6. Golovkina M.V. Periodic semiconductor structures with metamaterials. *International Siberian Conference on Control and Communications, SIBCON-2009*, 2009, pp. 133-137. DOI: 10.1109/SIBCON.2009.5044843.
7. Golovkina M.V. Electromagnetic wave propagation in a multilayered periodic structure containing negative index material. *IEEE International Siberian Conference on Control and Communi-cations, SIBCON-2007*, 2007, pp. 174-178. DOI: 10.1109/SIBCON.2007.371320.
8. Soyfer V.A. *Difrakzionnaia optika i nanofotonika.* [Diffractive optics and nanophotonics]. Moscow, Fizmatlit Publ., 2014, 608 p.
9. Negari A.B.H., Jauregui D., Hernandez J.M.S., Mina D.G., King B. J., Idehenre I., Powers P.E., Hansen K.M., Haus J.W. Phase sensitive signal analysis for bi-tapered optical fibers. *Proceedings SPIE, Biophysics, Biology, and Biophotonics: the Crossroads*, 2016, vol. 9719, pp. 971907. DOI: 10.1117/12.2208934.
10. Ri-Qing Lv, Qi Wang, Bo-Tao Wang, Yu Liu, Lingxin Kong. Polymer microfiber bridging Bi-tapered refractive index sensor based on evanescent field. *Optics Communications*. 2018. V. 414. P. 134-139. DOI: 10.1016/j.optcom.2017.12.063.

Received 15.11.2018

УДК 004.053: 004.054

МЕТОДИКА РАЗРАБОТКИ АРХИТЕКТУРНЫХ КОМПОНЕНТОВ iOS ПРИЛОЖЕНИЯ «ГОСУСЛУГИ МОСКВЫ» ЧЕРЕЗ ТЕСТИРОВАНИЕ

Богомолова М.А., Клементьев С.А.
Поволжский государственный университет телекоммуникаций и информатики, Самара, РФ
E-mail: bogomolova-ma@psuti.ru

Представлена методика тестирования модулей мобильного приложения на платформе iOS «Госуслуги Москвы» через тестирование. Предложенная методика ориентирована на архитектуру приложения iOS «Госуслуги Москвы» и может применяться при разработке сцен приложения. На примере разработки сцены регистрации социальной карты москвича, используя разработанную методику, по техническому заданию поэтапно реализованы три архитектурных компонента View Controller, Interactor и Presenter: сцены в виде классов, а также реализованы тесты модулей для этих классов. Написанный код показывает особенности разработки и может использоваться в качестве примера для создания новых сцен. Использование методики модульного тестирования мобильного приложения iOS «Госуслуги Москвы» позволяет уменьшить количество дефектов в приложении, уменьшает стоимость исправления дефектов, способствует написанию более чистого кода и проектированию удобных интерфейсов классов.

Ключевые слова: *модульное тестирование, unit-тестирование, test-driven development, TDD, Clean Swift, mock-объект, spy-объект, объект-шпион*

Введение

В настоящее время осуществляется проект разработки мобильного приложения iOS «Госуслуги Москвы», объединяющего востребованные услуги Правительства Москвы. На данный момент проект содержит более чем 1500 файлов с исходным кодом. Кроме того, постепенно в приложение добавляется новая функциональность, вследствие чего увеличивается количество файлов и меняется исходный код, в нем с большой вероятностью появляются регрессионные ошибки (когда работающий функционал сломали неаккуратными изменениями). Действенным решением этой проблемы является модульное тестирование (unit-тестирование) – одна из ключевых практик методологии экстремального программирования. Здесь unit (модуль) – это малый по объему самодостаточный участок кода, реализующий определенное поведение, который часто (но не всегда) является классом.

Модульное тестирование заключается в изолированной проверке на корректность и работоспособность каждого отдельного модуля исходного кода – тестирование с высокой гранулярностью. То есть идея состоит в том, чтобы создавать тесты для каждой нетривиальной функции или метода [1]. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже протестированных местах программы, а также облегчает их обнаружение и устранение.

В Unit-тестировании используется два подхода к созданию тестов:

– разработка через тестирование (test-driven development, TDD);

– создание тестов для готовых методов.

В данном исследовании применен первый подход. Рассмотрим предлагаемую методику разработки архитектурных компонентов мобильного приложения iOS «Госуслуги Москвы» через тестирование. Эта техника основана на итеративном цикле разработки: написание теста, покрывающего желаемое изменение, затем написание кода, который позволит пройти тест, далее рефакторинг нового кода к соответствующим стандартам. Так формируется устройство классов и модулей, в то время как все изменения в коде направляют модульные тесты [2-3].

Для написания исходного кода и кода тестов модулей выбран язык программирования Swift, применяется встроенная библиотека XCTest (предназначена для создания и запуска модульных тестов производительности и тестов пользовательского интерфейса), поддерживаемая средой Xcode. В Xcode все создаваемые приложения возможно тестировать без необходимости загрузки программ на реальные устройства.

Архитектура приложения

Предлагаемая методика ориентирована на архитектуру приложения iOS «Госуслуги Москвы» и может применяться при разработке сцен приложения. На рисунке 1 изображены архитектурные модули приложения. Большинство экранов приложения реализовано с использованием архитектуры Clean Swift.

