УДК 004.415.53

Андреева Т. И.,
студент магистратуры
2 курс, факультет «Информационные системы и технологии»
Пензенский государственный университет архитектуры и строительства
Россия, г. Пенза
Гвоздева И. Г.
старший преподаватель
Пензенский государственный университет архитектуры и строительства
Россия, г. Пенза
Андреев А.И.,
преподаватель,
ГАПОУ ПО "Пензенский колледж архитектуры и строительства"
Россия, г. Пенза

АВТОМАТИЗИРОВАНИЕ ТЕСТИРОВАНИЕ НА ЗАГЛУШКАХ ПРИ ПОМОЩИ ФРЕЙМВОРКА CITRUS

Аннотация

Статья посвящена освещению основных моментов интеграционного тестирования программного обеспечения, применение заглушек в автотестировании и результат использования фреймворка Citrus.

Ключевые слова:

Тестирование, автоматизированное тестирование, интеграционное тестирование, тестирование на заглушках, Citrus-framework.

Andreeva T. I.,
student of magisterial
2course, faculty “ Information systems and technologies”
Penza State University of architecture and construction
Russian Federation, Penza
Gvozdeva I.G.,
senior teacher
Penza State University of architecture and construction
Russian Federation, Penza
Andreev A.I.,
teacher,
State Autonomous Professional Educational Institution of the Penza
Region "Penza College of architecture and construction"
Russian Federation, Penza

AUTOMATION TESTING ON MOCK WITH CITRUS FRAMEWORK

Summary

The article is devoted to inform the main points of integration software testing, the use of stubs in auto-testing and the results of using the Citrus framework.

Key words:

Testing, automation testing, integration testing, mock-testing, Citrus-framework.

Тестирование программного обеспечения – постоянно развивающееся направление, начиная от методов тестирования и заканчивая инструментами как функционального, так и автоматизированного тестирования. Одним из видов тестирования является – интеграционное тестирование. [1]

Интеграционное тестирование - это фаза тестирования программного обеспечения, на которой отдельные программные модули комбинируются и тестируются в группе.

Основной целью интеграционного тестирования является подтверждение того, что результаты взаимосвязи между двумя и более компонентами отвечают функциональным требованиям.[2]

Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Исходя их «правильной» пирамиды тестирования интеграционное тестирование, находясь на ступень выше модульного, занимает около 10% от общего объема тестов.

На этапе интеграционного тестирования составляется тест план, по которому проверяется работоспособность системы, и далее пишутся тест-кейсы. При ограниченности человеко-ресурсов или сроков вместо тест-кейсов могут быть написаны чек-листы.

Интеграционное тестирование предполагает собой объединение модулей. Но зачастую возникают ситуации, когда один модуль может быть написан раньше другого, приложение, использующее API-сервис уже готово к тестированию, а сервис еще нет, обнаруженный дефект препятствует тестированию. Из-за высокой цены, ожидание «благополучного» стечения обстоятельств не может оправдать себя. Одним из выходов из ситуации может стать введение автоматизированного тестирования на заглушках.

На данный момент IT-рынок предоставляет большой выбор фреймворков для работы с заглушками. Среди них Citrus, AuTe Framework, Mockito, Wiremock и другие. В данной статье будет освещен процесс и результат работы с инструментом Citrus.

Citrus - это фреймворк с открытым исходным кодом, который может автоматизировать тесты для практически любого протокола обмена сообщениями или формата данных. [3]

Citrus подходит для тестирования интеграции обмена сообщениями при использовании любой из технологии передачи сообщений: HTTP, REST SOAP или JMS. [3]

Тестируемое приложение обменивается данными через шину с помощью настройщика очередей IBM MQ. Поэтому для тестирования на заглушках, необходимо, чтобы Citrus перехватывал сообщения, отправляемые приложением в MQ, и возвращал ответ. Для этого в настройках тестируемого приложения необходимо переопределить хост, куда будут отсылаться запросы, а так же указать имена очередей. Те же самые параметры необходимо указать и для Citrus. [3]

Для корректной работы фреймворка, в Citrus необходимо прописать сценарии возможных запросов и ответов на них. Каждый этап обмена информацией связан с проверкой и сопоставлением сообщений с ожидаемым шаблоном сообщения (например, данными XML или JSON). Помимо действий с сообщениями, Citrus также может выполнять произвольные другие тестовые действия. В качестве примера Citrus может выполнять запрос базы данных между запросами.

Сценарий представляет из себя spring-код, в котором указаны шаблоны входящего и исходящего сообщений. Далее будет представлен пример сценария.

```
<spring:beans
    xmlns="http://www.citrusframework.org/schema/testcase"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.citrusframework.org/schema/testcase
```

```
    http://www.citrusframework.org/schema/testcase/citrus-testcase.xsd">
    <testcase name="MyFirstTest">
        <description>
            First example showing the basic test case definition elements!
        </description>
        <variables>
            <variable name="text" value="Hello Test Framework"/>
        </variables>
        <actions>
            <echo>
                <message>${text}</message>
            </echo>
        </actions>
    </testcase>
</spring:beans>
```

Нередко на практике встречаются тестовые сценарии, которые меняют поведение системы (подключение услуги, перерасчеты и прочее). В таких тестах ответ от «шины» должен подчиняться некоторым условиям, которые задаются в запросе. Эту проблему можно решить с помощью ведения глобальных переменных.

Глобальные переменные находятся в отдельном файле и могут использоваться в любое паре request-response для любого сценария. Но мало просто подставить данные, необходимо еще их изменять в режиме реального времени. В этом может помочь использование groovy-скриптов. Таким образом, в связке с глобальными переменными и скриптами можно воспроизводить сложные тестовые сценарии. Однако это может оказаться плюсом только в том случае, если таковых сценариев мало. В большом проекте, как показала практика, все это сводится к огромным скрипт-файлам, которыми становится трудно управлять.

Фреймворк можно запустить как службу, и она будет хранить состояние переменных, скриптов вплоть до следующего перезапуска. Это влечет за собой проблемы, невозможности предугадать состояние заглушек на данный момент времени. Временным решением проблемы может стать перезапуск службы перед каждым запуском автотестов.

Разработка автотестов и активное их использование в течение трех месяцев показало ряд минус данного фреймворка:

1. Отдельный проект – для отладки и написания автотестов на заглушках необходимо держать открытыми два IDE[1]
2. Все пары request-response вшиты в проект и нет возможности управлять ими через API[2]
3. Сложно под один request сделать несколько response.
4. Для тестов, в которых необходимо менять поведение системы, необходимо использовать глобальные переменные и groovy-скрипты
5. Для сброса значений глобальный переменных необходим перезапуск
6. Нельзя запустить параллельно 2 одинаковых теста, которые изменяют поведение системы

[1] IDE – интегрированная среда разработки
[2] API - программный интерфейс приложения— описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

Список литературы:

1. Куликов С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 2 издание — Минск: Четыре четверти, 2017. — 312 с.
2. Интеграционное тестирование [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Интеграционное_тестирование. - (Дата обращения 28.11.2018)
3. Citrus Integration Tests [Электронный ресурс]. – Режим доступа: https://citrusframework.org/docs/home/. (Дата обращения: 27.11.2018)