УДК 004:054

ГЛОБАЛЬНОЕ ПОЭТАПНОЕ ТЕСТИРОВАНИЕ ДЛЯ ANDROID ПРИЛОЖЕНИЙ

Чоповенко А. О., Артемов А. О.

Национальный технический университет Украины "Киевский политехнический институт", Украина, Киев

В данной работе рассмотрены проблемы тестировния и его особенности для приложений разработаних под операционную систему Android с которыми сталкиваются большенство разработчиков при разработке своих програмных продуктов для операционной системы Android. Проведен осмотр существующих проблем тестирования и их анализ. Рассмотрено все аспекты тестирования приложений как тестирование трех основных составляющих приложения таких как бизнес логика приложения, интеграция API на стороне сервера и пользовательский интерфейс приложения. Предложенны некоторые подходы для написания тестов и проверки приложений под операционную систему Android.

Ключевые слова: Android разработка, программное тестирование, пользовательский интерфейс, бизнес логика приложений, API интеграция.

Чоповенко А. О., Артемов А. О. Глобальне поетапне тестування для Android додатків/ Національний технічний університет України "Київський політехнічний інститут", Україна, Київ

У даній роботі розглянуті проблеми тестуванняя і його особливості для додатків розроблених під операційну систему Android з якими стикаються більшість розробників при розробці своїх програмних продуктів для операційної системи Android. Проведено огляд існуючих проблем тестування і їх аналіз. Розглянуто всі аспекти тестування додатків як тестування трьох основних складових програми таких як бізнес логіка програми, інтеграція API на стороні сервера і призначений для

користувача інтерфейс програми. Запропоновані деякі підходи для написання тестів і перевірки додатків під операційну систему Android.

Ключові слова: Android розробка, програмне тестування, користувацький інтерфейс, бізнес логіка додатків, API інтеграція.

Chopovenko A. O., Artemov A. O. Global phased testing for Android applications/ National Technical University of Ukraine "Kyiv Polytechnic Institute", Ukraine, Kiev

This paper deals with the testing problems and its features for applications developed for the Android operating system encountered by most of the developers in the development of its software products for the Android operating system. Inspect existing testing problems and their analysis. It considers such aspects of application testing as testing of the three main components of the application such as application business logic, integration with a server side API and the application's user interface. It offer some of the approaches to writing tests for the applications on Android operating system.

Key words: Android development, software testing, user interface, application business logic, the API integration.

Введение. Не секрет что автоматизация тестирования ключ к успеху разработки программного обеспечения. Но начиная писать тесты для Android приложений, многие сталкиваются с такими проблемами как незнание что нужно тестировать? Как это нужно тестировать? Неготовность кода для тестирования, высокий уровень охвата кода с тестами которые проходят даже если приложение не работает или тесты которые проваливаются в случайном порядке без видимых на то причин, но проходящие при повторной сборке проекта.

Цель данной статьи: Решения подобных проблем путем налаживания поэтапного тестирование. В процессе этого поэтапного тестирования

определяется, что мы должны тестировать и как. Какие инструменты использовать и почему? Какова область наших испытаний?

И первым шагом будет определение основных аспектов, которые нужно тестировать:

-Тест бизнес логики независимо от Фреймворков и библиотек

-Тест интеграции API на стороне сервера

-Тест пользовательского интерфейса приложения

Но даже с налаженным поэтапным тестированием написание тестов может быть очень сложным, если тестируемый код не готов, поэтому код должен быть проверяемым, читаемым, должно иметь четкие и структурированные цепочки ответственностей, зависимости должны быть минимальными, а связность высокая.

Тестируя **бизнес логику**, крайне важно, проверить, действительно ли бизнес-логика реализует предопределенные требования к продукту. Для этого необходимо изолировать код, который будет проверятся, и смоделировать различные начальные сценарии для настройки поведения некоторых компонентов во время выполнения. Далее будет тестироваться код, выбирая детали которые мы хотим осуществить. После завершения, нам нужно проверить состояние программного обеспечения после тестирования исследуемого объекта.

