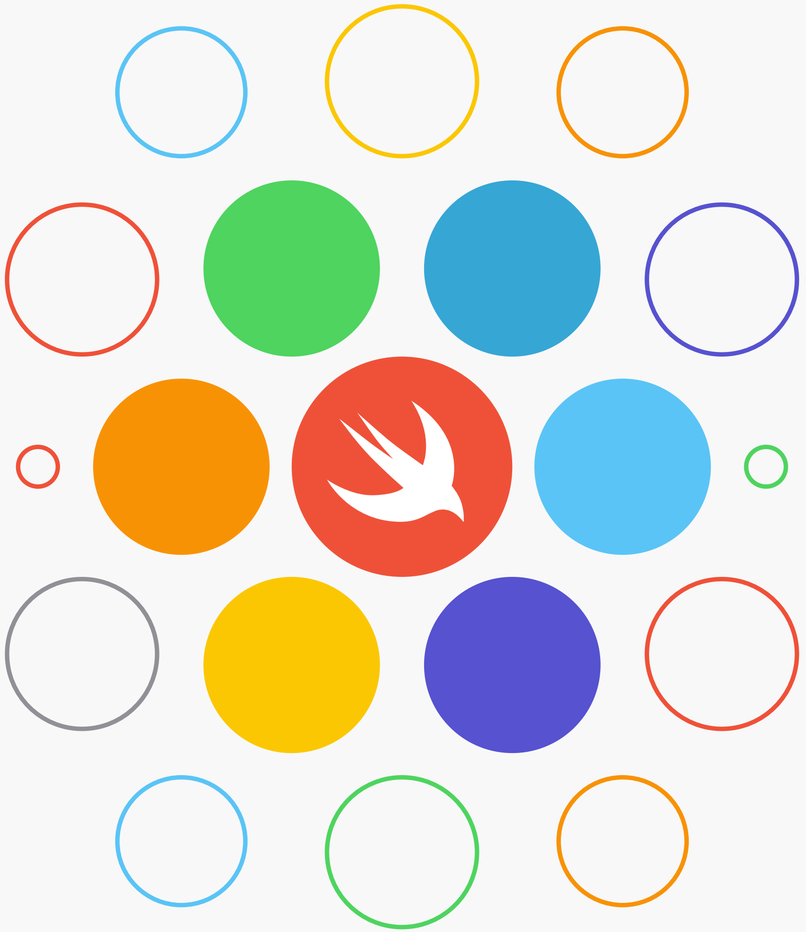


let Swift

iOS DEVELOPERS MEETUP

Adrian

+

Netguru

Testing...

Agenda

1. Standard ways of testing
2. Introduction to property-based testing
3. Integrating property-based testing with your existing architecture

Triple-A Testing

Arrange
Act
Assert

```
// 1. Arrange
beforeEach {
  sut = CGSize(width: 10, height: 20)
}

it("should scale appropriately") {

  // 2. Act
  sut.scale(by: 2)

  // 3. Assert
  expect(sut.width).to(equal(20))
  expect(sut.height).to(equal(40))

}
```

```
// 1. Arrange
override func setUp() {
  sut = CGSize(width: 10, height: 20)
}

func testScalesAppropriately() {

  // 2. Act
  sut.scale(by: 2)

  // 3. Assert
  XCTAssertEqual(sut.width, 20)
  XCTAssertEqual(sut.height, 40)

}
```

xUnit
xSpec

xUnit & xSpec

- 👍 Widely adopted
- 👍 Easy to learn and reason about
- 👍 Plays well with stateful objects and functions producing side-effects

xUnit & xSpec

- 👎 Doesn’t fit well with immutable values and pure functions
- 👎 You must predict all edge cases

```haskell
unsplit :: Char -> [String] -> String
unsplit c = concat . intersperse [c]

split :: Char -> String -> [String]
split c xs = xs' : if null xs'' then [] else split c (tail xs'')
    where xs' = takeWhile (/=c) xs
          xs''= dropWhile (/=c) xs

prop_split_inv xs
    = forAll (elements xs) $ \c ->
      unsplit c (split c xs) == xs

main = quickCheck prop_split_inv
```

