TDD + BDD = TBD

Триада инженерной культуры

Максим Абакумов
Ярослав Магин

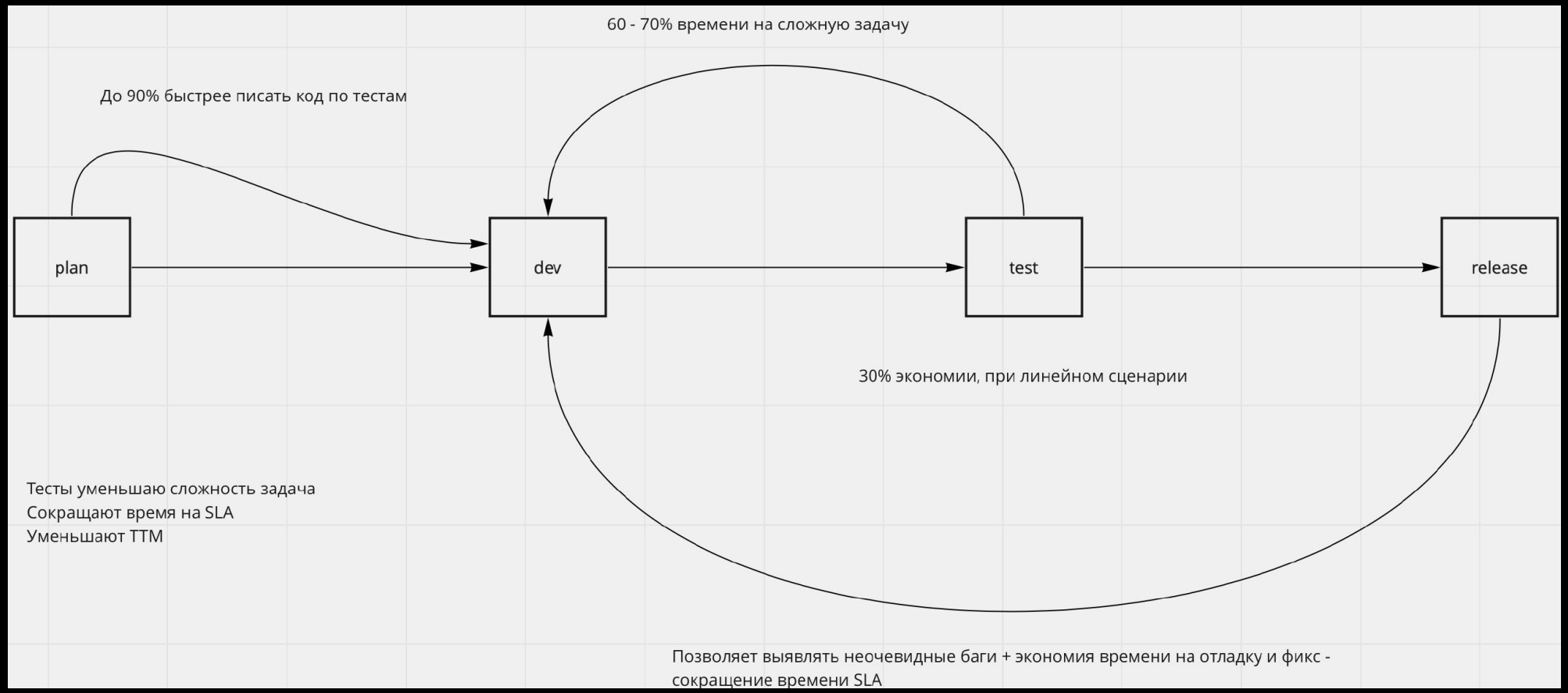
DELIVERY CLUB TECH

План

1. Что такое инженерная культура и причем тут тесты
2. Гигиена юнит-тестов
3. Как готовить BDD
4. Как готовить TDD
5. Резюме

Что если не дают писать тесты?

Время – деньги



Регресс без тестов - деньги на ветер

Unit test coverage - до 5%

Unit test coverage - от 10% и выше

Количество	Проценты
63	24%
34	13%
32	12%
22	9%
16	6%
16	6%
14	5%
14	5%
13	5%
11	4%
7	3%