Ключ к этому подходу тестирования является Принцип инверсии зависимостей. При написании кода, который зависит от абстракций, можно разделить программное обеспечение на разные слоя. Для того чтобы получить экземпляр зависимости, нужно запросить его от кого-то. В качестве альтернативы, мы можем получить его после того, как экземпляр создается. Есть части нашего программного обеспечения, где необходимо создать код, чтобы получить данные взаимодействующих компонентов. Для этого создаются тестовые двойники для имитации исходных сценариев или различного программного поведения, которые помогают в проектировании

тестов. С помощью тестовых двойников, становится возможным моделировать как поведение так и состояние производственного кода с заменой тестовыми двойниками. Также, это поможет выбрать объем для теста, который представляет собой количество кода для тестирования. Без инверсии зависимостей, все наши классы будут получать их зависимости независимо друг от друга. В результате, реализация класса будет связана с реализацией зависимостей и, следовательно, мы не можем ввести тестовых двойников, чтобы сократить поток выполнения производственного кода.

После того, как мы можем проверить, должным ли образом реализованы требования к нашей продукции, мы должны продолжать работать над нашим поэтапным тестированием. Следующие что, будет проверятся это **интеграция с внешними компонентами** замененными тест-двойниками на предыдущем этапе. На этом этапе основной целью тестов будет проверка обмена сообщений с API, парсинг ответов от API, реализация механизмов аутентификации и обработка ошибок API должным образом. Чтобы проверить эти аспекты нужно имитировать разные ответы со стороны сервера и манипулировать запросами со стороны клиента. Для проверки API клиента необходимо использовать тест-двойник и манипуляцию запросами по информации, переданной через сеть. В этом случае будет использоваться тест-двойник, известный как мок (mock) и сторонний инструмент, известный как MockWebServer.

MockWebServer является веб-сервер с программируемыми сценариями для тестирования HTTP-клиентов, реализованных Square и написанных на Java. Основной подход для достижения нашей цели является постановка в очередь некоторых предварительно настроенных ответов HTTP, выполнение минипуляции над отправленными запросами HTTP, и проверка состояния исследуемого объекта в конце выполнения теста. MockWebServer запустит встроенный сервер HTTP, где можно будет настроить наши ответы. Указав наш клиент API на хосте, MockWebServer и настройке некоторых HTTP-

ответов, мы сможем протестировать наш клиент API. Эта библиотека будет работать, как мок, но вместо того, чтобы заменить производственный код внутри клиента API, он заменит производственный код, реализованный на стороне сервера.

Таким образом, используя моки можно с легкостью воспроизвести различные сценарии и модели поведения API, чтобы проверить, реагирует ли реализация клиента API должным образом. В то же время, мы получаем «живую» документацию сервера API на основе всех написанных тестов и ответов HTTP. Так же с помощью подобных тестов можно воспроизвести те случаи, когда информация, полученная из ответов HTTP является неполной или сетевые условия являются неблагоприятными.

Большинство мобильных приложений основаны на мощных компонентах **пользовательского интерфейса**. Эти элементы или компоненты пользовательского интерфейса используются для отображения информации пользователю. Без этого компонента приложение является не более чем приложением командной строки. Если мы рассмотрим большинство приложений, то можно прийти к выводу, что почти все они являются лишь front-end частью для отображения информации, полученной от службы. Большая часть кода, написанного для разработки этих приложений является UI-кодом, который написан на вершине Android SDK. Именно поэтому очень важно эффективно протестировать UI код.

