How to Write Testable Code

Swift TO - 2019

Marwan Alani

@marwanalany

“I’ve been coding professionally for about 15 years, and I still struggle with writing testable code”

Marwan Alani

A Modern Gentle Introduction to Software Testing

Using Code Coverage & TDD

Gentle-r Introduction to Software Testing

Testabilitiness

Side Effects / Byproducts

Tests are the goal

What if…

The side effects were the goal?

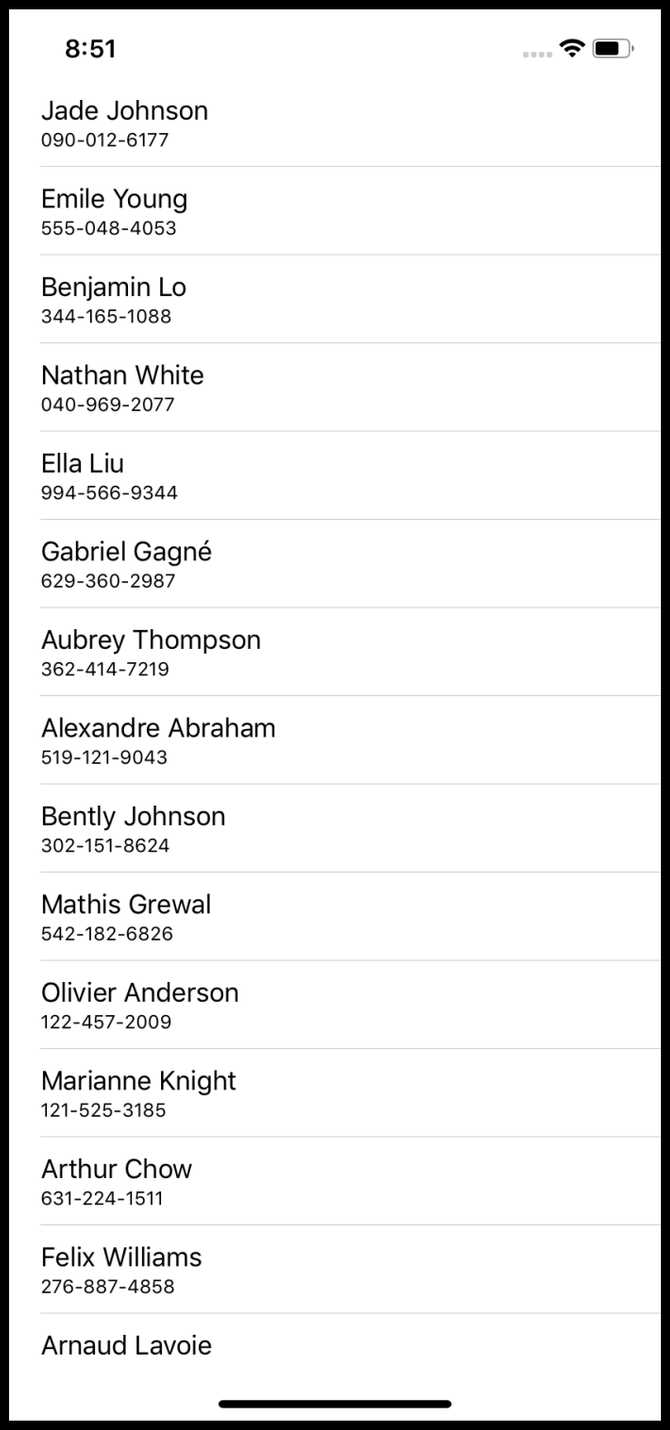
The Side Effects / Byproducts

- Units of code are short
- Data Models’ separation
- Units do a single (or very few) task(s)
- Dependencies are not hard-coded

“Talk is cheap. Show me the code!”

Linus Torvalds

The Bestest App in the World!



https://randomuser.me

WARNING: Low testabilitiness detected!

```
override func viewDidLoad() {
    super.viewDidLoad()
    guard let url = URL(string:
        "https://randomuser.me/api/?inc=name,phone&nat=ca&results=20") else {
        return }
    let networkCall = URLSession.shared.dataTask(with: url) { [weak self] (data,
        _, error) in
        guard let strongSelf = self,
            let unwrappedData = data,
            error == nil else {
                print("Something went wrong fetching users:
                    \(error.debugDescription)")
                return
        }

        let apiResponse = try? JSONDecoder().decode(ApiResponse.self, from:
            unwrappedData)
        if let unwrappedApiResponse = apiResponse {
            strongSelf.users = unwrappedApiResponse.results
            DispatchQueue.main.async {
                strongSelf.tableView.reloadData()
            }
        }
    }
    networkCall.resume()
}
```

The Side Effects / Byproducts

- Units of code are short ❌
- Data Models' separation ❌
- Units do a single (or very few) task(s) ❌
- Dependencies are not hard-coded ❌

