Always give 100% percent

Jakub Turek

19th September, 2018

About me

Jakub Turek

- https://jakubturek.com
- @KubaTurek
- turekj
- EL Passion

EL Passion

- Product house from Warsaw:
  - UX & UI.
  - Web apps.
  - Native mobile apps.
- Core principles:
  - Working closely with clients.
  - All-or-nothing approach to quality.

Agenda

1. Introduction to unit-testing & TDD.
2. Engineering for testability.
3. iOS testing traps.
4. Good engineers’ mistakes.
5. Refactoring test code.
6. Test code generation.
7. Metrics.

Motivation

Retrospective of a recent project in numbers:

- 3 people.
- 12 months of work.
- 217 764 lines of Swift code.
- 5 811 files.
- 8 internal frameworks.
- 7 353 test cases.
- **Hard deadline**.

Introduction to testing

Unit testing

A unit test is an automated piece of code that invokes ***a unit of work*** *in the system and then checks a single assumption about the behavior of that unit of work.*

— Roy Osherove, The art of unit testing

TDD

TDD is a programming technique which combines writing a test before writing just enough production code to fulfill that test and refactoring.

— Kent Beck, Test-Driven Development by example

Why TDD?

- Enables refactoring.
- Software design discipline.
- Improved quality:
  - 15 – 35% ⏰ increase.
  - 40 – 90% 🐛 density decrease.

TDD workflow

1. Red - write a failing test.
2. Green - write minimal amount of code to pass.
3. Refactor.

3 laws of TDD

1. You can't write any production code until you write a failing unit test.
2. You can't write more of a unit test than is sufficient to fail. *Not compiling is failing*.
3. You can't write more production code than is sufficient to pass currently failing unit test.

DEMO

Engineering for testability

Pure functions

How to write testable code?

1. *Pass values to functions.*
2. *Return values from functions.*

— @mdiep, Matt Diephouse, Twitter

Boundaries

Boundaries by Gary Bernhardt

This talk is about using simple values (as opposed to complex objects) not just for holding data, but also as the boundaries between components and subsystems.

https://destroyallsoftware.com/talks/boundaries

Imperative shell, functional core

Imperative shell:

- real world dependencies,
- side-effects,
- stateful operations.

Functional core:

- decisions,
- purely functional transformations.

Demo

Resources

Imperative shell, functional core

Detailed demo description, tons of additional resources.

https://jakubturek.com/imperative-shell-functional-core/

Dealing with massive controllers

Better MVC

`UIViewController` $\neq$ *screen. You don't have to fill the screen. A single screen can show lots of* `UIViewControllers`.

— Dave DeLong, App Builders 2018

More View Controllers

How to decouple controllers?

- ~~Never~~ Subclass as a last resort.
- Use child controllers for composition.
- Use protocols for controllers’ boundaries.
- Refer to controllers using compound types (`UIViewController & ControllerProtocol`).
- Expect to reuse controllers.

Adding a child controller

```swift
func embed(child: UIViewController,
           inside view: UIView) {
  addChildViewController(child)
  view.addSubview(child.view)
  child.view.edgeAnchors == view.edgeAnchors
  child.didMove(toParentViewController: self)
}
```

Decoupling controllers (1/2)

```
protocol ChildControllerType: class {
  var productSelected: ((String) -> Void)? { get set }
}

class ChildController: UIViewController,
                       ChildControllerType {
  init(/* dependencies */) { /* ... */ }

  var productSelected: ((String) -> Void)?

  override func viewDidLoad() {
    /* implementation */
  }
}
```

Decoupling controllers (2/2)

```swift
typealias ChildControlling =
      UIViewController & ChildControllerType

class ParentController: UIViewController {
  init(factory: () -> ChildControlling) {
    self.factory = factory
  }

  override func viewDidLoad() {
    super.viewDidLoad()
    embed(child: child, inside: view)
  }

  private lazy var child = factory()
  private let factory: () -> ChildControlling
}
```

Stubbing a child controller (1/2)

```swift
class ChildControllerStub: UIViewController,
                           ChildControllerType {
  var productSelected: ((String) -> Void)?
}
```

Stubbing a child controller (2/2)

```swift
class ParentControllerTests: XCTestCase {
  var childStub: ChildControllerStub!
  var sut: ParentController!

  override func setUp() {
    super.setUp()
    childStub = ChildControllerStub()
    sut = ParentController(factory: { childStub })
  }

  func testThatSelectedProductNameIsDisplayed() {
    childStub.productSelected?("Water")
    XCTAssertEqual(sut.view.label.text, "Water")
  }
}
```

Examples

- Buttons.
- Forms fields.
- API data coordination with immutable children.