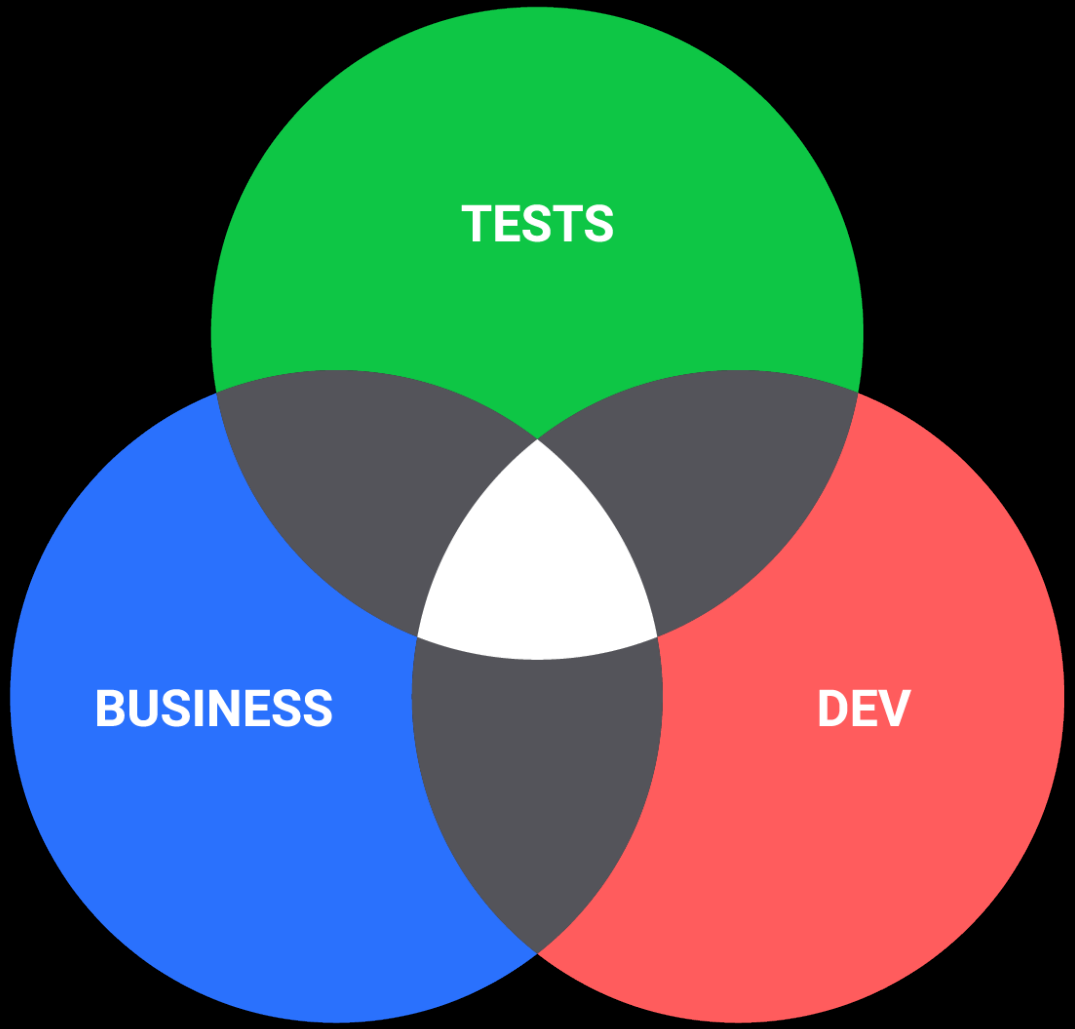
Почему доклад называется TBD?

Чаще всего тесты – то, что максимально откладывается на потом и пишется, в лучшем случае, в рамках техдолга.
Мы называем это To-Be-Done

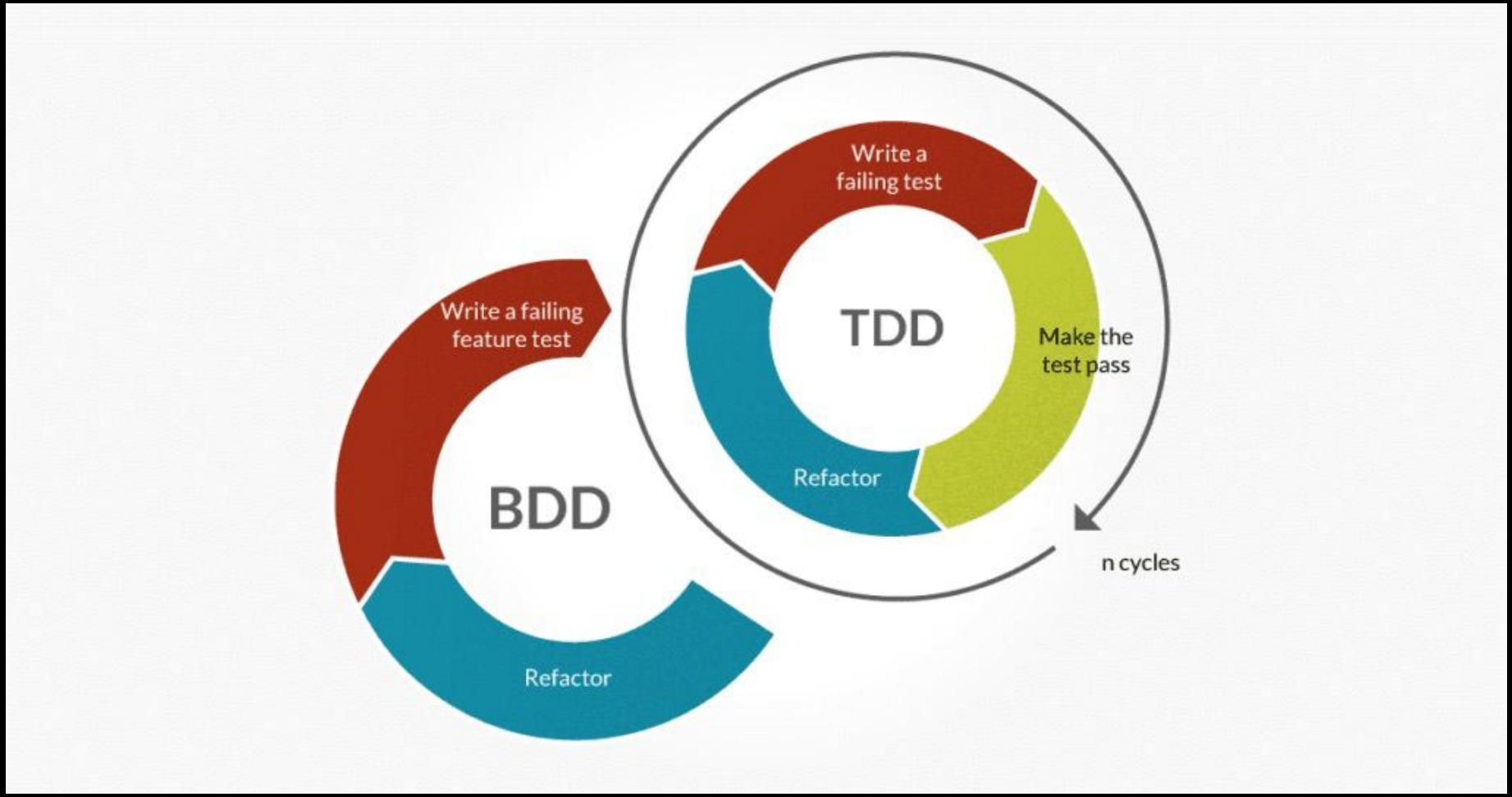
Наша интерпретация – Tests, Business, Development. Тесты помогают подружиться бизнесу и разработке.

