УДК 004.89 **Информационные технологии**

Сарычева Юлия Юрьевна, *студент-магистр, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

Белов Юрий Сергеевич, *к.ф.-м.н., доцент, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

ОБЗОР ИНСТУМЕНТОВ ДЛЯ ИЗМЕРЕНИЯ ТЕСТОВОГО ПОКРЫТИЯ КОДА

Аннотация: Чтобы количественно определить и измерить, какая часть приложения фактически исследуется инструментом автоматического тестирования, используется тестовое покрытие. Тестовое покрытие – одна из метрик оценки качества тестирования, которая оценивает плотность покрытия тестами кода приложения либо требования.

Чтобы измерить покрытие кода, достигнутое с помощью инструмента автоматического тестирования, нужен инструмент, который может измерить покрытие кода в режиме черного ящика, то есть без доступа к исходному коду.

Для этой цели было создано несколько инструментов, таких как: ACVTool, COSMO, ELLA, BBoxTester и InsDal.

Ключевые слова: тестовое покрытие, тестирование, мобильное приложение, автоматизированное тестирование.

Abstract: Test coverage is used to quantify and measure how much of an application is actually examined by an automated testing tool. Test coverage is one of the metrics for assessing the quality of testing, which evaluates the density of test coverage of application code or requirements.

To measure code coverage achieved with an automated testing tool, you need a

tool that can measure code coverage in black box mode, that is, without access to the source code.

Several tools have been created for this purpose, such as: ACVTool, COSMO, ELLA, BBoxTester and InsDal.

Keywords: test coverage, testing, mobile application, automated testing.

Введение. Тестовое покрытие – одна из метрик оценки качества тестирования, которая оценивает плотность покрытия тестами кода приложения либо требования [1]. Разделяют несколько подходов к изменению тестового покрытия:

1. Покрытие требований (Requirements Coverage) – оценка покрытия тестами функциональных и нефункциональных требований к продукту путем построения матриц трассировки (traceability matrix).

Рассчитывается по формуле:

$T_{cov} = (L_{cov}/L_{total}) * 100\%$, где

T_{cov} – тестовое покрытие,

L_{cov} – количество требований, проверяемых тест кейсами,

L_{total} – общее количество требований.

2. Покрытие кода (Code Coverage) – оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.

Рассчитывается по формуле:

$T_{cov} = (L_{tc}/L_{code}) * 100\%$, где

T_{cov} – тестовое покрытие,

L_{tc} – количество строк кода, покрытых тестами,

L_{code} – общее количество строк кода.

3. Тестовое покрытие на базе анализа потока управления – оценка покрытия, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей [2].

Основой для тестирования потоков управления является построение

графов потоков управления (Control Flow Graph), основными блоками которых являются:

- блок процесса – одна точка входа и одна точка выхода;
- точка альтернативы – одна точка входа, две и более точки выхода;
- точка соединения – две и более точек входа, одна точка выхода [3].

Чтобы количественно определить и измерить, какая часть приложения фактически исследуется инструментом автоматического тестирования, используется измерение покрытия кода. Покрытие кода — это измерение, которое отслеживает, какая часть исходного кода программы или байтового кода выполняется во время тестового прогона. Охват может быть измерен на нескольких уровнях, например, на уровне класса, метода или инструкции.

Существует 5 критериев покрытия кода:

1) Покрытие функций — функции в исходном коде, которые вызываются и выполняются хотя бы один раз.

2) Покрытие операторов — количество операторов, которые были успешно проверены в исходном коде.

3) Покрытие пути — потоки, содержащие последовательность элементов управления и условий, которые хорошо сработали хотя бы один раз.

4) Покрытие ветвей или решений — структуры управления решениями (например, циклы), которые выполнились нормально.

5) Покрытие условий — логические выражения, которые проверяются и выполняются как TRUE, так и FALSE в соответствии с тестовыми запусками [4].

Измерение покрытия кода в режиме белого ящика поддерживается Android Studio наряду с другими инструментами. Однако для измерения охвата в условиях белого ящика необходимо иметь доступ к исходному коду. При желании протестировать стороннее приложение — это не всегда возможно.

При черном ящике также можно измерить покрытие кода. Это делается путем инструментирования байт-кода. Такой подход больше подходит для сторонних приложений, так как исходный код не всегда доступен [5].

Чтобы измерить покрытие кода, достигнутое с помощью инструмента автоматического тестирования, нужен инструмент, который может измерить покрытие кода в режиме черного ящика, то есть без доступа к исходному коду.

Для этой цели было создано несколько инструментов, таких как: ACVTool, COSMO, ELLA, BBoxTester и InsDal.

