Создание унифицированных механизмов автоматизированного тестирования приложений для мобильных устройств

И.А. Барсуков, Н.А. Наумова, Д.К. Бострикова, Е.А. Машина

Национальный исследовательский университет ИТМО, Санкт-Петербург

Аннотация: Статья посвящена рассмотрению вопросов автоматизации процессов создания программного обеспечения, в частности, вопросу создания универсальных средств тестирования приложений мобильных устройств.

В работе подробно рассматривается процесс создания инструмента для автоматизированного тестирования, позволяющего осуществлять проверку приложений, созданных для Android и iOS, а также приводятся результаты проведенных исследований, позволяющие определить сокращение временных затрат на процедуры тестирования мобильных приложений, возникающих при использовании разработанного инструмента для всех квалификационных категорий специалистов, задействованных в тестировании приложений.

Предложенный авторами модульный способ построения решения позволит в дальнейшем достаточно просто расширить функциональность решения путем добавления новых алгоритмов и режимов тестирования, существенно увеличивая «зону покрытия» автоматизированными тестовыми сценариями.

Ключевые слова: автоматизация тестирования, тестирование пользовательских интерфейсов, генерация кода, мобильное тестирование, паттерн PageObject, тестирование программного обеспечения, quality assurance, обеспечение качества, управление качеством, функциональное тестирование.

Введение

Одной из современных тенденций развития технических и организационных изменений в области разработки программного обеспечения является существенное увеличение уровня автоматизации всех процессов данной области деятельности.

Происходящий процесс автоматизации удобен не только тем, что позволяет существенно упростить выполнение большого ряда рутинных операций создания программного продукта, повысив тем самым производительность разработчиков, но и тем, что позволяет ускорить процесс выпуска новых версий приложения, отвечающего представленным к нему требованиям качества.

При этом, с точки зрения пользователя сама процедура разработки

программного продукта все более превращается в использование «черного ящика», скрывающего от него как отдельные детали, так и общую структуру процесса создания конечного решения, все более приближая его к сервисной модели создания программного обеспечения (Software as a Service, SaaS).

Это, с одной стороны, существенно снижает специализированные квалификационные требования к разработчикам таких продуктов и позволяет перенести процесс разработки ПО непосредственно в организацию-заказчика, с другой стороны, требует существенной модернизации средств подобной разработки, в которых особенно возрастает роль автоматизированных компонентов, выполняющих операции тестирования различного уровня, представляющие собой систематизированные исследования создаваемого ПО с целью получения исчерпывающей информации о работоспособности и качестве продукта.

На сегодняшний день существенная часть программных решений представляет собой веб-ориентированные приложения, использующиеся на мобильных устройствах. Применительно к созданию таких решений под процессами автоматизации тестирования обычно понимают создание сценариев, эмулирующих повторяющиеся действия инженера по контролю качества и создающие необходимые условия для тестирования приложения. При этом, при существенном росте разнообразия создаваемых веб-ориентированных приложений и внешнего программно-аппаратного окружения особый интерес начинают представлять унифицированные решения.

Поэтому, с ростом популярности мобильных устройств и приложений, тестирование становится еще более важным элементом процесса разработки [1]. Однако, ручное тестирование при этом оказывается очень трудоемким и затратным процессом, особенно при расширении функциональности тестируемых приложений [2].

При этом следует иметь в виду, что и переход к полной автоматизации тестирования, по большей части, представляет собой достаточно затратный процесс, что связано, в первую очередь, со сложностями обеспечения взаимодействия элементов создаваемых решений. Именно поэтому элементы инструментария автоматизированного тестирования создаваемых приложений перед передачей их в промышленную эксплуатацию должны проходить целый ряд специализированных проверок эффективности.

В связи с этим, целью настоящей работы является не только демонстрация возможностей создания автоматизированных систем тестирования приложений, но и проведение фактических исследований повышения производительности процесса разработки при использовании подобных средств автоматизации.

Представленное в настоящей работе решение представляет собой инновационный подход к созданию унифицированных средств автоматизации тестирования программного обеспечения, который включает в себя генерацию тестовых сценариев, создание объектов страниц для них и запуск разработанных тестовых сценариев в облачной среде.

Описание исследования

Целью проводимого исследования является определение влияния процессов внедрения автоматизации тестирования пользовательского интерфейса мобильных приложений на время выхода новой функциональности приложения.

