// In production
let controller = MyViewController()

// In Tests
let testController = MyViewController(orderManager: FakeOrder(), defaults: FakeDefaults(), urlSession: FakeSession())
```

***When we break dependencies, we can often write tests that can make more invasive changes safer. The trick is to do these initial refractories very conservatively.***

-- Michael C. Feathers

# 6. Small, incremental improvements

# Common refactoring traps

# Common refactoring traps

✓ Getting over ambitious

# Common refactoring traps

- ✓ Getting over ambitious
- ✓ Not setting a goal

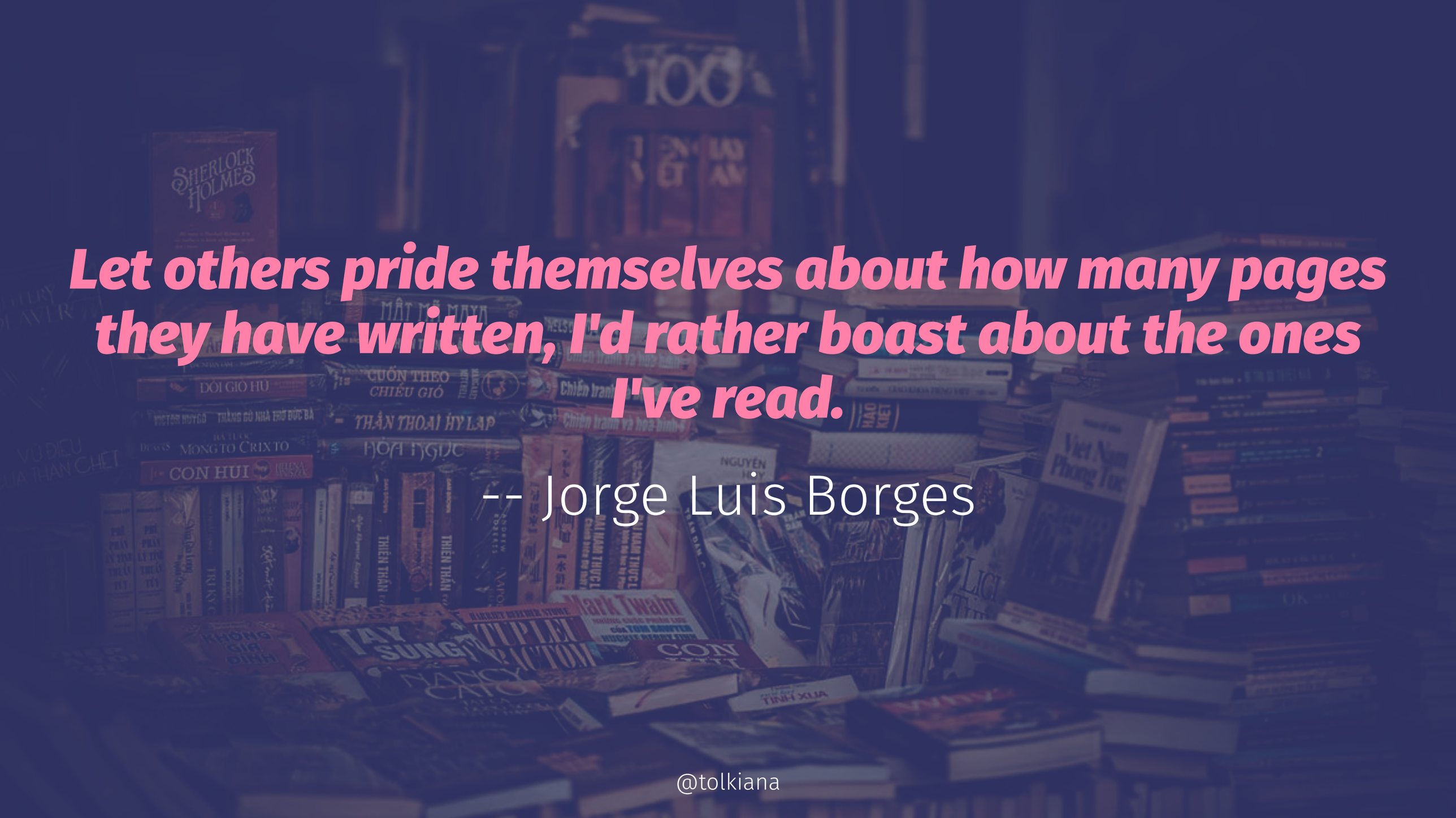
# Common refactoring traps

- ✓ Getting over ambitious
- ✓ Not setting a goal
- ✓ Not sharing our goal/architecture changes.

***Working Effectively With Legacy Code***
Michael C. Feathers

***Refactoring***
Martin Fowler

@tolkiana




***Let others pride themselves about how many pages they have written, I'd rather boast about the ones I've read.***

-- Jorge Luis Borges

@tolkiana

# Thank you!

@tolkiana

# Questions?


## Nelida Velázquez

@tolkiana

nelida.velazquez@detroitlabs.com