Рассмотрим VIP цикл в архитектуре Clean Swift. View Controller, Interactor, Presenter – это три основных компонента сцены в CleanSwift, каждый из которых ответственен за выполнение своей задачи. Interactor содержит бизнес-логику и выполняет работу над данными, совершает сетевые запросы. Presenter осуществляет форматирование данных. View Controller работает с графическим интерфейсом и отображает данные пользователю.

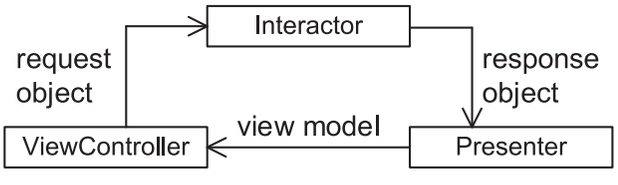

Рисунок 1. Архитектура Clean Swift

Типичный сценарий при этом включает следующие действия.

1. Пользователь нажимает на кнопку в пользовательском интерфейсе приложения.

2. Жест нажатия попадает в View Controller через IBAction.

3. View Controller создает объект запроса (request object) и отправляет его в Interactor.

4. Interactor получает объект запроса, выполняет работу, создает объект ответа, помещает в него результат работы, отправляет объект ответа (response object) в Presenter.

5. Presenter получает объект ответа, форматирует результаты, помещает результаты в модель отображения (view model), отправляет модель отображения компоненту View Controller.

6. View Controller отображает результат пользователю.

Блок-схема алгоритма методики

Для описания разработанной методики использована блок-схема алгоритма действий, выполняемых программистом при разработке сцены приложения (см. рисунок 2).

В блоках 2-17 описана последовательность действий подготовки классов, необходимых для написания кода тестов.

В блоках 2-4 создаются три класса, являющиеся архитектурными модулями сцены: (Имя сцены) Interactor, (Имя сцены) View Controller, (Имя сцены) Presenter Spy.

В блоке 5 архитектурные модули связываются между собой через протоколы. Протоколы описывают, какие данные и сообщения могут передавать друг другу архитектурные модули. Класс (Имя сцены) Interactor соответствует протоколу выхода (Имя сцены). В данном исследовании применен первый подход. Рассмотрим предлагаемую методику разработки архитектурных компонентов мобильного приложения iOS «Госуслуги Москвы» через тестирование.

В модуле View Controller класс (Имя сцены) соответствует протоколу выхода (Имя сцены) Presenter. Класс (Имя сцены) Presenter соответствует протоколу выхода (Имя сцены) Interactor.

В блоках 6-8 создаются классы, содержащие тесты. Один класс тестов соответствует одному архитектурному модулю. Например, класс (Имя сцены) PresenterTests содержит тесты для модуля (Имя сцены) Presenter.

В блоках 9-11 определяются классы-шпионы (Spy) в каждом классе, содержащем тесты. Так как архитектурные модули соединены между собой, используя объекты-шпионы, можно отслеживать данные, передаваемые между модулями. В классе (Имя сцены) Interactor Tests объявляется класс шпион (Имя сцены) Presenter Spy. В классе (Имя сцены) PresenterTests объявляется класс-

шпион (Имя сцены) View Controller Spy. В классе (Имя сцены)View Controller Tests объявляется класс-шпион (Имя сцены) Interactor Spy.

В блоках 12-14 в классах тестов создаются ссылки типа тестируемых объектов, то есть класс (Имя сцены) PresenterTests класс будет содержать ссылку на (Имя сцены) Presenter.

В блоках 15-17 описываются методы Setup (Имя сцены) Presenter, Setup (Имя сцены) View Controller, Setup (Имя сцены) Interactor. Методы предназначены для базовой настройки тестовых объектов и вызываются перед запуском тестовых методов. Это позволяет не загромождать код самих тестов и избавляет от повторения кода.

В данных методах происходит создание тестируемого объекта архитектурного модуля, присвоение ссылок, созданных в пунктах 12-14 на созданный объект. Создается объект-шпион и устанавливается выходом для объекта архитектурного модуля. Возможны и другие настройки специфичные для объекта.

В блоке 18 происходит выбор одного из архитектурных модулей для реализации. Порядок реализации не принципиален.

Блоки 19-33 содержат цикл разработки тестов и рабочего функционала для выбранного архитектурного модуля.

Алгоритм считается завершенным, когда для всех трех модулей реализован полный набор тестов и рабочий код, который дает положительный результат прохождения тестов.

Разработка сцены приложения регистрации социальной карты москвича

С целью демонстрации принципов работы разработанной методики описан процесс разработки сцены приложения и ее покрытия тестами. Разработанная сцена предназначена для регистрации пользователем социальной карты москвича (СКМ) (см. рисунок 3).

Рассмотрим пользовательский сценарий. Предусловие: пользователь авторизован в приложении.

1. Пользователь на экране услуги СКМ инициирует регистрацию СКМ.

2. Клиент отображает экран регистрации СКМ: номер телефона по умолчанию заполнен номером телефона из профиля пользователя и недоступен для редактирования.

3. Пользователь вводит в поле номер и серию карты одним из следующих способов:

– пользователь вводит вручную (в полиграфическом виде): клиент отображает введенный номер на экране и преобразует его в канонический вид для отправки в запросе;

– пользователь сканирует карту: клиент получает номер в каноническом виде, преобразует его в полиграфический вид и отображает в поле ввода на экране.