ACVTool измеряет покрытие кода на уровне инструкций, используя smali-представление байт-кода Android. По словам авторов, ACVTool может успешно инструментировать и выполнять 96,9% Android-приложений. Это также делает его подходящим для крупномасштабных испытаний, поскольку не требует слишком больших накладных расходов или длительного времени на инструментальную обработку. ACVTool создает отчеты, комбинируя отчеты времени выполнения и отчеты инструментирования, чтобы сопоставить датчики с их исходными инструкциями. Сгенерированные отчеты доступны в форматах html и xml, что делает их полезными для визуальной проверки, а также для автоматической проверки и обработки отчетов.

COSMO — это автоматизированный инструментарий, который работает как с Gradle, так и с скомпилированными приложениями. Скомпилированное приложение сначала инструментируется путем инструментирования байт-кода Java с помощью JaCoCo.

Инструментальная версия преобразуется обратно в Dalvik и добавляется в исходный файл APK. После установки и изучения приложения можно создать отчет. Авторы сообщают, что основное различие между ACVTool и COSMO для процесса скомпилированного приложения заключается в том, что разработчикам не нужно предоставлять дополнительные разрешения инструментальным приложениям. COSMO успешно тестирует 86,9% приложений, а 71,6% приложений работают без ошибок.

ELLA — инструмент, который можно использовать для измерения покрытия кода для приложений Android. Он инструментирует приложения на уровне методов, вставляя зонды и отслеживая их выполнение. Он также может отслеживать трассировку выполненных методов и значения аргументов,

переданных на сайтах вызовов, и многое другое.

На данный момент ELLA больше не поддерживается. Генерируются отчеты, содержащие список сигнатур методов и список идентификаторов методов, которые были выполнены, как и в ELLA, InsDal также измеряет покрытие кода на уровне метода.

BboxTester представляет собой инструмент, который можно использовать для создания отчетов о покрытии кода и других показателей покрытия для приложений, источник которых недоступен. BBoxTester обрабатывает JAR-файлы с помощью EMMA, которые затем собираются обратно в новый APK. BBoxTester успешно тестирует 65% приложений.

Принимая во внимание различные инструменты, ACVTool используется в экспериментах, которые выполняются из-за его высокой скорости инструментирования, небольшого времени инструментирования и низких накладных расходов. ACVTool также подходит для тестирования в больших масштабах и легко интегрируется в конвейер тестирования/экспериментов. Отчеты также понятны и просты для анализа, поэтому в качестве инструмента для выбора кода был выбран ACVTool.

ACVTool может анализировать файлы Android APK и генерировать отчет о покрытии кода без необходимости доступа к исходному коду APK. Рабочий процесс ACVTool для одного APK сводится к следующим 6 шагам:

1. Инструмент;
2. Установка;
3. Старт;
4. Тестирование;
5. Остановка;
6. Отчет.

Инструмент: ACVTool измеряет охват, вставляя зонды после каждой smali-инструкции декомпилированного APK. Apktool используется для декомпиляции, а также повторной сборки приложения вместе с apksigner и zipalign для создания работающего приложения, которое можно установить на

реальное устройство или эмулятор. Создается инструментальная версия APK, которую можно использовать на следующем этапе. Также создается инструментальный отчет, который используется для сопоставления зондов с их малыми размерами инструкции.

Установка: инструментальная версия APK, созданная на предыдущем шаге, устанавливается на эмулятор или устройство с помощью adb и готова к открытию.

Запуск: процесс инструментирования в эмуляторе запускается, вызывая процесс сбора информации о времени выполнения.

Тестирование: на этом этапе приложение тестируется и проверяется либо вручную, либо с помощью инструмента автоматизированного тестирования.

Остановка: процесс сбора информации во время выполнения останавливается, и информация, собранная во время выполнения, сохраняется в отчете об устройстве или эмуляторе.

Отчет: на этом этапе отчет о времени выполнения и отчет об инструментировании используются для сопоставления зондов с их инструкциями для создания отчета о покрытии. Отчет о покрытии доступен как в формате xml, так и в формате html, причем первый больше подходит для автоматической обработки, а второй — для визуальной проверки. В отчете охват на уровне класса, метода и инструкции.

Был проведен эксперимент для оценки генератора тестов, изучив, действительно ли он может улучшить тестовое покрытие. В нем модель использовалась для создания тестовых входов для приложений Android.

Помимо распределения событий по умолчанию, 10 различных распределений событий тестируются в тесте приложений в течение 5 минут для каждого приложения, и их покрытие кода измеряется на уровне инструкций, методов и классов. Также сообщается о количестве приложений, вышедших из строя во время тестирования. Результаты показывают, что изменение распределения событий может увеличить покрытие кода, а также количество сбоев во время тестирования с помощью Monkey или модели.