Работая с пользовательским интерфейсом приложения, необходимо удостовериться, что он показывает правильную информацию пользователю после загрузки пользовательского интерфейса; показывает правильные сообщения пользователю и отображает правильные экраны при взаимодействии с пользователем. Для того что бы проверить все три требования нам необходимо работающую среду для запуска тестов, такую как Android эмулятор или устройство, использующие Android в качестве ОС. После того, как среда была инициализирована, необходимо выполнять

различные действия в пользовательском интерфейсе приложения, а также выполнять манипуляции информацией, отображаемой пользователю. Для того, чтобы проверить, работает ли пользовательский интерфейс, как и ожидалось, нужно провести манипуляции над компонентами пользовательского интерфейса и выполнить моделирование взаимодействий с его компонентами.

Для того чтобы полностью контролировать тестовый сценарий, нужно снова использовать тесты-двойники с помощью которых мы будем испытывать исследуемый объект. Соответственно, уменьшение сценария тестирования и улучшения детерминизма наших испытаний уменьшит их погрешность. Используя тесты-двойники можно будет контролировать все данные к которым приложение может получить доступ, от пользователя, вошедшего в систему к информации, отображаемой пользователю. Так же стоит отметить что данные с тестов-моков не будут использовать подключение к Интернету.

Инструменты, которые могут помочь нам, являются Dagger 2 в качестве основного компонента для замены производственного кода с тест-двойников и Espresso, чтобы взаимодействовать с устройством и выполнять манипуляции. Чтобы использовать Dagger 2 для целей тестирования можно использовать правило JUnit названое DaggerMock правилом.

Так же одним из ключевых моментов в написании этих тестов является то, что при использовании инжектора зависимостей, мы можем настроить тестовых двойников, необходимых для воссоздания первоначального сценария на испытание исследуемого объекта.

Эти тесты имеют большой объем и могут быть рассмотрены как глобальные которые охватывают все приложение. Мы испытываем одновременно слой представления логики и весь код, необходимый для взаимодействия с Android SDK и отображения информации пользователю. Другой обоснованный подход может быть основан на интенсивном

использовании Принципа инверсии зависимостей. Следуя шаблону Model View Presenter и написанию тестируемого кода, можно было бы заменить реализацию View моком и убедится, что информация, направляемая в пользовательский интерфейс является правильной. Но при этом нужно помнить, что делая это, необходимо использовать шаблон Model View Presenter или Model View View Model и переместить всю нашу логику представления для классов вне фреймворков как Presenter. Используя этот подход, можно протестировать наш код с помощью виртуальной машины Java вместо Android эмулятора.

Выводы. Таким образом используя подходы, описанные выше можно легко создавать воспроизводимые тестовые сценарии, где бизнес-логика и пользовательский интерфейс будет тестироваться в изолированной среде, а также как протестировать интеграцию API с помощью мок-вызовов HTTP.

Литература:

1. MockWebServer documentation. [Электронный-ресурс] / Github [Офиц. сайт]. URL: https://github.com/square/okhttp/tree/master/mockwebserver

2. Dagger2 documentation [Электронный-ресурс] / Github [Офиц. сайт]. URL: https://github.com/google/dagger.

3. Android - Architecture / [Электронный-ресурс] / Tutorialspoint [Офиц. сайт]. URL: http://www.tutorialspoint.com/android/android_architecture.htm.

4. DaggerMock documentation. [Электронный-ресурс] / Github [Офиц. сайт]. URL: https://github.com/fabioCollini/DaggerMock.

5. Espresso documentation. [Электронный-ресурс] / Github [Офиц. сайт]. URL: https://google.github.io/android-testing-support-library/docs/espresso

6. Test Double by Martin Fowler. [Электронный-ресурс] / MartinFowler.com [Офиц. сайт]. URL: http://www.martinfowler.com/bliki/TestDouble.html.

7. WireMock documentation [Электронный-ресурс] / WireMock [Офиц. сайт]. URL: http://wiremock.org.

8. П. Дайтел. Android для программистов. Создаем приложения. Питер.2013.

9. Inversion of Control Containers and the Dependency Injection pattern by Martin Fowler [Электронный-ресурс] / MartinFowler.com [Офиц. сайт]. URL: http://martinfowler.com/articles/injection.html.