4. Пользователь продолжает процесс регистрации;

5. Клиент осуществляет запрос методом registrCard (зарегистрировать карту) и обрабатывает ответ сервера:

– если регистрация прошла успешно, переход на следующий шаг;

– если сервер вернул ошибку регистрации, клиент отображает диалоговое окно с сообщением об ошибке.

6. Отображает главный экран услуги СКМ с данными привязанной карты.

Завершение сценария.

Модуль Interactor должен содержать следующую функциональность:

– загружать номер телефона пользователя из профиля пользователя;

– преобразовывать номер пользователя в канонический вид из полиграфического вида;

– осуществлять запрос на регистрацию карты.

Разработан шаблон класса CardRegistrationInteractorTests, содержащий следующие тестовые методы:

```
class CardRegistrationInteractorTests:
XCTestCase
{  // MARK: Subject under test
  var sut: GUCardRegistrationBusinessLogic!
   // MARK: Test lifecycle
  override func setUp() {
      super.setUp()
      setupCardRegistrationInteractor()
   }
  override func tearDown() {
      super.tearDown()
   }
  // MARK: Test setup
  func setupCardRegistrationInteractor() {
  }
}
```

Класс Card Registration Interactor Tests предназначен для тестирования рабочего класса GUCard Registration Interactor. Перед вызовом каждого тестового метода будет выполнен метод setUp. После того, как тестовый метод завершит работу, будет вызван метод tearDown. Метод