Для достижения поставленной цели были решены следующие задачи:

- продемонстрирована практическая осуществимость концепции использования тестовых сценариев и объектов страниц,
- проанализированы объекты страниц и определен общий шаблон,
- разработан генератор объектов страниц (при этом созданный генератор

страниц был реализован и интегрирован в жизненный цикл разработки, позволяя упростить и ускорить процесс тестирования),

- проведены измерения времени, затрачиваемого на тестирование,
- произведен анализ полученных результатов, что позволило определить эффективность разработанного инструмента.

В данной работе рассмотрен процесс тестирования (в ручном и автоматизированном режиме) двух мобильных приложений, которые имеют одинаковую логику, сделаны по одной спецификации и общему дизайну, но различны в силу того, что реализованы нативно на различных языках: под Android и iOS.

При использовании механизмов ручного тестирования инженеры по контролю качества вручную проверяют составленные тест-кейсы сначала на одном приложении, а затем – на втором; кроме того, важной частью тестирования перед выпуском является проверка совместимости приложения с различными версиями операционных систем и устройств. При этом, инженер по контролю качества должен убедиться, что приложение работает корректно на разных версиях операционных систем и на разных моделях мобильных устройств.

С увеличением сложности тестируемых приложений и ограниченности времени на их разработку ручное тестирование становится крайне трудоемким и затратным процессом, поскольку необходимо вручную взаимодействовать со множеством элементов пользовательского интерфейса приложения [3].

В связи с этим, появляется необходимость автоматизировать тестирование в тех случаях, когда это возможно. При этом, необходимо отметить, что фреймворки для автоматизированного тестирования, в большинстве случаев, имеют плохую поддержку взаимодействия с биометрическими данными, поэтому предполагается, что тестовые сценарии,

проверяющие эту функциональность, будут реализованы инженерами по контролю качества вручную [4].

Диаграмма последовательности проведения тестирования двух рассматриваемых приложений представлена на рисунке 1.

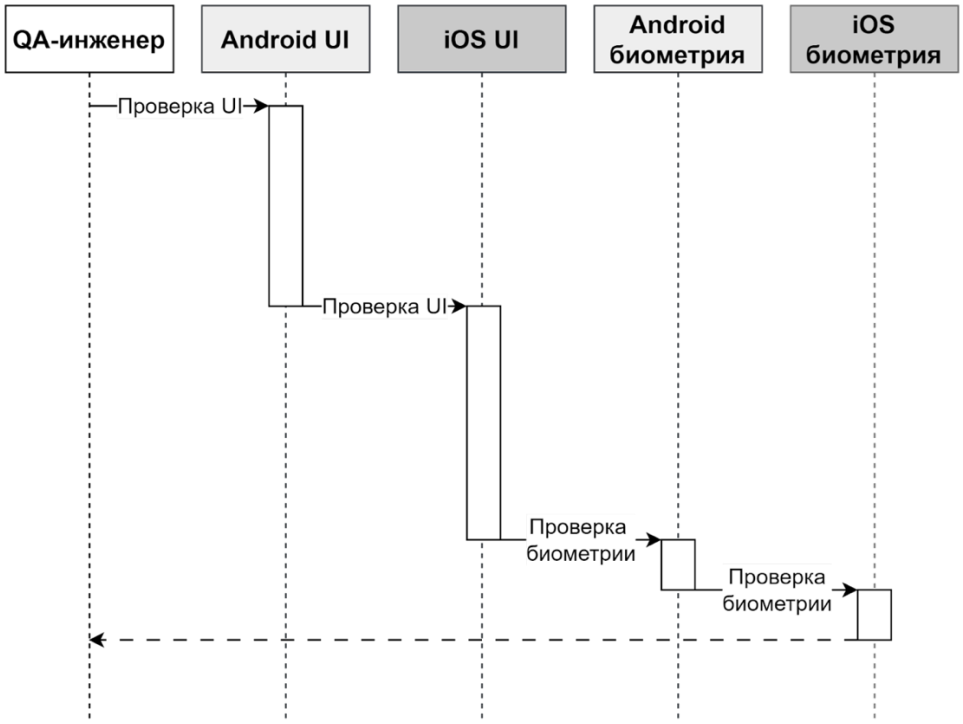

Рис. 1. – Диаграмма последовательности проверки приложений без использования автоматизированного тестирования

На сегодняшний день существует множество инструментов автоматизации тестирования пользовательского интерфейса приложений [5, 6]. Поэтому, при выборе инструмента автоматизации необходимо учитывать не только его технические возможности (в частности, предоставление возможности проведения кроссплатформенного тестирования), но и соответствие языку программирования, с которым знакомы инженеры по контролю качества [7-9]. Это позволит избежать дополнительных затрат на обучение персонала и повысить эффективность автоматизированного тестирования.