Our metrics

- Longest file (controller) in a project: 140 lines.
- Controllers in total: 164.
- Screens in total: 38.
- Average 4.32 controller per screen.
- Average 75.16 lines of code per controller.

Views

iOSSnapshotTestCase

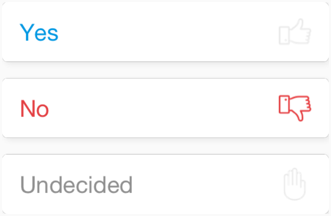
iOSSnapshotTestCase

`iOSSnapshotTestCase` takes preconfigured view and renders its snapshot. It compares snapshot to a “reference image” stored in repository and fails if the two images don’t match.

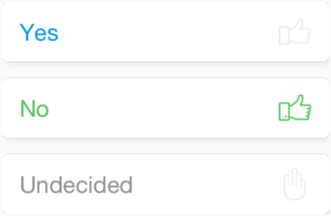
https://github.com/uber/ios-snapshot-test-case

iOSSnapshotTestCase diff

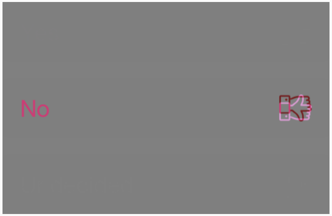
Expected



Actual



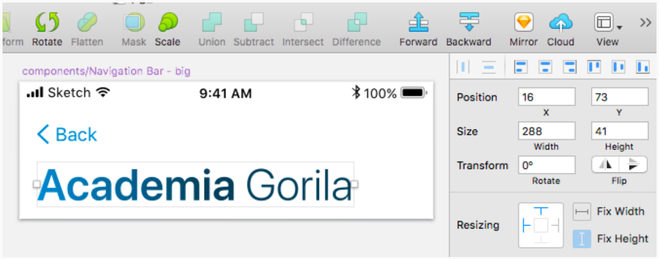
Diff



Instant feedback for views

Red phase (1/3)

Note down the size from design (≈ 300 × 40).



Position 16 X, 73 Y
Size 288 Width, 41 Height
Transform 0° Rotate, Flip
Resizing

Red phase (2/3)

```
override func setUp() {
  super.setUp()
  recordMode = true

  sut = GradientLabelView(text: "Academia Gorila")
  sut.frame = CGRect(width: 300, height: 40)
}

func testLabelMatchesSnapshot() {
  FBSnapshotVerifyView(sut)
}

class GradientLabelView { // ... implementation }
```

Red phase (3/3)

Build the view in iterations:

1. Change the code.
2. Run the tests.
3. Compare a reference image to the design:
   - Repeat the cycle if needed.

Green phase

```
override func setUp() {
  super.setUp()
  recordMode = false

  sut = GradientLabelView(text: "Academia Gorila")
  sut.frame = CGRect(width: 300, height: 40)
}

func testLabelMatchesSnapshot() {
  FBSnapshotVerifyView(sut)
}
```

Refactor phase

- Move the view to a production target.
- Refactor the view.

Global methods

Global method example

```
static func drawImage(of size: CGSize,
                      using drawer: (CGContext) -> Void)
                      -> UIImage? {
  UIGraphicsBeginImage...(size, false, 0.0)

  defer { UIGraphicsEndImageContext() }

  guard let ctx = UIGraphicsGetCurrentContext() else {
      return nil
  }

  drawer(ctx)

  return UIGraphicsGetImage...()
}
```

Swift namespace resolution (1/2)

Shipping your own controller type:

```
class UIViewController {
  func theTimeHasComeToLoadAView() {
    myOwnPersonalView = SomeView()
  }
}

let controller = UIViewController()
controller.theTimeHasComeToLoadAView() // works
```

Swift namespace resolution (2/2)

Shipping your own controller library:

```
# Podfile
pod 'MyOwnController', '~> 0.1'
```

```
// SourceFile.swift
import MyOwnController
import UIKit

UIViewController() // compilation error
```