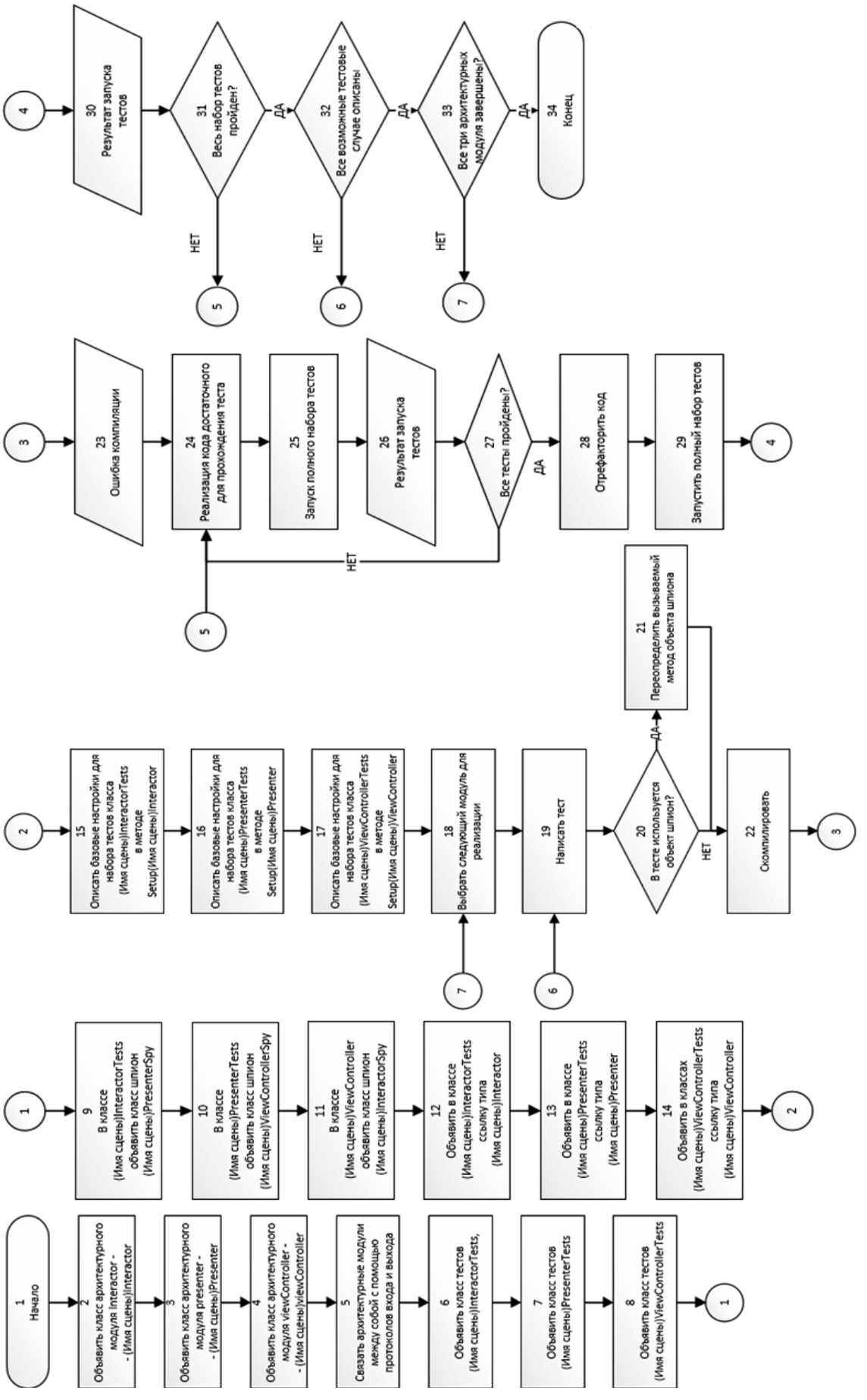

Рисунок 2. Блок-схема алгоритма методики

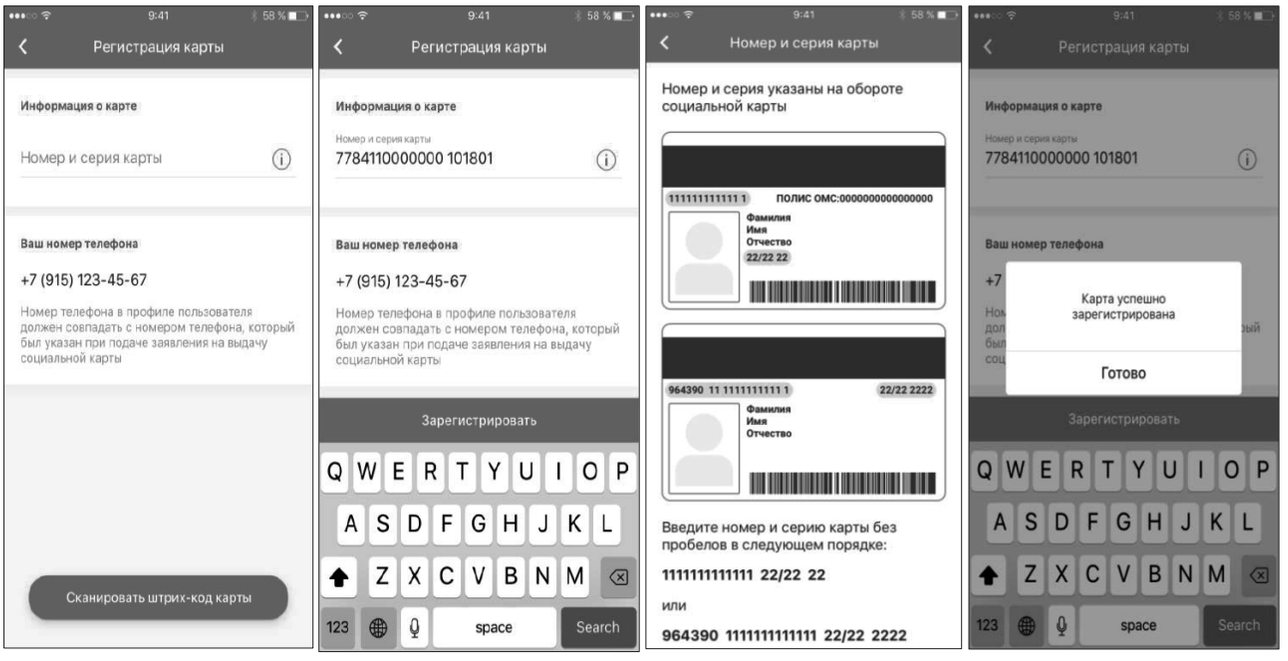

Рисунок 3. Экраны регистрации карты, ввода номера карты, информации о вводе номера СКМ, успешной регистрации

setup Card Registration Interactor () предназначен для инициализации начальных значений. Переменная sut (System under test) – это ссылка на экземпляр тестируемого объекта. Для реализации тестов необходимо симулировать контекст, в котором выполняются тестируемые функции, для этих случаев существует понятие mock-объектов (spy-объект), представляющих собой специально созданные сущности, единственная цель которых – это симуляция контекста, в котором выполняется тестируемая функция.

Моком называется тестовый двойник, способный запоминать аргументы вызовов своих методов, количество вызовов, часто также способный самостоятельно проводить проверки. Как правило, каркасы изоляции позволяют наделять моки дополнительным настраиваемым поведением вплоть до полной имитации работы класса с подменой одного небольшого аспекта [4].

Продемонстрируем использование mock-объектов и объектов-шпионов при тестировании модулей приложения. Далее отображены классы этих объектов.

В классе GU-Card Registration Presentation Logic Spy (используется для мониторинга результата работы тестируемого класса) переопределены базовые методы. Шпион отслеживает, были ли вызваны его методы, анализирует значения переданных параметров:

```
class GUCardRegistrationPresentationLogicSpy: GUCardRegistrationPresentationLogic {
    var presentCardRegisteredCalled = false
    var presentedErrorStringCalled = false
    var presentErrorCalled = false
    var presentPhoneCalled = false
    var displayModule: CommonDisplayLogic?

    func presentCardRegistered() {
        presentCardRegisteredCalled = true
    }

    func present(errorString: String, cancelTitle: String, completion: (() -> Void)?) {
        presentedErrorStringCalled = true
    }

    func presentError(error: Error, completion: (() -> Void)?) {
        presentErrorCalled = true
    }

    func present(phone: String) {
        presentPhoneCalled = true
    }
}
```

GUCardRegistrationServiceSpy эмулирует поведение класса GUCardRegistrationMainService:

```
class GUCardRegistrationServiceSpy: GUMSCSer-vice {
```

```
        func getBarCode(cardId: String, completion: @escaping ((GUResult<MSCGetBarCodeModel>) -> Void)) {}
                func getCard(completion: @escaping ((GURe-sult<GUMoscowSocialCard>) -> Void)) {}
                func verifyCard(order: String, phone: String, completion: @escaping ((GURe-sult<GUMscVerifyCardItem>) -> Void)) {}
        func removeCard(withId cardId: String, completion: @escaping ((Error?) -> Void)) {}
            func submitCard(cardSeries: String, cardNumber: String, completion: @escaping ((GURe-sult<GUMoscowSocialCard>) -> Void)) {
          if (cardNumber == «19110188» || cardSeries == «19110188») {
                  completion(.failure(NSError(domain: «as-dasd», code: 213, userInfo: nil)))
              }
      completion(.success(GUMoscowSocialCard()))
          }
       }
```

В соответствии с разработанной методикой для базовой настройки тестовых объектов используются методы с префиксом setup. Они вызываются перед запуском тестовых методов. Это позволяет не загромождать код самих тестов и избавляет от повторения кода. Разработаны примеры реализации методов инициализации для всех видов архитектурных модулей.