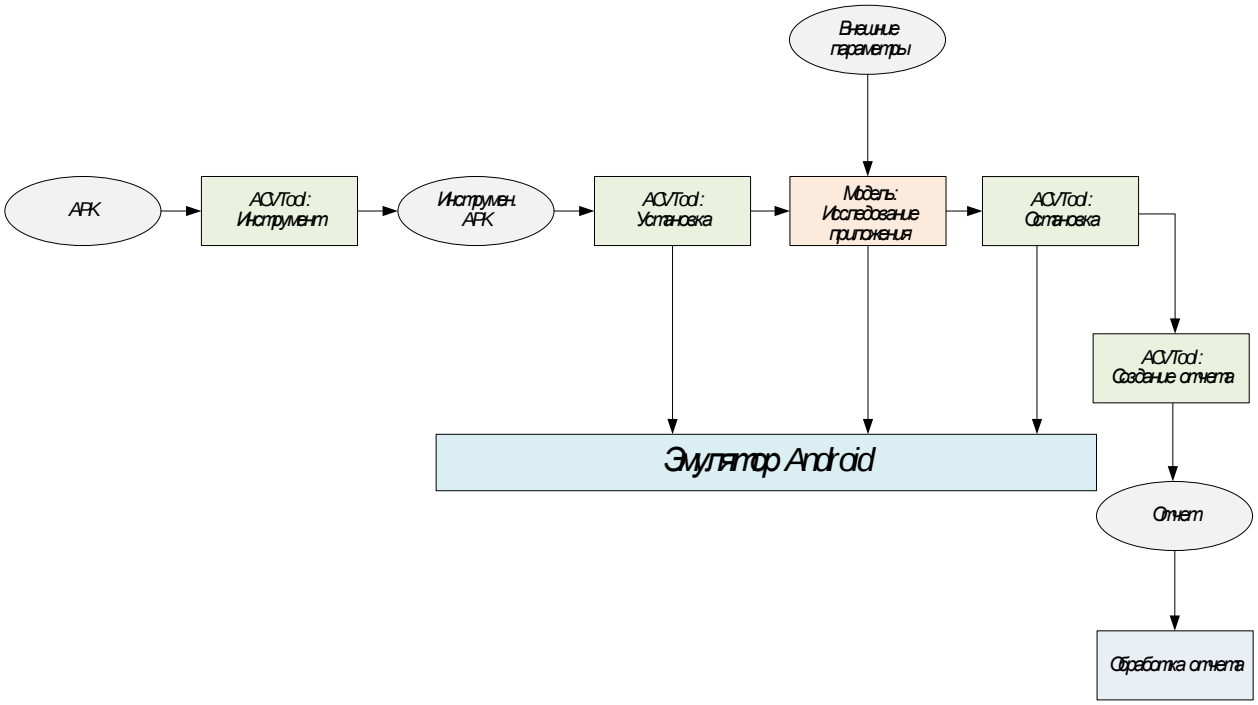

Рис. 1. Эксперимент по исследованию тестового покрытия приложения

На рис. 1 показана структура процесса эксперимента для одного APK. Этот процесс повторяется для всех APK в тесте приложений.

В начале запускается новый эмулятор, который не содержит никаких предыдущих данных или состояний. После завершения загрузки эмулятора начинается следующий шаг. В первую очередь из набора приложения выбирается APK. Затем ACVTool приступает к инструментированию этого APK, создавая инструментальную версию APK. После этого инструментированный APK устанавливается на эмулятор. После этого инструмент для автоматизации будет исследовать приложение. По истечении времени инструмент останавливается и ACVTool останавливает сбор информации о времени выполнения. Затем ACVTool создает отчет, который затем собирается программой обработки отчетов. После того, как отчет собран, процесс начинается снова для следующего APK.

После инструментирования отчет о покрытии, сгенерированный ACVTool после тестирования приложения, в формате xml должен быть проанализирован. ACVTool сообщает о покрытии на уровне инструкций, методов и классов. Чтобы

извлечь метрики покрытия из xml-отчета, будет использоваться ElementTree XML API из стандартной библиотеки Python.

Структура отчета xml показана на рис. 2. В корне дерева находится элемент отчета. Под узлом отчета находятся все пакеты приложения, под узлами пакетов — узлы классов. В узлах классов содержатся узлы методов. Резюмируя, порядок такой:

отчет → пакет → класс → метод.

В узлах метода расположены элементы-счетчики, содержащие информацию о количестве пройденных и пропущенных инструкций/методов. Элементы класса также содержат элементы счетчика об общих и пропущенных инструкциях/методах этого класса.