Тесты - не «нагрузка», а брокер между постоянно уточняющимися требованиями и технической реализацией.

Инженерная культура



Что выбрать – TDD или BDD?



Инструментарий в Delivery Club

- Sourcery для генерации моков (еще есть SwiftyMocky)
- Quick & Nimble для behavior testing
- Code Coverage в Xcode
- Комбинация TDD и BDD подходов

Гигиена unit-тестов

Правило 3A

```
class RestaurantsTestCase: XCTestCase {

    func testOpenRestaurant_HasMenu() {
        // Arrange
        let restaurant = Restaurant(vendorId: 111, name: "Aeroport")

        // Act
        restaurant.loadMenu()

        // Assert
        XCTAssertEqual(restaurant.hasMenu, true)
    }
}
```

… оно же Given-When-Then

```
describe("Загрузка меню на старте") {
    var restaurant: Restaurant!
    beforeEach {
        restaurant = Restaurant(vendorId: 111, name: "Aeroport") // Given
    }
    context("в случае успешной загрузки") {
        beforeEach {
            restaurant.loadMenu() // When
        }
        it("меню должно существовать") {
            expect(restaurant.hasMenu).to(beTrue()) // Then
        }
    }
}
```

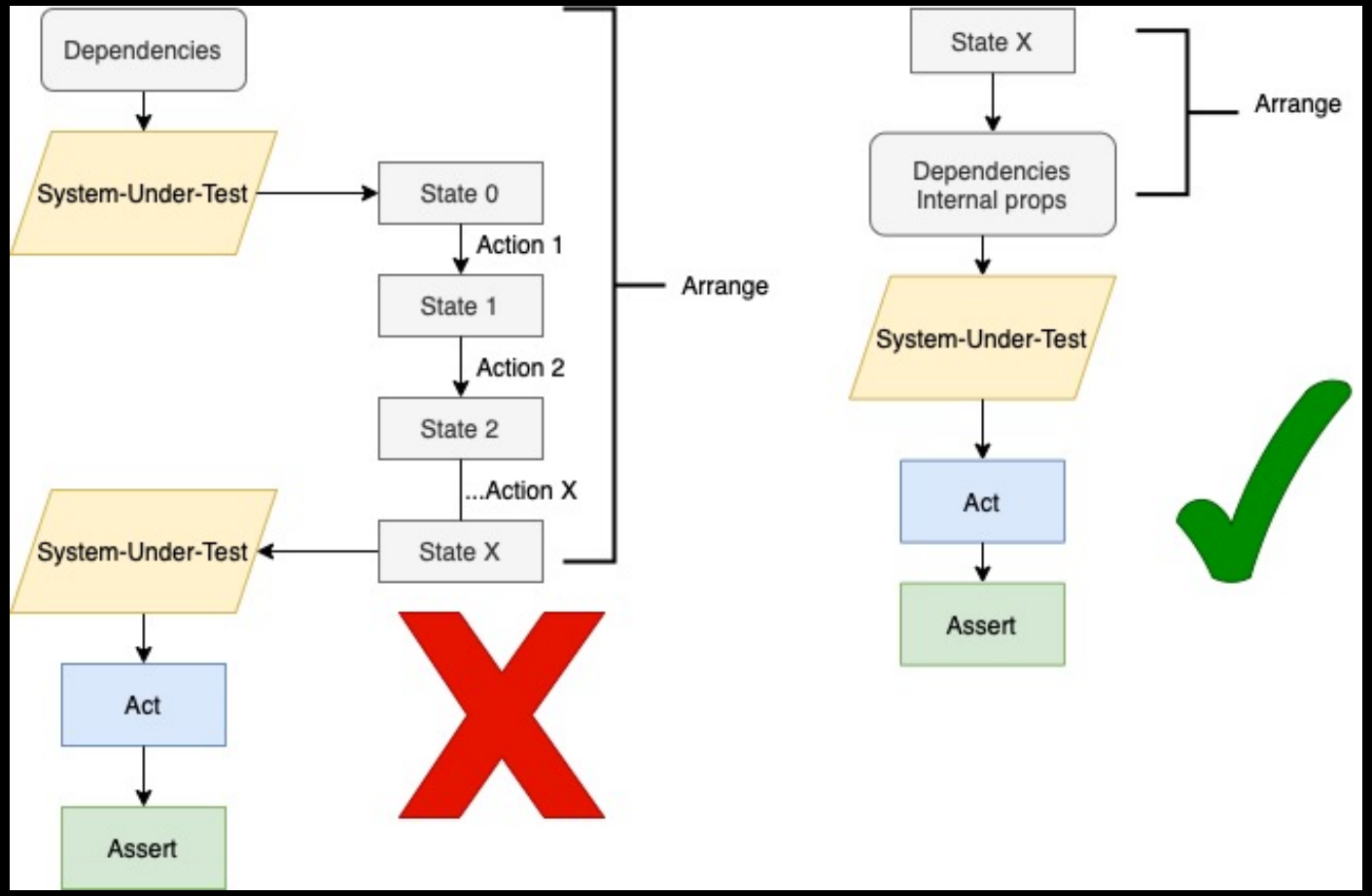
Что не так с тестами?