Инициализируем свойства Interactor объектами-шпионами:

```
    class                CardRegistrationInteractorTests: XCTestCase {
       // MARK: Subject under test

       var sut: GUCardRegistrationBusinessLogic!
      var presenter: GUCardRegistrationPresentation-LogicSpy!
       // MARK: Test lifecycle

       override func setUp() {
          super.setUp()
          setupCardRegistrationInteractor()
       }

    func setupCardRegistrationInteractor() {
       let interactor = GUCardRegistrationInteractor()
      presenter = GUCardRegistrationPresentationLo-gicSpy()
       interactor.presenter = presenter
     interactor.cardService=GUCardRegistrationSer-viceSpy()
            interactor.card = GUMSCCardRegistra-tion(canonical: «96439077030252000651 9110177»)
       sut = interactor
    }
```

Продемонстрируем реализацию тестовых методов для всех архитектурных модулей. В качестве примера приведем разработанные тесты для модуля Presenter:

```
    // MARK: Test doubles
    func testPhonePresented() {
       var phoneToDisplay = «21231»
       sut.present(phone: phoneToDisplay)
                XCTAs-sertTrue(viewControllerSpy.phoneDisplayed == phoneToDisplay)
    }

    func testCardRegisteredPresented() {
       sut.presentCardRegistered()
                XCTAs-sertTrue(viewControllerSpy.cardRegisteredDisplayed)
    }
```

Все unit-тесты выполняют непосредственную сверку полученных результатов с ожидаемыми при помощи предопределенных функций библиотеки тестирования, возвращающих положительный или отрицательный результат. Примером такой функции может служить XCTAssert – вариант проверяющей функции, успешно завершающейся, если входной логический параметр действителен.

Заключение

Использование методики модульного тестирования мобильного приложения iOS «Госуслуги Москвы» позволяет уменьшить количество дефектов в приложении, уменьшает стоимость исправления дефектов, способствует написанию более чистого кода и проектированию удобных интерфейсов классов. В процессе разработки было обнаружено, что из-за большого числа файлов в проекте набор тестов запускается недостаточно быстро, что замедляет скорость разработки. В перспективе следует произвести поиск возможных программных и аппаратных способов увеличения быстроты сборки проекта и запуска набора тестов.

Литература

1. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения. СПб.: Питер, 2012. – 608 с.
2. Бек К. Экстремальное программирование: разработка через тестирование. СПб.: Питер, 2003. – 224 с.
3. Beck K. Extreme Programming Explained: Embrace Change. Boston: Addison Wesley, 2004. – 224 p.
4. Osherove R. The Art of Unit Testing with Examples in .NET. Greenwich: Manning, 2009. – 324 с.
5. Meszaros G. xUnit Test Patterns. Refactoring Test Patterns. Boston: Addison-Wesley, 2007. – 948 с.
6. Симан М. Внедрение зависимостей в .NET. Спб.: Питер, 2013. – 464 с.
7. Фаулер М. Шаблоны корпоративных приложений. М.: Вильямс, 2011. – 544 с.
8. Фаулер М. Рефакторинг. Улучшение существующего кода. СПб.: Символ-плюс, 2010. – 432 с.
9. Astels D. Test Driven Development: A Practical Guide, Upper Saddle River. NJ: Prentice Hall PTR, 2003. – 592 p.
10. Schmitt W. Automated Unit Testing of Embedded ARM Applications // Information Quarterly. – 2014. – Vol. 3. – No. – P. 29.

Получено 17.09.2018

Богомолова Мария Анатольевна, к.т.н., доцент, декан Факультета информационных систем и технологий Поволжского государственного университета телекоммуникаций и информатики (ПГУТИ). Тел. (8-846) 339-11-96. E-mail: bogomolova-ma@psuti.ru

Клементьев Сергей Александрович, магистрант Кафедры прикладной информатики ПГУТИ. Тел. (8-846) 228-00-36. E-mail: klementyevsa@yandex.ru

TEST-DRIVEN DEVELOPMENT METHODOLOGY OF ARCHITECTURAL COMPONENTS FOR IOS MOBILE APP «THE MOSCOW SERVICES»

Bogomolova M.A., Klementyev S.A.
Povolzhskiy State University of Telecommunications and Informatics, Samara, Russian Federation
E-mail: bogomolova-ma@psuti.ru

The proposed methodology is focused on the iOS mobile app “State services of Moscow” architecture and can be used for the scenes app development. Using this methodology, the three architectural components (View Controller, Interactor and Presenter) for the scene of citizen’s social card registration were developed step by step: scenes in the form of classes, and also unit-tests for these classes were implemented. The written code shows the development features and can be used as an example for creating new scenes. Using the unit-testing methodology of the iOS mobile app “The Moscow Services” allows to minimize the number of defects in the app, reducing the correction defects cost, and also to contribute for cleaner code writing and user-friendly class interfaces development.