Создание универсального средства автоматизации тестирования приложений на Android и iOS

Для реализации инструмента автоматизации тестирования мобильных приложений на платформах Android и iOS был выбран фреймворк Appium [10]. Тестовый модуль реализован на языке программирования Kotlin, который в настоящее время активно используется специалистами по контролю качества.

Тестирование пользовательского интерфейса при проведении исследований было организовано путем создания общего тестового сценария под обе платформы (Android и iOS), но с использованием различных объектов-реализаций паттерна PageObject, специфичных для платформы тестируемого приложения [11].

Поскольку взаимодействие с элементами пользовательского интерфейса целевых мобильных приложений задается конечным списком действий, таких как клик по элементу, ввод текстовых и числовых значений в текстовое поле, которые в реализации объектов страниц, с одной стороны, представляют собой большое количество однотипного кода, а, с другой стороны, отличаются в зависимости от платформы, создание объектов модели взаимодействия с пользовательским интерфейсом реализовано с использованием инструмента генерации кода из определенного описания. При этом, для решения данной задачи необходимо обеспечить не только реализацию тестовых сценариев, но и включение данного инструмента в цикл разработки мобильных приложений путем интеграции с сервисами непрерывной интеграции – облачной фермой мобильных устройств и удаленным центром приложений, предоставляющим сборки приложений для тестирования [12].

Архитектура разработанного решения представлена на рисунке 2. При проектировании архитектуры учитывалась необходимость обеспечить

возможность локального запуска тестовых сценариев для первичной проверки их корректности. В связи с этим, для реализации поставленной задачи была выбрана модульная архитектура.

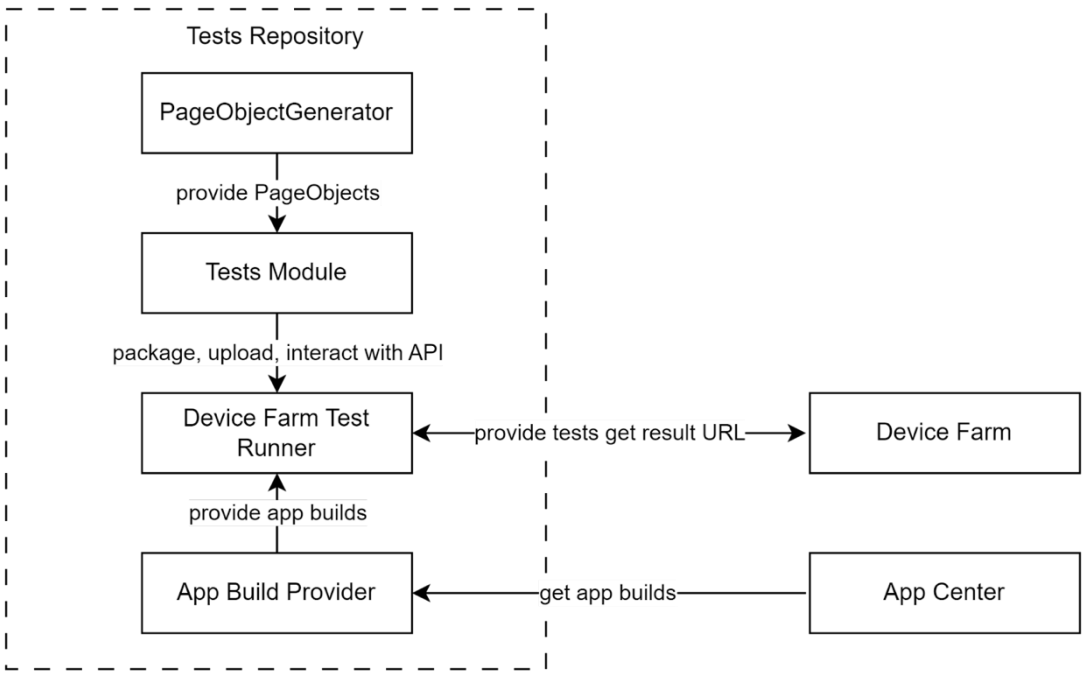

Рис. 2. – Архитектура решения

Рассмотрим далее особенности каждого модуля.