Units of code are small

```
override func viewDidLoad() {
    super.viewDidLoad()
    fetchUsers(from: USERS_URL) { [weak self] users in
        self?.users = users
        DispatchQueue.main.async {
            self?.tableView.reloadData()
        }
    }
}
```

Data models’ separation

```
struct ApiName: Codable {
    let first: String
    let last: String
}

struct ApiUser: Codable {
    let name: ApiName
    let phone: String
}
```

```
struct ApiResponse: Codable {
    let results: [ApiUser]
}

struct User: Equatable{
    let name: String
    let phone: String
}
```

Units do a single (or very few) task(s)

```
func createUser(from apiUser: ApiUser) → User {
    let user = User(name: "\(apiUser.name.first.capitalized) \(apiUser.name.last.capitalized)",
        phone: apiUser.phone)
    return user
}
```

Dependencies are not hard-coded

```
func fetchUsers(from apiUrlString: String,
                using session: URLSession = URLSession.shared,
                completionHandler: @escaping FetchUsersCompletionHandler) {
    guard let url = URL(string: apiUrlString) else { return }

    let networkCall = session.dataTask(with: url) { (data, _ , error) in
        // Make sure everything went well with the network fetch
        guard let unwrappedData = data,
            error == nil else {
                completionHandler([User]()) // something went wrong! return an empty array
                return }

        // Decode retrived data
        let apiResponse = try? JSONDecoder().decode(ApiResponse.self, from: unwrappedData)
        if let unwrappedApiResponse = apiResponse {
            completionHandler(unwrappedApiResponse.results.map({ User(fromNetworkUser: $0) }))
        }
    }
    networkCall.resume()
}
```

Amazing! Can we test something now?

createUser(from:)

```
func testUserGenerationFromApiUser() {
    let apiUser1 = ApiUser(name: ApiName(first: "John", last: "doe"), phone: "1234567890")
    let apiUser2 = ApiUser(name: ApiName(first: "jane", last: ""), phone: "12345678901 23")
    let apiUser3 = ApiUser(name: ApiName(first: "Jimmy 🎸", last: "Hendrix"), phone: "N/A")

    let user1 = createUser(from: apiUser1)
    let user2 = createUser(from: apiUser2)
    let user3 = createUser(from: apiUser3)

    XCTAssert(user1.name == "John Doe")
    XCTAssert(user1.phone == "1234567890")

    XCTAssert(user2.name == "Jane ")
    XCTAssert(user2.phone == "12345678901 23")

    XCTAssert(user3.name == "Jimmy 🎸 Hendrix")
    XCTAssert(user3.phone == "N/A")
}
```

fetchUsers(from:using:completionHandler:)

```
class MockURLSessionDataTask: URLSessionDataTask {
    private let closure: () → Void

    init(withClosure closure: @escaping () → Void) {
        self.closure = closure
    }

    override func resume() {
        closure()
    }
}

class MockURLSession: URLSession {
    typealias CompletionHandler = (Data?, URLResponse?, Error?) → Void

    var data: Data?
    var error: Error?

    override func dataTask(with url: URL, completionHandler: @escaping
        CompletionHandler) → URLSessionDataTask {
        let data = self.data
        let error = self.error

        return MockURLSessionDataTask {
            completionHandler(data, nil, error)
        }
    }
}

class MockError: Error { }
```

fetchUsers(from:using:completionHandler:)

```
func mocks() → (MockURLSession, [User], Data, Error) {
    let mockSession = MockURLSession()
    let mockUsers = [User(name: "John Doe", phone: "123-456-7890"),
                     User(name: "Jane Doe", phone: "321-654-0987")]
    let mockApiUsers = [ApiUser(name: ApiName(first: "John", last: "Doe"), phone:
        "123-456-7890"),
                        ApiUser(name: ApiName(first: "Jane", last: "Doe"), phone:
                            "321-654-0987")]
    let mockResponse = ApiResponse(results: mockApiUsers)
    let mockResponseData = try! JSONEncoder().encode(mockResponse)
    let mockError = MockError()

    return (mockSession, mockUsers, mockResponseData, mockError)
}
```

fetchUsers(from:using:completionHandler:)

```swift
func testFetchUsersValidUrlSuccess() {
    let (mockSession, mockUsers, mockResponseData, _) = mocks()

    mockSession.data = mockResponseData
    mockSession.error = nil

    fetchUsers(from: "https://google.ca", using: mockSession) { (users) in
        XCTAssert(users == mockUsers)
    }
}
```

```swift
func testFetchUsersValidUrlFailure() {
    let (mockSession, _ , _ , mockError) = mocks()

    mockSession.error = mockError

    fetchUsers(from: "https://google.ca", using: mockSession) { (users) in
        XCTAssert(users.count == 0)
    }
}
```

The Side Effects / Byproducts

- Units of code are short
- Data Models’ separation
- Units do a single (or very few) task(s)
- Dependencies are not hard-coded

One more thing…

Maintainable code is naturally testable

Thank You! 🙏

@marwanalany