Самые распространенные проблемы:

- существуют «для галочки»
- «апрув не глядя» на code review
- часто ломаются и тяжело поддерживаются.

Решение – к тестовому коду могут (**и должны**) применяться те же принципы, что и к продуктовому!

Рекомендация №1 – Arrange по щелчку пальцев



Рекомендация №2 – не используйте XCTAssertTrue

XCTAssertTrue(result == 5)	XCTAssertEqual(result, 5)
XCTAssertTrue failed	**XCTAssertEqual failed: ("0") is not equal to ("5")**

Рекомендация №3 – не злоупотреблять setUp

```
class RestaurantsTestCase: XCTestCase {
    var restaurant: Restaurant!

    override func setUp() {
        super.setUp()
        restaurant = Restaurant(vendorId: 111, name: "Aeroport")
    }

    func testOpenRestaurant_HasMenu() {
        restaurant.loadMenu()
        XCTAssertEqual(restaurant.hasMenu, true)
    }
}
```

Выход – вспомогательный фабричный метод

```
class RestaurantsTestCase: XCTestCase {
    func testRestaurant_HasExtraMenuForDcPro() {
        let restaurant = createDefaultRestaurant()
        restaurant.loadSubscriptionInfo(for: "dcpro")
        XCTAssertEqual(restaurant.hasExtraMenu, true)
    }

    private func createDefaultRestaurant(
        vendorId: Int = 111,
        name: String = "Aeroport"
    ) -> Restaurant {
        return Restaurant(vendorId: vendorId, name: name)
    }
}
```

Рекомендация №4 – custom assertions

```
func testVkusnoITochka_HasExtraMenuForNonDcPro() {
    let vkusnoITochkaVendorId = 355
    let restaurant = createDefaultRestaurant(vendorId: vkusnoITochkaVendorId)

    restaurant.loadSubscriptionInfo(for: "non-dcpro")

    assertRestaurantHasSpecialDcProMenu(restaurant)
}

private func assertRestaurantHasSpecialDcProMenu(_ restaurant: Restaurant) {
    XCTAssertEqual(restaurant.hasExtraMenu, true)
}
```

Как готовить BDD

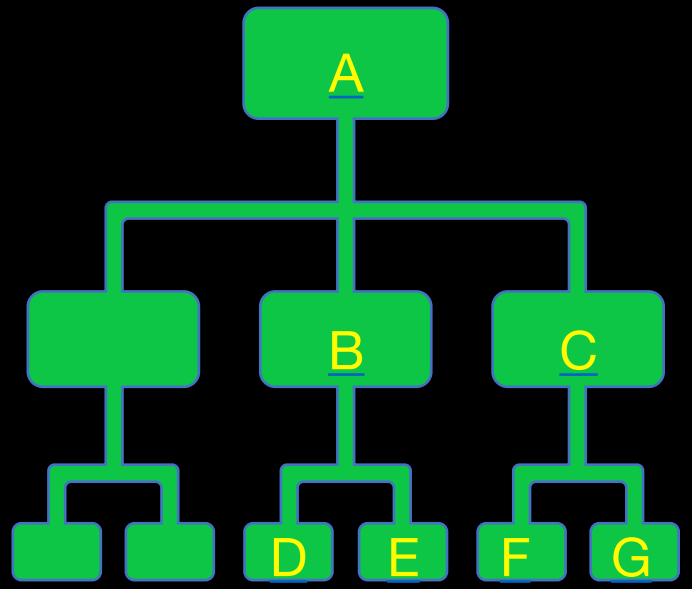
«Кто ясно мыслит, ясно излагает»

Названия тест-кейсов == формулировки бизнес-требований

- Уточнение требований в процессе - вечная история. 3Amigo
- Оформление тест-кейсов в behaviour тестах = знание ТЗ
- Входные данные / модели / инпуты == Given
- «Когда» и «если» == When
- «Тогда», «ожидается результат», «должно быть…» == Then