Testing global methods (1/2)

```
import UIKit

func UIGraphicsGetImageFromCurrentImageContext()
     -> UIImage? {
  return imageFromContext()
}

var imageFromContext: () -> UIImage? =
  UIKit.UIGraphicsGetImageFromCurrentImageContext
```

Testing global methods (2/2)

```
override func setUp() {
  imageFromContext = { UIImage.catMeme }
}

override func tearDown() {
  imageFromContext = UIKit.UIGraphicsGetImage...
}

func testThatDrawImageReturnsImageFromContext() {
  let image = CGContext.drawImage(of: .zero) { _ in }

  XCTAssertEqual(
    UIImagePNGRepresentation(image),
    UIImagePNGRepresentation(imageFromContext())
  )
}
```

Apple

Push notifications (1/2)

```
open class UNNotificationSettings: NSObject,
                                   NSCopying,
                                   NSSecureCoding {
  open var authorizationStatus:
    UNAuthorizationStatus { get }

  public init?(coder aDecoder: NSCoder)
    // NS_DESIGNATED_INITIALIZER
}
```

Push notifications (2/2)

```
class DecoderFake: NSCoder {
  override func decodeInt64(forKey _: String) -> Int64 {
    return 0
  }

  override func decodeBool(forKey _: String) -> Bool {
    return false
  }

  override func decodeObject() -> Any? {
    return nil
  }

  override var allowsKeyedCoding: Bool {
    return false
  }
}
```

Good engineer’s mistakes

Stubbing to death

Mock across architecturally significant boundaries, but not within those boundaries.

— Robert C. Martin, When to Mock

Overmocking (1/3)

```swift
class URLBuilderSpy: URLBuilding {
  private(set) var routeSpy: [(String, ImageType)] = []

  func route(forImageWithURL imageURL: String,
             of type: ImageType) throws -> URL {
    routeSpy.append((imageURL, type))
    return URL(string: "https://google.com")!
  }
}
```

Overmocking (2/3)

```
class ImageFetcherSpy: ImageFetching {
  private(set) var imageSpy: [URL] = []

  func image(for url: URL) -> Single<UIImage> {
    imageSpy.append(url)
    return Single.just(UIImage.testImage)
  }
}
```

Overmocking (3/3)

```
var urlBuilderSpy: URLBuilderSpy!
var fetcherSpy: ImageFetcherSpy!

func testThatAvatarsAreFetched() {
  let images = try! sut.fetchAvatars()
    .toBlocking().first()

  XCTAssertEqual(["one", "two"],
    urlBuilderSpy.routeSpy.map { $0.0 })
  XCTAssertEqual([.userAvatar, .userAvatar],
    urlBuilderSpy.map { $0.1 })
  XCTAssertEqual([URL.google, URL.google],
    fetcherSpy.imageSpy)
  XCTAssertImages([.testImage, .testImage], images)
}
```

Maintaining unreliable tests

Unit test

Good unit test is:

- automated,
- fast,
- tests a single logical concept in the system,
- **trustworthy**.

Keeping false positives

✂ **Unreliable tests**

A single false positive will eventually kill the purpose of thousands meaningful tests in a suite.

Refactoring test code

Testing models (1/2)

```
struct User {
  let id: Int
  let born: Date
}
```

Testing models (2/2)

```
func testThatUsersBornInJanuaryGetAPrize() {
  let u1 = User(
    id: 3,
    born: Date(timeIntervalSince1970: 631886400)
  )
  let u2 = User(
    id: 6,
    born: Date(timeIntervalSince1970: 634233600)
  )
  let u3 = User(
    id: 8,
    born: Date(timeIntervalSince1970: 727466400)
  )

  XCTAssertEqual(sut.winners([u1, u2, u3]), [3, 8])
}
```

Test data

```swift
extension User {
 static var bornJanuary1990: User {
  return User(id: 3, born: "1990-01-09 12:00".date())
 }

 static var bornFebruary1990: User {
  return User(id: 6, born: "1990-02-05 16:00".date())
 }

 static var bornJanuary1993: User {
  return User(id: 8, born: "1993-01-19 18:00".date())
 }
}
```