Модуль с тестовыми сценариями и объектами страниц

Описываемый модуль является набором тестовых сценариев и объектов страниц на языке программирования Kotlin. В основе тестовых сценариев лежит последовательность действий по созданию объектов страниц в зависимости от выбранной платформы, вызову методов объектов страниц и проверке корректности получаемых в процессе работы тестового сценария данных. Реализация объектов страниц выполнена в виде двух отдельных сущностей: интерфейса, описывающего методы и поля, обеспечивающие взаимодействие с объектом страницы, и двух классов-реализаций, специфичных для конкретных платформ. Классы содержат реализацию членов, описанных в интерфейсе, с учетом специфики конкретной платформы. Для наглядной иллюстрации различий в описании

функции взаимодействия с кнопкой диалогового окна на различных операционных системах, представлены данные в таблице 1, общие фрагменты выделены полужирным.

Таблица № 1

Описания взаимодействий с компонентами пользовательского интерфейса

iOS	Android

override fun clickButton() {
Actions(driver).apply {
moveToElement(
driver.findElement(
By.id(someBtnId)
)
)
.click()
perform()
}
}
	override fun clickButton() {
	
	driver.findElement(
	By.id(someBtnId)
	)
	
	.click()
	
	}

Пример класса-реализации диалогового окна представлен в листинге 1. Все классы объектов страниц имеют одинаковую структуру и состоят из следующих элементов:

- объект ожидания `WebDriverWait`, который необходим для ожидания появления элемента на странице;
- поле `LAYOUT_ID` – уникальный идентификатор для определения того, правильная ли страница открыта;
- идентификаторы элементов страницы, с которыми происходит взаимодействие внутри методов;
- блок инициализации, внутри которого происходит ожидание и проверка того, что дальнейшее взаимодействие происходит с правильным объектом страницы;
- реализации методов взаимодействия с элементами страниц, состоящие из последовательности ограниченного списка действий:
    - `sendKeys` – запись значения в поле ввода;

- `clear` – очистка текстового поля от введенных ранее символов;
- `click` – нажатие на элемент;
- `wait.until(ExpectedConditions.visibilityOfElementLocated(...))` – остановка выполнения и ожидание, пока элемент не появится на экране и станет готов к взаимодействию.

Листинг 1. Пример объекта диалогового окна в Android

```
class AndroidSetupDialog (
       private val driver: AppiumDriver<MobileElement>
): SetupPasscodeDialogPageObject {

    val wait = WebDriverWait(driver, 60)
    override val LAYOUT_ID = androidId("confirm_dialog")

    private val okBtnId = androidId("ok_btn")
    private val skipBtnId = androidId("skip_btn")

    init {
       wait.until(presenceOfElementLocated(
          By.id(LAYOUT_ID)
       )
    }

    override fun setup() {
       driver.findElement(By.id(okBtnId)).click()
    }

    override fun cancel() {
       driver.findElement(By.id(skipBtnId)).click()
    }
}
```

Модуль генерации объектов страниц

Поскольку интерфейсы и классы объектов страниц имеют одинаковую структуру, модуль их генерации реализован на основе Yet Another Markup Language (YAML) описания. Структура описания представлена в таблице 2.

После чтения файла полученные данные разбиваются на три модели: модель верхнего уровня, модель интерфейса и модель реализации. Модель верхнего уровня содержит данные обо всей спецификации, модель

интерфейса содержит список полей и методов для описания интерфейса, а модель реализации содержит конечный тип страницы, словарь названий и значений полей и словарь названий и списка действий функций.