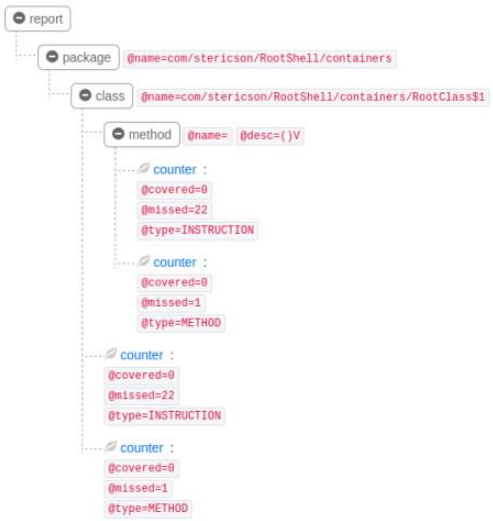

Рис. 2. Пример структуры XML-отчета, сгенерированного ACVTool

Точные показатели покрытия, определенные из этих XML-отчетов, представляют собой процент покрытых строк, процент покрытых методов и процент покрытых классов. Также извлекается общее количество строк, методов и классов.

Покрытие инструкций и методов рассчитывается путем суммирования информации обо всех элементах счетчика, которые являются прямыми дочерними элементами узлов класса, и вычисление процента покрытых

строк/методов путем деления количества строк на общее количество строк/методов [6].

Покрытие класса определяется проверкой того, покрыт ли элемент узла счетчика метода, т.е. является прямым потомком узла класса больше нуля. Затем общее количество занятий делится на количество охваченных занятий, чтобы получить процент охваченных занятий [7].

Все эти рассчитанные проценты вместе с общим количеством строк, методов и классов хранятся в CSV-файле вместе с соответствующим именем пакета APK.

Заключение. Обеспечение качества — это путь, которым тщательно занимаются команда QA и инженеры-тестировщики.

По мере роста спроса на рынке программного обеспечения организациям необходимо решать свои задачи и опережать конкурентов. Автоматизированное тестирование может сократить эти усилия, не теряя качества. Тестовое покрытие – одна из метрик оценки качества тестирования, которая оценивает плотность покрытия тестами кода приложения либо требования.

Чтобы измерить покрытие кода, достигнутое с помощью инструмента автоматического тестирования, нужен инструмент, который может измерить покрытие кода в режиме черного ящика, то есть без доступа к исходному коду.

В данной статье было рассмотрено несколько инструментов, таких как: ACVTool, COSMO, ELLA, BBoxTester и InsDal.

Библиографический список:

1. Naja Fa., Mansur Sy., Wibawanto Ad. Automated Software Testing on Mobile Applications: A Review with Special Focus on Android Platform // 20th International Conference on Advances in ICT for Emerging Regions. 2020. pp. 4-6.

2. Василенко, Р. И. Автоматизированное тестирование мобильных приложений / Р. И. Василенко, С. А. Белоусова // Инновационные технологии и дидактика в обучении: сборник статей III Международной научно-практической конференции, Краснодар, 29–30 июня 2015 года / Борисова Е.А. – Краснодар:

Издательство Южного Федерального Университета, 2015. – С. 31-34. – EDN UTWWDN.

3. Михалевская, К. А. Сравнение инструментов для автоматизации тестирования мобильных приложений на ОС Android / К. А. Михалевская, М. А. Сергачева, И. Н. Мерзляков // КОГРАФ - 2020: сборник материалов 30-й Всероссийской научно-практической конференции по графическим информационным технологиям и системам, Нижний Новгород, 13–16 апреля 2020 года. – Нижний Новгород: Нижегородский государственный технический университет им. Р.Е. Алексеева, 2020. – С. 250-255. – DOI 10.46960/43791586_2020_250. – EDN XZGGPU.

4. Сарычева, Ю.Ю., Белов Ю.С. Применение искусственного интеллекта в автоматизированном тестировании GUI // Научные исследования в современном мире. Теория и практика: сборник избранных статей Всероссийской (национальной) научно-практической конференции. 2022. С. 55-56.

5. Ананьев, В. Ю. Автоматизированное тестирование мобильных приложений при помощи nunit и Appium / В. Ю. Ананьев // Инновационные технологии, экономика и менеджмент в промышленности: Сборник научных статей X международной научной конференции, Волгоград, 21–22 октября 2021 года. – Волгоград: Общество с ограниченной ответственностью "КОНВЕРТ", 2021. – С.136-138. – EDN BBDGXU.

6. Pan M., Xu To., Pei Yu. GUI-Guided Test Script Repair for Mobile Apps // IEEE Transactions on Software Engineering. 2022. Vol. 48. №3. pp. 3-5.

7. Сарычева, Ю. Ю. Тестирование GUI мобильного приложения при помощи monkey // E-Scio. 2022. №6 (69). URL: https://cyberleninka.ru/article/n/testirovanie-gui-mobilnogo-prilozheniya-pri-pomoschi-monkey (дата обращения: 24.12.2022).