Refactored tests

```swift
func testThatUsersBornInJanuaryGetAPrize() {
  let winnerIDs = sut.winners([
    .bornJanuary1990,
    .bornFebruary1990,
    .bornJanuary1993
  ])

  XCTAssertEqual(winnerIDs, [3, 8])
}
```

Test code generation

Sourcery

Sourcery

Sourcery is a code generator for Swift language, built on top of Apple’s own SourceKit. It extends the language abstractions to allow you to generate boilerplate code automatically.

https://github.com/krzysztofzablocki/Sourcery

Problem: required initializers

0% code coverage when not using inteface builder 😭

```
required init?(coder aDecoder: NSCoder) {
  return nil
}
```

AllViews.stencil

```
extension UIView {

 static var allInitializers: [(NSCoder) -> UIView?] {
  return [{% for view in types.classes where view.based.UIView %}
   {% set spacer %}{% if not forloop.last %},{% endif %}{% endset %}
    {% for initializer in view.initializers %}
    {% if initializer.selectorName == "init(coder:)" %}
     {{ view.name }}.init(coder:){{ spacer }}
    {% endif %}
    {% endfor %}
   {% endfor %}]
 }

}
```

AllViews.generated.swift

```swift
extension UIView {

  static var allInitializers: [(NSCoder) -> UIView?] {
    return [
      ActionBarView.init(coder:),
      AuthorizeErrorView.init(coder:),
      BlurView.init(coder:),
      BorderedButtonView.init(coder:),
      FilterView.init(coder:),
      HeaderView.init(coder:),
      /* ... */
    ]
  }

}
```

AllViewsTests.swift

```
func testThatAllViewsAreNonCodable() {
  UIView.allInitializers.forEach { initializer in
   XCTAssertNil(initializer(NSCoder()))
  }
}
```

Other usages

- Automatic synthesizing of `Equatable` conformance in *extensions*.
- Mock object generation.
- Complex assertions:
  - There is a factory class for every `Route`.
  - There is an integration test for every request.

Metrics

Metrics

Measured on every pull request:

- Project size:
    - lines of code,
    - files count,
    - average file size.
- Static analysis:
    - `SwiftLint`: consistent coding style,
    - `jscpd`: automatic copy-paste detection.
- Code coverage.

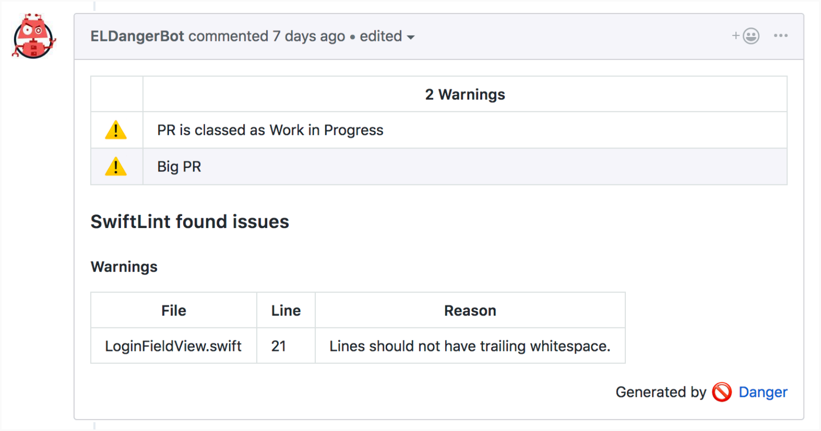
Metrics - visualization

Danger

`Danger` runs during your CI process, and gives teams the chance to automate common code review chores.

http://danger.systems/js/swift.html

Danger



ELDangerBot commented 7 days ago • edited

	2 Warnings
⚠️	PR is classed as Work in Progress
⚠️	Big PR

SwiftLint found issues

Warnings

File	Line	Reason
LoginFieldView.swift	21	Lines should not have trailing whitespace.