Таблица № 2

Составные части файла описания объектов страницы

YAML-описание	Пояснение
`name: SetupPasscodeDialog`	Название для интерфейса и классов реализаций. Для реализаций добавлен соответствующий префикс
`type: dialog`	Тип страницы: `fragment` или `dialog`
`basePackage: myapp.pageobjects.authorized`	Название пакета для создания иерархии
`common:`	Блок описания интерфейса
`fields:` `  - LAYOUT_ID`	Названия общедоступных полей
`functions:` `  - setup`	Названия общедоступных методов
`android:`	Блок описания реализации: `android` или `ios`

`fields:`
`  LAYOUT_ID: confirm_dialog`
`  okBtnId: ok_btn`
	Названия и значения идентификаторов полей

`functions:`
`  setup:`
`  - click okBtnId`
	Реализации методов, состоящие из последовательности действий

Далее происходит генерация интерфейсов и реализаций. Генерация интерфейсов использует модель верхнего уровня для организации иерархии объектов страниц и определения названий для файла и интерфейса. Затем используется основная модель интерфейса для генерации интерфейса.

Генерация классов также использует модель верхнего уровня для организации иерархии объектов страниц и определения названий для файла и класса. Модель реализации используется для генерации содержимого класса. Словарь полей преобразуется в поля класса, а на основе конечного типа и описания функций генерируется реализация функций.

Модуль получения сборок приложений

Данный модуль выполняет ряд действий для реализации процесса получения приложения путем взаимодействия с API в удаленном центре приложений. Сначала выполняется запрос списка доступных версий, затем производится выбор необходимой версии приложения. Затем, используя второй API-запрос, модуль загружает выбранную версию приложения в формате ipa (iOS package App Store) для iOS приложения и aab (Android App Bundle) для Android приложения [13]. Для запуска на устройстве приложения для Android необходимо преобразовать его в apk-файл, поэтому модуль преобразует архив приложения в формат apk (Android package). После завершения процесса преобразования, модуль возвращает полученный .apk или .ipa файл, который может быть использован для тестирования приложения.

Модуль интеграции с фермой мобильных устройств

Ферма мобильных устройств – это сервис для тестирования мобильных и веб-приложений, позволяющий производить проверки системы в различных браузерах и на разных устройствах. Данный сервис позволяет запускать тестовые сценарии параллельно на нескольких устройствах для ускорения процесса и получать автоматически сформированные отчеты о выполнении.

Данный модуль отвечает за интеграцию с фермой мобильных устройств. Кроме того, он позволяет выбирать, где тесты будут запускаться — локально или удаленно. В случае, когда выбран локальный запуск, модуль

установит необходимые конфигурационные параметры и произведет запуск тестовых сценариев локально. Если же выбран удаленный запуск, то будет произведена отправка на сервис фермы мобильных устройств полученного приложения и архива с тестовыми сценариями, а также выполнится выбор пула мобильных устройств, на которых будет произведен запуск тестовых сценариев. В результате работы данного модуля будет получена ссылка на просмотр результатов выполнения тестовых сценариев.

Результаты проведенных исследований и направления дальнейшего развития работ

Измерение среднего времени, затрачиваемого на проверку приложений, проводилось при участии четырех инженеров по контролю качества разной квалификации, которые проводили проверки одного приложения одной мажорной версии, но разных минорных версий (более и менее стабильных). На основании усредненных показателей затраченного ими на проверку приложений времени было измерено, сколько времени у специалиста занимает выполнение тестовых сценариев, анализ результатов и заведение дефектов [14].

На рисунке 3 продемонстрирована диаграмма охвата тестирования и процент автоматизированных тестов относительно всего объема тестовых сценариев. Как можно заметить, созданным решением уже покрыто 38% стандартных тестовых задач. На следующих этапах планируется провести работы по автоматизации еще 53% тестовых задач. При этом следует иметь в виду, что на настоящий момент не все сценарии могут быть покрыты автоматизированными тестами. Это связано с тем, что некоторые сценарии включают использование биометрических данных, которые нельзя в полном объеме эмулировать в тестовой среде. В связи с чем, около 9% задач тестирования на настоящее время планируется проводить в ручном режиме.

Результаты измерения временных затрат инженеров в случае проверки

приложений без использования автоматизированных тестовых сценариев представлены в таблице 3.