Keywords: *unit-testing, test-driven development, TDD, Clean Swift, mock-object, spy-object*

DOI: 10.18469/ikt.2019.17.1.11

Bogomolova Mariya Anatolevna, Povolzhskiy State University of Telecommunications and Informatics, 23 L. Tolstoy str., Samara, 443010, Russian Federation; Dean of the Faculty of Information Systems and Technologies, PhD in Technical Sciences, Associated Professor. Tel. +78463391196. E-mail: bogomolova-ma@psuti.ru

Клементьев Сергей Александрович, Povolzhskiy State University of Telecommunications and Informatics, 23 L. Tolstoy str., Samara 443010, Russian Federation; master’s student of the Department of Applied Informatics. Tel. +78462280036. E-mail: klementyevsa@yandex.ru

References

1. Orlov S.A., Tsilker B.Y. *Tehnologii razrabotki programmnogo obespechenija* [Software development technologies]. St. Petersburg, Piter Publ., 2012. 608 p.
2. Beck K. *Ehkstremal’noe programmirovanie: razrabotka cherez testirovanie* [Test-driven development by example]. St. Petersburg, Piter Publ., 2003. 224 p.

3. Beck K. *Extreme Programming Explained: Embrace Change. 2nd ed*. Addison Wesley, Boston, 2004. 224 p.
4. Osherove R. *The Art of Unit Testing with Examples in .NET*. Manning, Greenwich, 2009. 324 p.
5. Meszaros G. *xUnit Test Patterns. Refactoring Test Patterns*. Addison-Wesley, Boston, 2007. 948 p.
6. Siman M. *Vnedrenie zavisimostej v .NET* [Deployment of dependencies in .NET]. St. Petersburg, Piter Publ., 2013. 464 p.
7. Fowler M. *Shablony korporativnyh prilozhenij* [Patterns of Enterprise Application Architecture]. Williams, Moscow, 2011. 544 p.
8. Fowler M. Refaktoring. *Uluchshenie sushchestvuyushchego koda* [Refactoring: Improving the Design of Existing Code]. Symbol plus, St. Petersburg, 2010. 432 p.
9. Astels D. *Test Driven Development: A Practical Guide, Upper Saddle River*. Prentice Hall PTR, New Jork, 2003. 592 p.
10. Schmitt W., Automated Unit Testing of Embedded ARM Applications, *Information Quarterly*. 2004, vol. 3, no. 4, p. 29.

Received 17.09.2018

ТЕХНОЛОГИИ РАДИОСВЯЗИ, РАДИОВЕЩАНИЯ И ТЕЛЕВИДЕНИЯ

УДК 621.396

ПРИМЕНЕНИЕ МЕТОДОВ СЛЕПОЙ КОРРЕКЦИИ ИЗОБРАЖЕНИЙ В АППАРАТНО-ПРОГРАММНОМ КОМПЛЕКСЕ «ОРЛАН»

Горячкин О.В.[1], Гусев Н.А.[2]
[1] Поволжский государственный университет телекоммуникаций и информатики, Самара, РФ
[2] Самарский национальный исследовательский университет им. акад. С.П. Королева, Самара, РФ
E-mail: nikolay.gusev@spacekennel.ru

В статье проводится анализ возможности применения работы алгоритма слепого восстановления изображений, основанного на вариационном байесовском подходе в программно-аппаратном комплексе идентификации автомобилей «Орлан». Приводятся количественные характеристики эффективности применения алгоритмов распознавания автомобильных номеров на искаженных снимках. Также приводятся результаты эффективности работы алгоритма при различных степенях искажения исходного изображение. Дается оценка эффективности работы в сравнении с результатами работы алгоритма слепого восстановления открытого программного обеспечения «SmartDeblur». Проведен экспериментальный анализ качества идентификации автомобильных номеров по искаженным изображениям. Проведено сравнение с известными алгоритмами восстановления. Алгоритм на основе байесовского подхода позволил повысить количество верно распознанных номеров при реализации его в аппаратно-программном комплексе «Орлан», другие рассмотренные алгоритмы не позволили повысить качество распознавания на близком к уровню рассматриваемого алгоритма.

Ключевые слова: *вариационный байесовский подход, деконволюция изображений, аппаратно-программный комплекс «ОРЛАН», идентификация номерных знаков*

Введение

Проблемы машинного зрения в настоящее время представляют интерес практически для каждой крупной IT-компании. Яндекс и Uber занимается разработкой автономного такси, Tesla и Google активно работают над автопилотами для автомобилей. Все эти разработки основаны на системах с оптическими или инфракрасными цифровыми камерами. Развитие технических систем обработки информации позволяет решать прикладные задачи, связанные с цифровой обработкой изображений. Под техническими системами измерений и обработки информации в данной статье подразумеваются системы, включающие устройства распознавания объектов на оптических изображениях.