Generated by 🚫 Danger

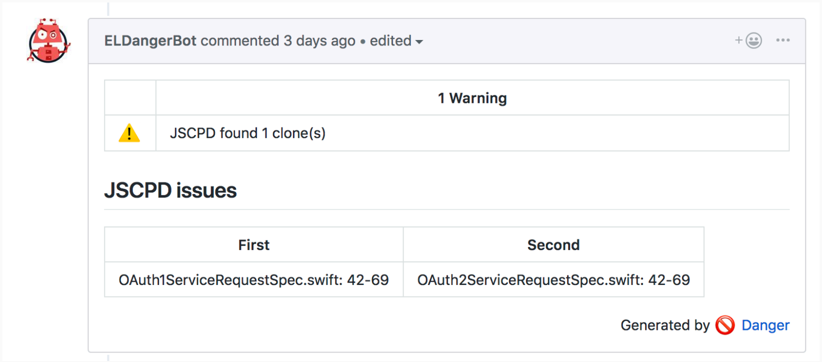
Automatic copy-paste detection

JSCPD

`jscpd` is a tool for detect copy/paste ”design pattern” in programming source code.

https://github.com/kucherenko/jscpd

JSCPD + Danger = ❤️ (1/3)



ELDangerBot commented 3 days ago • edited

	1 Warning
⚠️	JSCPD found 1 clone(s)

JSCPD issues

First	Second
OAuth1ServiceRequestSpec.swift: 42-69	OAuth2ServiceRequestSpec.swift: 42-69

Generated by 🚫 Danger

JSCPD + Danger = ❤️ (2/3)

cpd.yaml:

```yaml
languages:
  - swift
files:
  - "Sources/**"
exclude:
  - "**/*.generated.swift"
reporter: json
output: jscpd_report.json
```

JSCPD + Danger = ❤️ (3/3)

Dangerfile:

```ruby
def find_duplicates
  `jscpd`

  rep = JSON.parse(File.read('jscpd_report.json'))
  clones = rep["statistics"]["clones"]

  if clones > 0
    warn("JSCPD found #{clones} clone(s)")
  end
end
```

Full version: https://tinyurl.com/yc23t4mb

Code coverage

Code coverage is a percentage of code which is covered by automated tests.

> *Test coverage is a useful tool for finding untested parts of a codebase. Test coverage is of little use as a numeric statement of how good your tests are.*
>
> — *Martin Fowler, TestCoverage*

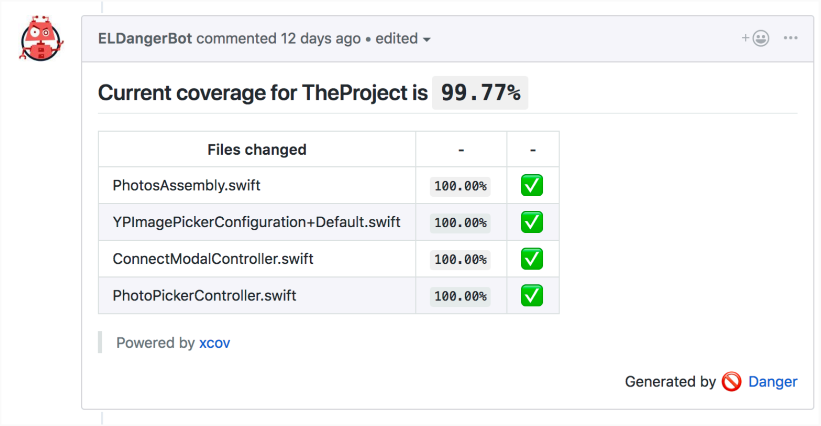
DANGER-XCOV (1/2)

danger-xcov

`danger-xcov` is the Danger plugin of xcov, a friendly visualizer for Xcode's code coverage files.

https://github.com/nakiostudio/danger-xcov

DANGER-XCOV (2/2)


ELDangerBot commented 12 days ago • edited

Current coverage for TheProject is `99.77%`

Files changed	-	-
PhotosAssembly.swift	`100.00%`	✅
YPImagePickerConfiguration+Default.swift	`100.00%`	✅
ConnectModalController.swift	`100.00%`	✅
PhotoPickerController.swift	`100.00%`	✅

> Powered by xcov

Generated by 🚫 Danger

Thank you!

Additional resources

- Jakub Turek
  Always give one hundred percent
  https://jakubturek.com/talks/
- Jon Reid
  Quality Coding - Blog
  https://qualitycoding.org/blog/
- Kasper B. Graversen
  Gist: functional core, imperative shell
  https://tinyurl.com/y9cxblm8

Feedback ❤️

@elpassion