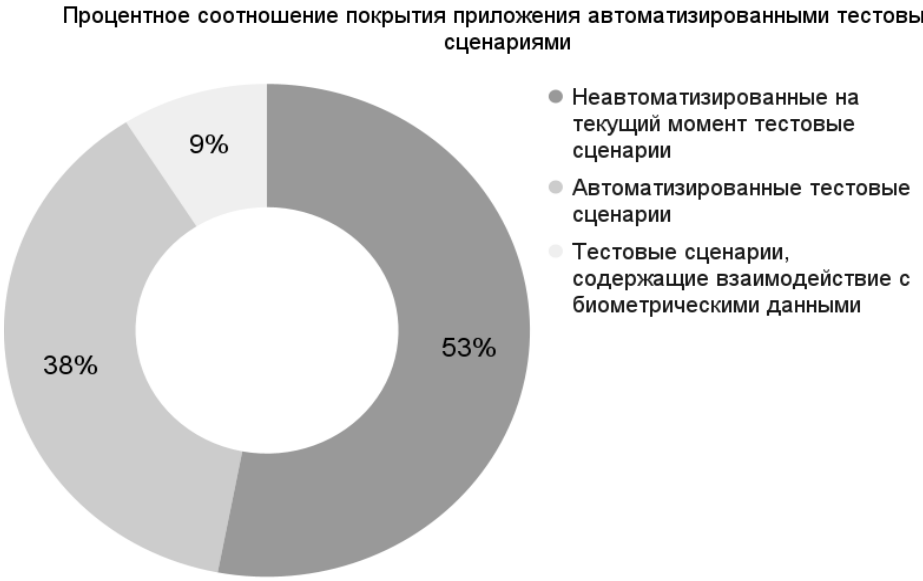

Рис. 3. – Процент покрытия автоматизированными тестовыми сценариями

Из приведенных результатов производственного хронометража можно сделать вывод, что специалист по контролю качества может проверить приложение для iOS в среднем за 8.3ч., а для Android – в среднем за 7.5ч. без применения средств автоматизации, а с применением средств автоматизации инженер по контролю качества может проверить приложение для iOS в среднем за 4.5ч., а для Android – в среднем за 5.1ч., что свидетельствует о том, что время, затраченное на проверку приложений, сократилось.

Изменение рабочего процесса представлено на рисунке 4. Как можно заметить, некоторые части тестирования теперь происходят без участия инженера по контролю качества, а время тестирования сократилось. За счет использования средств автоматизации процесса тестирования становится возможным запускать реализованные автоматизированные сценарии параллельно с работой инженера по контролю качества, который может заниматься проверкой неавтоматизированных сценариев, а также сценариев,

связанных с проверкой биометрических данных, причём независимо от выполнения автоматических тестов. После завершения автоматических тестов инженеру по контролю качества необходимо проанализировать результаты выполнения и завести отчеты о дефектах при необходимости.

Таблица № 3

Сводная таблица среднего времени проведения проверок приложений

	Затраченное время на ручное тестирование, ч.				Затраченное время на ручное и автоматизированное тестирование, ч.			
	Android		iOS		Android		iOS	
	190.0 (unstable)	190.3 (stable)	162.0 (unstable)	162.4 (stable)	190.0 (unstable)	190.3 (stable)	162.0 (unstable)	162.4 (stable)
Junior QA	9.4	8.8	10.2	9.7	6.7	6.6	7.4	7.1
Middle QA	8.0	7.7	8.1	7.7	5.1	4.4	5.3	5.1
Middle QA	7.6	7.3	8.8	8.6	4.5	4.3	5.9	5.5
Senior QA	5.9	5.2	6.6	6.3	3.6	3.2	4.2	3.7
Автоматизированные тесты	-	-	-	-	3.2	3.1	3.3	3.3
Среднее время работы QA	7.5		8.3		4.5		5.1	
Среднее время на автоматизированные тесты	-				3.2			
Суммарное время, затраченное QA	15.7				9.6			

В компоненте генерации объектов страниц мобильных приложений планируется расширение функциональности и повышение эффективности решения при помощи добавления поддержки новых технологий и инструментов. Одним из таких инструментов является Jetpack Compose – новый инструментарий для создания пользовательских интерфейсов в Android. Планируется добавление поддержки Jetpack Compose и XPath – языка запросов для поиска элементов в дереве XML. Это даст возможность более точно настраивать поиск элементов и создавать более гибкие тестовые сценарии. Предполагается также добавление возможности генерации

исходного кода тестовых сценариев, что позволит ускорить процесс автоматизации.