Quick - дерево

Запуск тест-кейса == обход дерева в глубину



describe: что / с какими входными данными

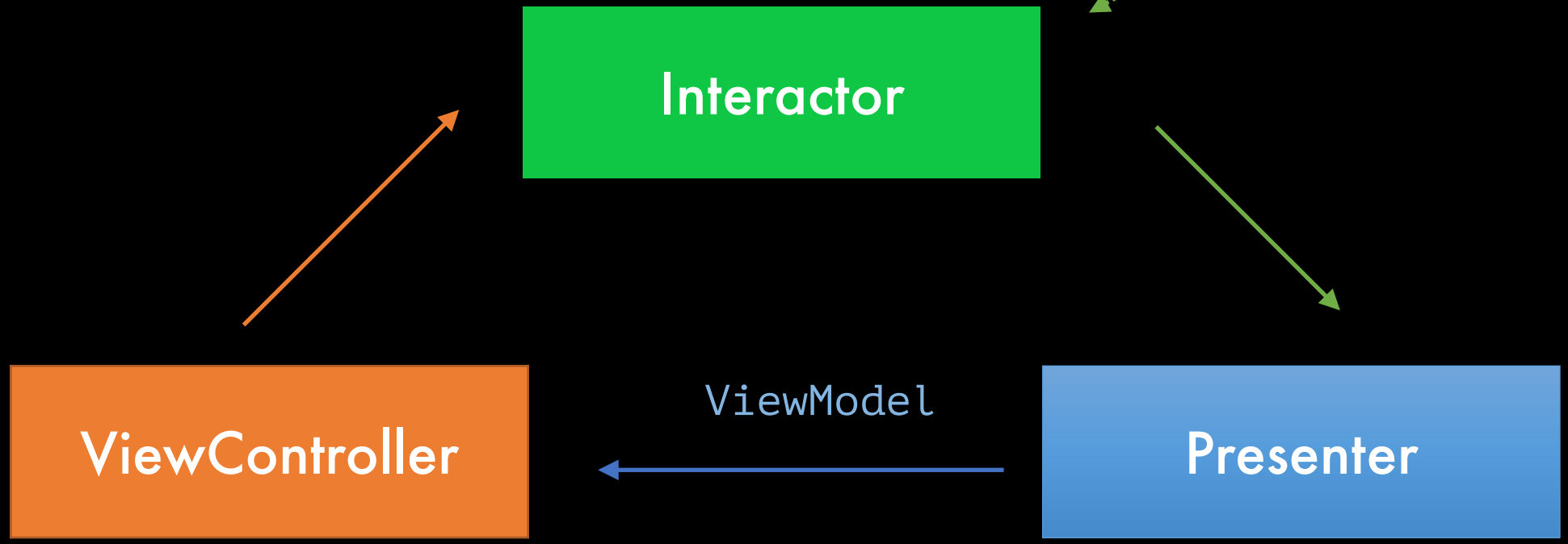
context: когда / если / в случае

it: тогда / в результате / (не) должно быть

beforeEach / afterEach: внутри spec(), describe, context == setUp / tearDown

Архитектура модуля VIP

View - Interactor - Presenter. Clean Swift



Разработка сценариев

Given - When - Then [состояние, поведение - контекст, результат]

Given [запускаем модуль]

When [у клиента есть специальные предложения]

Then [отображаем предложения]

Given [модуль на экране]

When [надо добавить предложений, обновить состояние]

Then [отображаем новые предложения, если нет ошибки]

Разработка интерфейса

Given - When - Then [arrange , act, assert: тест-кейс(ы)]

Given [init] - **приводим в нужное состояние**

When [–] - **под капотом вызывается** setup()

Then [**проверяем подписку на метод зависимости** (mock)]

Given [viewDidLoad]

When [**вызов** viewDidLoad **сам по себе задает начальный контекст**]

Then [**проверяем, что презентер отобразит полученные данные**]