В системах обработки информации изображение обычно регистрируется матрицей сенсоров на основе приборов с зарядовой связью (ПЗС-матрицы). При этом выполняются оцифровка, дискретизация и квантование изображения. Наиболее часто используются полутоновые (серые) изображения с градацией яркости, особенно в аппаратно-программных комплексах машинного зрения (промышленные роботы, автоматические

камеры фиксации нарушений), а также в аэрокосмической съемке, цифровых телескопах, микроскопах и т.д.

Под изображениями будем понимать цифровые полутоновые изображения с градацией серого цвета. В качестве объектов могут выступать фотоснимки человека, текста, движущихся целей (самолет, автомобиль), объекта природы (в том числе сделанные из космоса), теле- и киноизображение и т.д.

Что касается комплексов фиксации нарушений правил дорожного движения, то в настоящее время одной из ключевых задач при их разработке является повышение точности распознавания автомобильных номеров. Так, например, в аппаратно-программном комплексе (АПК) «Орлан», на программно-аппаратной базе которого нами проводились эксперименты, одной из причин возникновения проблем в распознавании является отсутствие автофокуса, при котором объекты (в данном случае автомобили), находящиеся вне области фокуса, подвержены искажению.

Рисунок 1. Внешний вид аппаратно-программного комплекса «ОРЛАН»

На практике эта проблема часто решается путем подбора камеры с улучшенными характеристиками автофокусировки, что обычно влечет за собой увеличение стоимости конечного решения. Другой вариант – использование программных средств улучшения изображений. Одним из эффективных способов компенсации смазов и (или) расфокусировок оптических изображений является так называемое слепое восстановление изображений [1].

В настоящее время создано большое многообразие алгоритмов слепой коррекции изображений (в литературе часто встречается термин «слепая деконволюция») [2], обладающих разными характеристиками и, соответственно, разными возможностями для приложений. В работе анализируются возможность и особенности применения алгоритма слепой коррекции изображений, описанного в [3]. В этой работе рассматривается применение вариационного байесовского подхода и гамма-распределений при оценке неизвестных параметров для предотвращения сходимости к нежелательным оценкам изображения и параметров размытия в процессе работы алгоритма восстановления. В настоящей статье проанализирована эффективность алгоритма детектирования автомобильных номеров АПК «Орлан» в условиях применения алгоритмов слепой коррекции.

Математическая модель изображения

Слепая деконволюция относится к классу задач следующего вида [3-4]:

$$g(x) = h(x) * f(x) + n(x), x = (x_1, x_2) \in I, \quad (1)$$

где $I \subset R^2$ принадлежит изображению; $f(x)$; $g(x)$; $h(x)$; $n(x)$ – представляют собой, соответственно, оригинальное изображение, наблюдаемое изображение, некоторую функцию искажения (функцию распределения точки), представленную размытием и видимым шумом.

Это выражение можно записать в векторной форме

$$g = \boldsymbol{H}f + n, \quad (2)$$

где H – матрица Теплица, полученная из функции искажения. При классическом восстановлении изображения предполагается, что функция размытия известна, и процесс искажения компенсируется с использованием одного из многих существующих алгоритмов восстановления изображений. В рассматриваемом алгоритме функция размытия неизвестна, но ее оценка включена в процедуру восстановления.

Когда оценка функции размытия выполняется совместно с процессом восстановления, большинство алгоритмов решают задачу слепой деконволюции путем включения априорных статистических данных в процесс восстановления. В рамках задач, рассмотренных в [3], применяется байесовская парадигма для совместной оценки изображений, размытия и неизвестных гиперпараметров в задаче слепого восстановления.

Байесовский подход к задаче слепой деконволюции

Целью метода слепой деконволюции является получение оценок h и f на основе наблюдаемого изображения g. Основная задача, которую необходимо решить при разработке алгоритма восстановления, заключается в нахождении совместного распределения $p(\Omega, f, h, g)$, где $\Omega = (a_{im}, a_{bl}, \beta)$ – вектор неизвестных параметров распределений шума, изображения и функции размытия.

Для Ω, f, h и g можно определить условное совместное распределение в виде

$$p(\Omega, f, h|g) = \frac{p(\Omega)p(f|\Omega)p(h|\Omega)p(g|f,h,\Omega)}{p(g)}. \quad (3)$$

Большинство оптических изображений можно описать априорным распределением вида [4]

$$p(f|\Omega) = a_{im}^{\frac{N}{2}} exp\left\{-\frac{1}{2}a_{im}\|\boldsymbol{C}f\|^2\right\}, \quad (4)$$

где C – оператор Лапласа, $N = P \times Q$ – размер вектора столбца, обозначающего лексикографически упорядоченные по строкам пиксели изображения; a_{im}^{-1} – параметр распределения (4). Аналогично можно записать функцию распределения пикселей изображения точечного источника

$$p(h|\Omega) = a_{bl}^{M/2} exp\left\{-\frac{1}{2}a_{bl}\|\boldsymbol{C}h\|^2\right\}. \quad (5)$$

Предположим, что шум является гауссовским с нулевым средним, дисперсией равной β^{-1}. Тогда плотность вероятностей наблюдаемого изображения можно выразить следующим образом:

$$p(g|f, h, \Omega) = \beta^{\frac{N}{2}} exp\left[-\frac{1}{2}\beta\|g - \boldsymbol{H}f\|^2\right]. \quad (6)$$

Оценка параметров Ω производится алгоритмом максимума апостериорной вероятности (МАВ) в виде

$$\widehat{\Omega} = arg\max_{\Omega} p(\Omega|g) = \\ = \int_f \int_h p(\Omega, f, h|g) df dh. \quad (7)$$