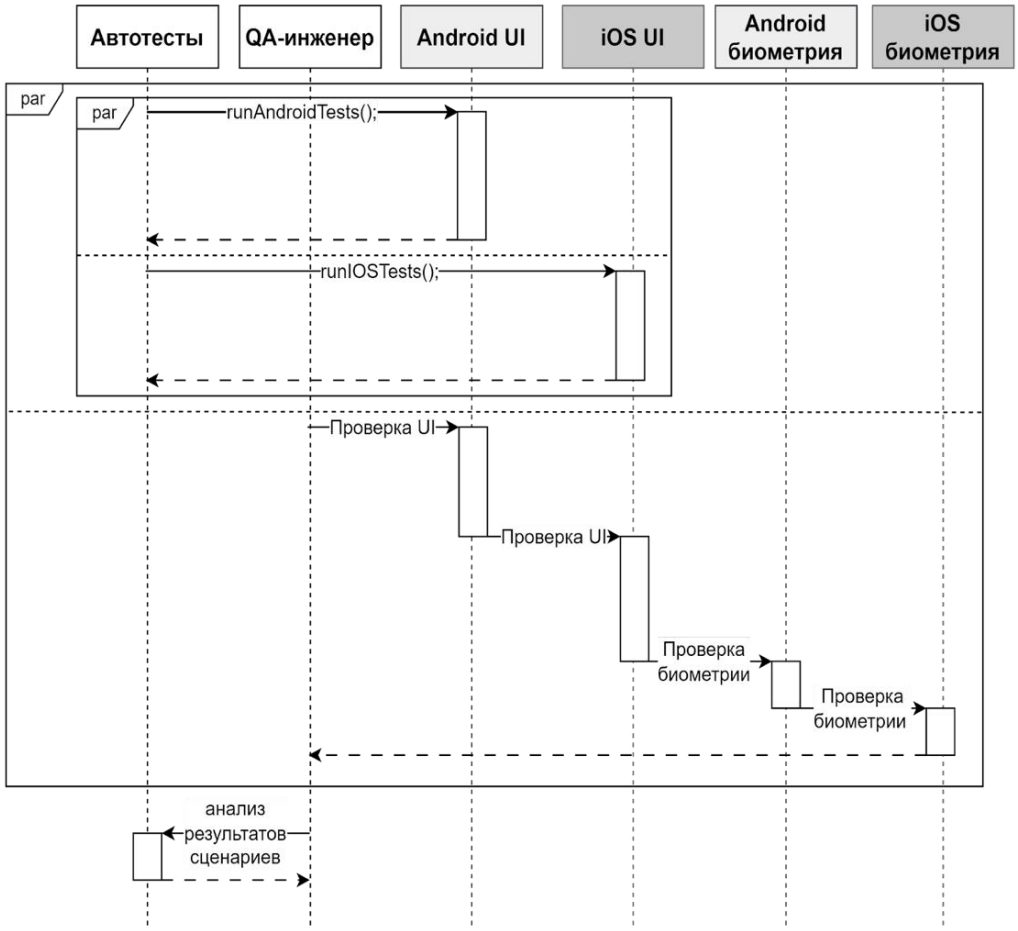

Рис. 4. – Диаграмма последовательности проверки приложений с использованием разработанного универсального решения для автоматизированного тестирования

Выводы, обсуждения

Приведенные в таблице 3 данные свидетельствуют о существенном сокращении временных затрат на процесс тестирования мобильных приложений с использованием разработанного решения.

При этом существенный вклад в ускорение процессов тестирования вносит универсальный характер реализации решения, пригодный для тестирования приложений, созданных как для Android, так и для iOS.

Предложенный авторами модульный способ построения решения позволит в дальнейшем достаточно просто нарастить функциональность решения путем добавления новых алгоритмов и режимов тестирования, существенно увеличивая «зону покрытия» автоматизированными тестовыми сценариями.

Применение описанного в настоящей статье решения позволит значительно упростить процесс тестирования приложений для мобильных устройств и существенно его ускорить.

Практическая реализация следующих этапов работ по автоматизации тестирования создаваемых мобильных приложений будет проводиться в рамках гибкой методологии разработки через тестирование (TDD — test-driven development), что, как предполагается, позволит как существенно ускорить процесс создания новых версий продуктов, так и значительно повысить качество реализации конечных решений.