Разработка интерфейса: код

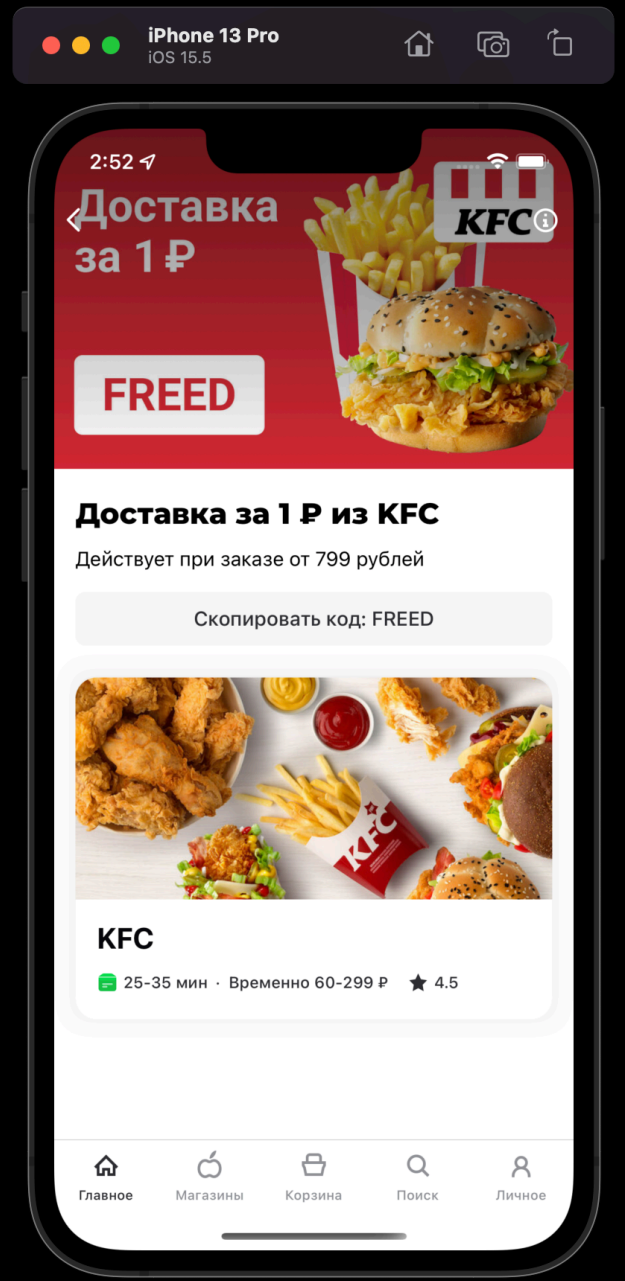
Начало

```
final class SelectionsInteractor {
    init(router: SelectionsRouterProtocol,
         presenter: SelectionsPresenterProtocol,
         repository: SelectionsRepositoryProtocol,
         input: SelectionsInput,
         favoritesManager: FavoritesManagerProtocol,
         analytics: AnalyticsAdapterProtocol,
         checkoutManager: RestaurantsCheckoutManagerProtocol)
}

extension SelectionsInteractor : SelectionsInteractorProtocol {
    func viewDidLoad()
    func willDisplayLoadingView()
    func didTapService(serviceId: Int)
    func didTapBack()
    func didTapInfo()
}
```

Разработка интерфейса: код

Given - When - Then [arrange , act, assert: тест-кейс(ы)]

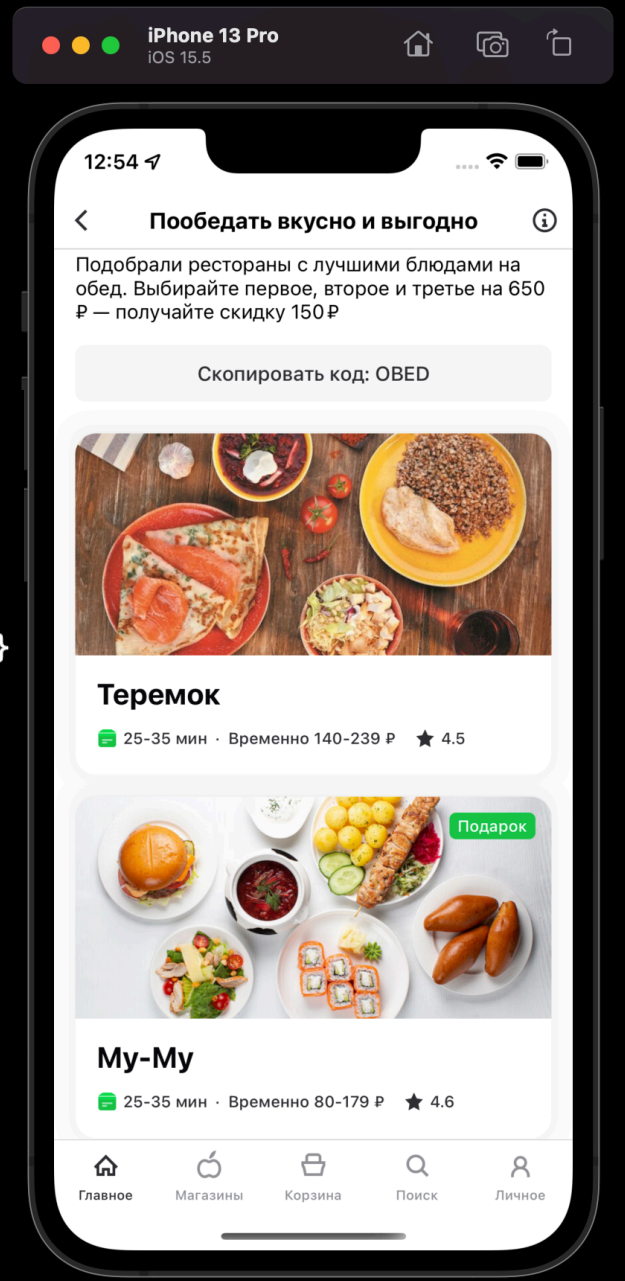
```
describe("init") {
    it("подписались на обновления favorites") {}
}

describe("viewDidLoad") {
    it("в случае успеха презентер показывает данные") {}
}

extension SelectionsInteractor: FavoritesManagerObserver {

    func didTapPromocode() {}
    func didUpdateFavorites() {}
    func didAddFavorite(identifier: Int) {}
    func didRemoveFavorite(identifier: Int) {}
```