Соответственно МАВ оценку истинного изображения и функции искажения можно получить в виде

$$\hat{f}, \hat{h} = arg\max_{f,h} p\left(f, h|\widehat{\Omega}, g\right) = \\ = arg\,max_{f,h}\, p\left(f\middle|\widehat{\Omega}\right)p\left(h\middle|\widehat{\Omega}\right)p\left(g\middle|f, h, \widehat{\Omega}\right). \quad (8)$$

В рамках вариационного байесовского подхода оценку неизвестных параметров проводят следующим образом (см. подробнее [3; 5]). Обозначим через Θ все неизвестные нам параметры:

$$\Theta = (\Omega, f, h) = (a_{im}, a_{bl}, \beta, f, h). \quad (9)$$

Для $\Theta \in \{a_{im}, a_{bl}, \beta, f, h\}$ обозначим как Θ_θ подмножество параметров Θ за исключением параметра θ. Например, если $\theta = f$, то $\Theta_f = (a_{im}, a_{bl}, \beta, h)$. Тогда расстояние Кульбака-Лейблера между аппроксимирующим распределением с независимыми компонентами и распределением параметров, полученных на основе наблюдаемого изображения g, будет иметь следующий вид

$$D_{KL}(q(\Theta)||p(\Theta|g)) = \\ = D_{KL}(q(\theta)q(\Theta_\theta)\big||p(\Theta|g)\big). \quad (10)$$

Оценка неизвестного параметра будет иметь распределение, при котором расхождение минимально:

$$\hat{q}(\theta) = arg\max_{q(\theta)} \mathrm{C}_{KL}(q(\theta)q(\Theta_\theta)\big||p(\Theta|g)\big). \quad (11)$$

Таким образом в рамках данного подхода задача слепой деконволюции изображения сводится к итерационному поиску $q(\theta)$,если θ = f и нахождению оценки в виде

$$\hat{f} = \int q(f) df. \quad (12)$$

Распределение гиперпараметров распределений может быть выбрано различным способом, в рассматриваемом алгоритме $p(\Omega)$ – гамма распределение.

Особенности анализа изображений в АПК «ОРЛАН»

Анализ изображения в комплексе фиксации нарушений правил дорожного движения производится в три этапа: детектирование прецедента, детектирование и локализация регистрационного номера транспортного средства, распознавание номерного знака. Пример изображения, полученного при помощи АПК «ОРЛАН», представлен на рисунке 2.

Первоначальной целью исследования являлось повышение точности распознавания среди детектируемых номерных знаков. Однако, как выяснилось, уязвимым местом в работе АПК «ОРЛАН» является именно детектирование номера, а не точность его распознавания.

В основе алгоритма детектирования автомобильного номера лежит довольно распространённый метод анализа изображения с применением оператора Собеля [4]. Оператор Собеля – это дискретный дифференциальный оператор, вычисляющий приближенное значение градиента яркости изображения. Очевидно, что при искажении типа «размытие» или «смаз» картина градиента яркостей становится слабовыраженной, что затрудняет решение задач машинного зрения. При расфокусированном изображении программная часть АПК «ОРЛАН» не сможет обнаружить номерные знаки и вынесет решение, что автомобиля на фотографии нет,

так как отличительный признак транспортного средства отсутствует.

Рисунок 2. Изображение, полученное при помощи АПК «ОРЛАН»

Если локализация номерного знака прошла успешно, то изображение с большой вероятностью обладает достаточным качеством и распознавание будет выполнено с необходимой высокой точностью. Другими словами, в таком случае распознавание успешно осуществляется без использования слепой деконволюции.

Тестирование алгоритма слепой деконволюции

При подготовке материала для тестирования алгоритма была использована фотография, изображенная на рисунке 3. Для получения изображения использовался фотоаппарат модели Cannon EOS 1300D.

Рисунок 3. Исходное изображение

Чтобы оценить эффективность работы алгоритма в зависимости от степени искажения, при помощи пакета программ Matlab была получена серия снимков с заведомо известной функцией искажения. При решении задач обработки изображений применяется функция фильтрации imfilter в паре с функцией fspecial, которая позволяет задавать различные типы масок фильтра. В данном случае в качестве функции искажения использовался фильтр Гаусса размером 50×50 пикселей с переменным стандартным отклонением: *fspecial* («*gaussian*», 50, *x*), где *x* – стандартное отклонение. Стандартное отклонение x изменялось в диапазоне от 0,1 до 3,0 с шагом 0,1. В результате получается 30 изображений с разной степенью размытия по Гауссу. Изображения со степенью размытия 0,4; 1,2; 2,0 и 2,8 представлены на рисунке 4.

Рисунок 4. Синтетически деградированные изображения с разной степенью размытия

При восстановлении деградированного изображения необходимо произвести операцию обратной свертки и при этом учитывать уровень шума.

Рисунок 5. Изображения со стандартными отклонениями 1,0; 1,5 и 2,0

Рассматривая модель изображения (1), можно прийти к выводу, что для нахождения оригинального изображения $f(x)$ необходимо поделить левую и правую часть выражения (1) на $h(x)$. Однако в таком случае достаточно шума даже с низким уровнем (который всегда присутствует на изобра-