Литература

1. Sommerville I., Software Engineering. 10th ed. – Pearson, 2015. 816 p.

2. Michel N., Emil A., Robert F. Why many challenges with GUI test automation (will) remain // Information and Software Technology. 2021. vol. 138: 106625.

3. Stocco A., Leotta M., Ricca F., Tonella P. APOGEN: automatic page object generator for web testing // Software Quality Journal. 2017. vol. 25, pp. 1007-1039.

4. Яловой И.О. Анализ требований и управление изменениями программных проектов // Инженерный вестник Дона, 2008, №4. URL: ivdon.ru/ru/magazine/archive/n4y2008/102.

5. Ramya, P., Sindhura V., Sagar P. Testing using selenium web Driver // 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). 2017. pp. 1-7.

6. Ubaidilah M.F., Permatasari D.I., Hardiansyah F.F. Automated Test on Multiple Platform Framework Development // 7th International Conference on Sustainable Information Engineering and Technology 2022. 2022. pp. 316-319.

7. Menegassi A.A., Endo A.T. Automated tests for cross-platform mobile apps in multiple configurations // IET software. 2020. vol. 14(1). pp. 27-38.

8. Talebipour S., Zhao Y., Dojcilovi'c L., Li C., Medvidovi'c N. UI test migration across mobile platforms // 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2021. pp. 756-767.

9. Qin X., Zhong H., Wang X. Testmig: Migrating gui test cases from ios to android // Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021. pp. 284-295.

10. Wang J., Wu J. Research on mobile application automation testing technology based on appium // 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS). 2019. pp. 247-250.

11. Morgado I.C., Paiva A.C., Faria J.P. Automated pattern-based testing of mobile applications // 2014 9th International Conference on the Quality of Information and Communications Technology. 2014. pp. 294-299.

12. Chowdhury R.R., Hossain S.S., Arafat Y., Siddiqui B.J. Configuring Appium for iOS Applications and Test Automation in Multiple Devices // Proceedings of the 2020 Asia Service Sciences and Software Engineering Conference. 2020. pp. 63-69.

13. About Android App Bundles. Date Views 21.04.2023 URL: developer.android.com/guide/app-bundle.

14. Шахраева А. Е. Проблема оценки результатов и эффективности труда персонала в инновационном процессе промышленного предприятия: терминологические аспекты исследования // Инженерный вестник Дона, 2012, №3. URL: ivdon.ru/ru/magazine/archive/n3y2012/893.

References

1. Sommerville I., Software Engineering. 10th ed. Pearson, 2015. 816p.

2. Michel N., Emil A., Robert F. Information and Software Technology. 2021. vol. 138: 106625.

3. Stocco A., Leotta M., Ricca F., Tonella P. Software Quality Journal. 2017. vol. 25, pp. 1007-1039.

4. Yalovoy I. O. Inzhenernyj vestnik Dona. 2008. №4. URL: ivdon.ru/ru/magazine/archive/n4y2008/102.

5. Ramya, P., Sindhura V., Sagar P. 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). 2017. pp. 1-7.

6. Ubaidilah M.F., Permatasari D.I., Hardiansyah F.F. 7th International Conference on Sustainable Information Engineering and Technology 2022. 2022. pp. 316-319.

7. Menegassi A.A., Endo A.T. IET software. 2020. vol. 14(1). pp. 27-38.

8. Talebipour S., Zhao Y., Dojcilovi'c L., Li C., Medvidovi'c N. 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2021. pp. 756-767.

9. Qin X., Zhong H., Wang X. Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021. pp. 284-295.

10. Wang J., Wu J. 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS). 2019. pp. 247-250.

11. Morgado I.C., Paiva A.C., Faria J.P. 9th International Conference on the Quality of Information and Communications Technology. 2014. pp. 294-299.

12. Chowdhury R.R., Hossain S.S., Arafat Y., Siddiqui B.J. Proceedings of the 2020 Asia Service Sciences and Software Engineering Conference. 2020. pp. 63-69.

13. About Android App Bundles. Date Views 21.04.2023 URL: developer.android.com/guide/app-bundle.

14. Shakhrayeva A. Y. Inzhenernyj vestnik Dona. 2012. № 3. URL: ivdon.ru/ru/magazine/archive/n3y2012/893.