Разработка интерфейса: код

Тестирование внутренней логики

```
describe("init") {
    it("подписались на обновления favorites") { /* Nimble matchers */ }
}

describe("viewDidLoad") {
    it("в случае успеха презентер показывает данные") { /* Nimble matchers */ }
}

// тестирование внутренней логики - вводим приватное свойство canLoadMore
describe("willDisplayLoadingView") {
    beforeEach {
        subject.viewDidLoad()
    }
    context("когда canLoadMore == true") {
        it("презентер покажет состояние загрузки") { /* Nimble matchers */ }
    }
}
```



Разработка интерфейса: код

Тестирование внутренней логики: имплементация

```
extension SelectionsInteractor: SelectionsInteractorProtocol {
    func viewDidLoad() {
        presenter.showLoading(true)
        loadSelections()
    }

    func willDisplayLoadingView() {
        guard canLoadMore, let selectionsData = selectionsData else { return }
        canLoadMore = false
        loadSelections(offset: selectionsData.services.count)
    }
```

Разработка интерфейса: код

Тестирование внутренней логики: интерактивная часть сценария

```
// моделирование входных данных в реализации через инпут
describe("didTapService") {
    context("когда кликнули по ячейке с каким-то serviceId") {
        it("в зависимости от categoryId сервиса, открываем или ресторан, или магазин") {
        }
    }
}
                private func makeSelectionsData() -> SelectionsData {
                    let service = DCServiceObject()
                    service.serviceId = 1
                    let id = DCServicesCategory(rawValue: 1)!
                    service.categoryId = id
                    return .init(
                        selection: DCSelectionObject(),
                        services: [service],
```

Разработка интерфейса: код

Тестирование внутренней логики: интерактивная часть сценария

```
func didTapService(serviceId: Int) {
    guard let service = selectionsData?.services.firstVendor(serviceId),
          let model = ServiceModel(service)
    else { return }

    switch model {
    case .restaurant(let restaurantService, _):
        router.openRestaurant(service: restaurantService)
    case .store(let store):
        router.openStore(store)
    }

    trackVendorClick(service: service)
}
```

Что я делаю не так?

Крокодил не ловится, coverage не растет

Test Readability

Coverage: не растет

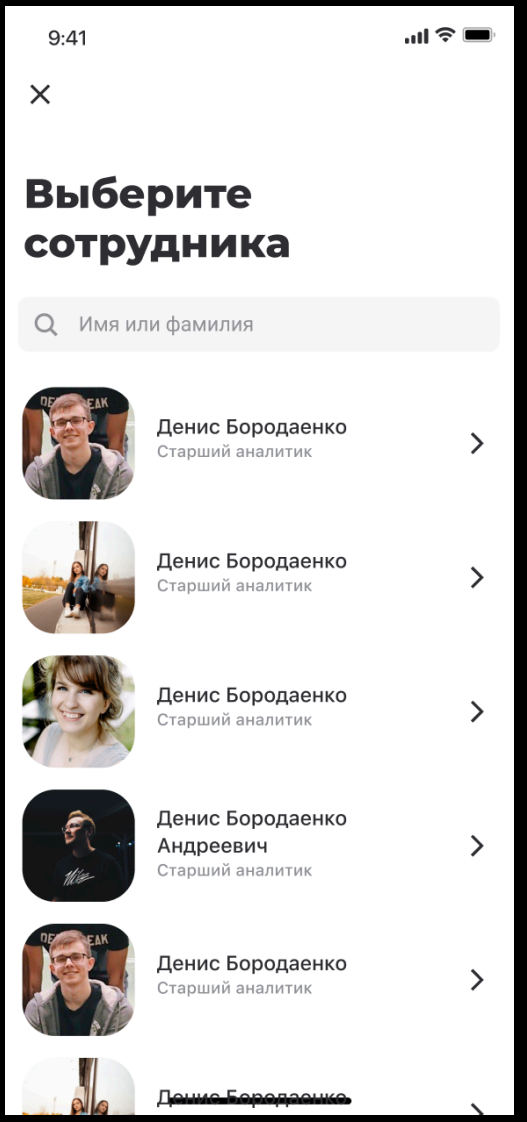
Coverage: достаточно %%?