QuickCheck

SwiftCheck

github.com/typelift/SwiftCheck

Testing with SwiftCheck

1. You write assertions about logical properties
2. SwiftCheck attempts to find a failing case
3. If it finds one, SwiftCheck reduces input to the smallest value that causes failure

```
struct Point: Equatable {

    var x: Double
    var y: Double

    init(string: String) throws

    var string: String { get }

}
```

```
Point(x: 1, y: 2)
// => Point(x: 1, y: 2)

Point(x: 3, y: 4).string
// => "{3, 4}"

try Point(string: "{5, 6}")
// => Point(x: 5, y: 6)
```

```
func testPointToStringConversion() {

    XCTAssertEqual(
        Point(x: 1, y: 2).string,
        "{1, 2}"
    )

    XCTAssertEqual(
        Point(x: -1, y: -2).string,
        "{-1, -2}"
    )

}
```

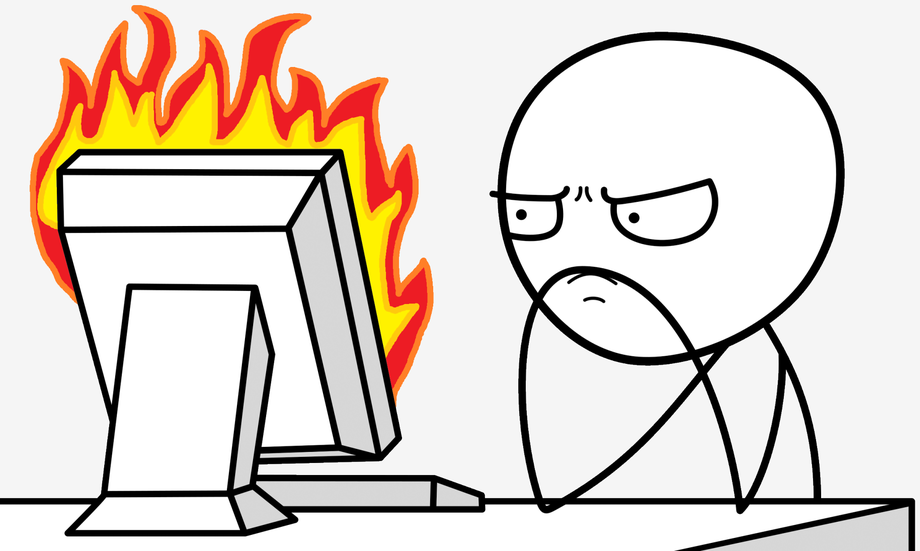
```
func testPointToStringConversion() {

    property("Point is convertible to and from String")
    <- forAll { (x: Double, y: Double) in

        let originalPoint = Point(x: x, y: y)
        let string = point.string
        let secondaryPoint = try Point(string: string)

        return secondaryPoint == originalPoint
    }

}
```

```
var string: String {
    return String(format: "{%d, %d}", x, y)
}
```

```
 Point(x: 235, y: 7893) => Point(x: 235, y: 7893)
    Point(x: 0, y: -93) => Point(x: 0, y: -93)

..

Point(x: 56.7, y: 22.1) => Point(x: 56, y: 22)

Point(x: 10.9, y: -3.4) => Point(x: 10, y: -3)
 Point(x: -1.4, y: 5.1) => Point(x: -1, y: 5)

..

    Point(x: 0.1, y: 0) => Point(x: 0, y: 0)
```

```
func testPointToStringConversion() {

    property("Point is convertible to and from Str Property failed: (0.1, 0)
    <- forAll { (x: Double, y: Double) in

        let originalPoint = Point(x: x, y: y)
        let string = point.string
        let secondaryPoint = try Point(string: string)

        return secondaryPoint == originalPoint
    }

}
```

Demo



Summary

Don’t write tests…
Generate them!

```
property("all questions have answers")
<- forAll { (question: Question) in
    Answer(to: question) != nil
}
```

@akashivskyy

Twitter, GitHub

akashivskyy/talks

Repository