Нетестируемое

Как готовить TDD

Постановка требований

- Пагинация
- Загрузка списка официантов
- Фильтрация



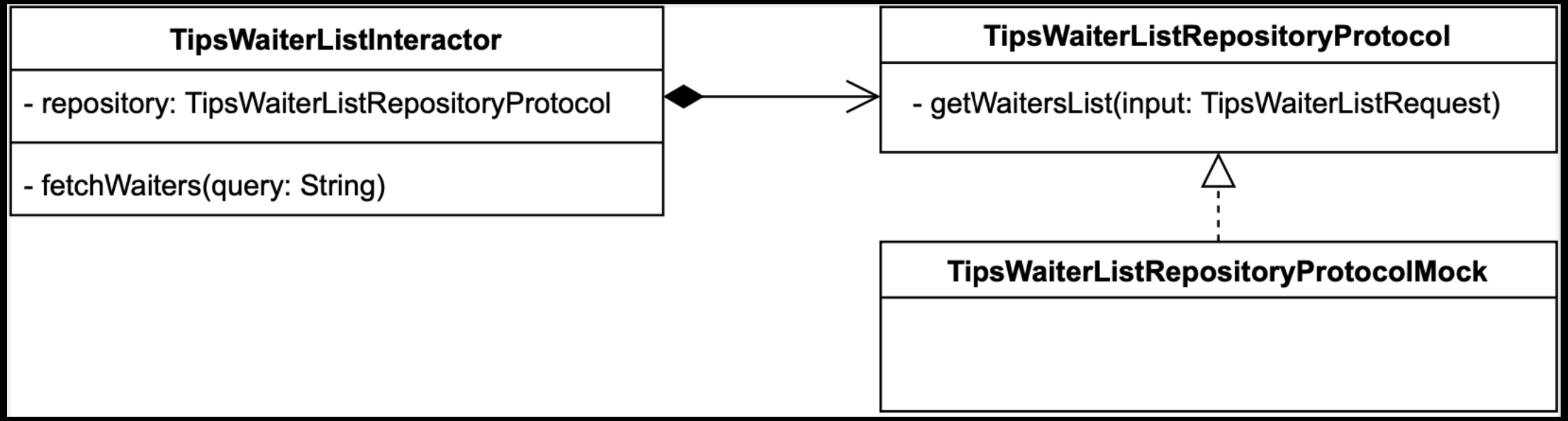
Пагинация: тест

```
func testCreateWaitersWithPaginationRequest() {
    interactor = createInteractor()
    interactor.waiters = defaultWaiters

    let request = interactor.createWaitersRequest()

    XCTAssertEqual(request.from, defaultWaiters.count)
    XCTAssertEqual(request.query, nil)
}
```

Пагинация: имплементация

```
// MARK: - Internal tested methods
extension TipsWaiterListInteractor {
    func createWaitersRequest(query: String? = nil) ->
      TipsWaiterList.Request {
        return TipsWaiterList.Request(
            orgUid: input.orgUid,
            query: query,
            from: waiters.count,
            count: .pageCount
        )
    }
}
```

Загрузка официантов: внедрение mock-а



Загрузка официантов: тест на ошибку

```
func testRequestWaitersWithError() async {
    setupWaitersListRepositoryWithFail()
    interactor = createInteractor()

    let result = await interactor.fetchWaiters()

    XCTAssertNil(result)
}
```

Резюме

- Тесты – чистый бизнес. Покрытый качественными тестами код – меньше багов на регрессе и затрат на QA
- TDD/BDD – взаимодополняющие друг друга подходы, отлично уживающиеся в одном проекте (bottom-top, top-bottom)
- Если тесты непонятны разработке и не отвечают требованиям бизнеса – их проще сразу выкинуть

Полезные книги

Ссылки

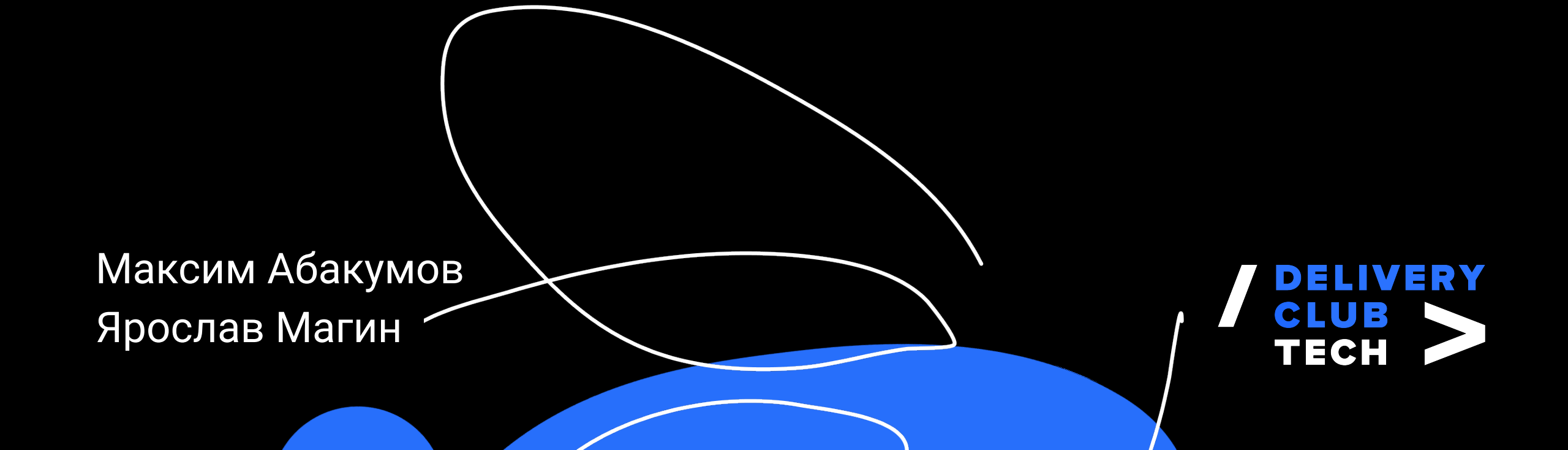
https://youtu.be/amVstam84Xo Mobius 2017 Piter,
J.Sundell — Writing Swift code with great testability

https://habr.com/ru/post/352694/ очень живенький туториал по Quick/Nimble

Коллекция занимательных кастомных матчеров

Спасибо за внимание

Максим Абакумов
Ярослав Магин



DELIVERY CLUB TECH