ПОДХОДЫ К АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ

Жданко Татьяна Владимировна

магистрант Белорусского государственного университета информатики и радиоэлектроники, Республика Беларусь, г. Минск

Большинство программных продуктов, выпускаемых сегодня, являются веб-ориентированными приложениями, рассчитанными на работу в интернет-браузере. Разработка веб-приложения подразумевает под собой решение множества задач из совершенно разных областей, таких как: работа с большими объемами данных, вычислительные системы, сетевая безопасность, дизайн и прочее. Тестирование является важной и неотъемлемой частью разработки программного обеспечения. Возрастающее количество исходного кода, а также возрастающее число разработчиков, вызывает все большее количество ошибок в разрабатываемом приложении. Одной из основных функций тестирования является обнаружение этих ошибок как можно раньше. Скорость, с которой можно идентифицировать эти ошибки, полностью зависит от подхода тестирования, количества тестировщиков, их опыта, а также инструментов, используемых в процессе тестирования.

Существует множество подходов к автоматизации тестирования. Все чаще мы слышим такие аббревиатуры, как TDD (англ. Test Driven Development), BDD (англ. Behaviour Driven Development), KDT (англ. Keyword Driven Testing), DDT (англ. Data-driven testing) и многие строго следуют этим подходам в разработке.

Тестирование, управляемое данными (Data-Driven Testing) представляет собой такой подход к тестированию, при котором тестовые данные хранятся отдельно от тест-кейсов, например, в файле или базе данных. Такое разделение делает тесты логически более простыми [1].

Данный подход применяется на проектах, где требуется тестирование отдельных приложений в нескольких средах с большими наборами данных и стабильными тест-кейсами.

Как правило, тестирование, управляемое данными, выполняет следующие операции:

- извлекает часть тестовых данных из хранилища;
- водит данные в форму приложения;
- проверяет результаты;
- продолжает тестирование со следующим набором входных данных.

На рисунке 1 представлен DDT подход.

Жданко Т.В. Подходы к автоматизации тестирования веб-приложений // Студенческий форум: электрон. научн. журн. 2017. № 21(21). URL: https://nauchforum.ru/journal/stud/21/29539 (дата обращения: 09.09.2025).

Рисунок 1. Data-Driven Testing подход

Для успешной проверки приложения требуются разные комбинации данных.

Одним из инструментов программной поддержки DDT является TestComplete, который включает в себя ряд специальных функций и программных объектов, которые помогут подготовить и получить доступ к тестовым данным.

Keyword Driven Testing (Тесты, управляемые ключевыми словами) – это подход, в котором используются ключевые слова, описывающие набор действий, необходимых для выполнения определенного шага тестового сценария. Сперва определяется набор ключевых слов, а затем ассоциируется действие (или функция), связанное с эти ключевым словом. Т.е. каждый шаг теста, такой как открытие или закрытие браузера, щелчок мышью, нажатие клавиши и т.д., описывается ключевым словом, таким как «открыть» (openbrowser), «нажать» (click).

При KDT тестировании вы можете создавать простые функциональные тесты на ранних этапах разработки, тестируя приложение по частям. Самый простой способ составить KDT тест, записать его. После записи тест можно изменить и настроить в соответствии с требованием. Когда тест-кейсы выполняются, ключевые слова интерпретируются тестовой библиотекой.

Основные этапы разработки KDT тестов:

Шаг 1. Определение ключевых слов;

Шаг 2. Реализация ключевых слов как исполняемых файлов;

Шаг 3. Создание тест-кейсов;

Шаг 4. Создание скриптов;

Шаг 5. Выполнение автоматизированных сценариев.

Преимущества данного подхода:

1) Данный подход позволяет функциональным тестировщикам планировать автоматизацию тестирования до того, как разрабатываемое приложение будет готово.

2) Тесты могут быть разработаны без знания программирования.

3) Данный подход не зависит от конкретного языка программирования или инструмента [2].

Методика разработки через тестирование (TDD) заключается в организации автоматического тестирования разрабатываемых приложений путем написания модульных, интеграционных и функциональных тестов, определяющих требования к коду непосредственно перед написанием этого самого кода. Сначала пишется тест, который проверяет корректность работы еще ненаписанного программного кода. Этот тест, разумеется, не проходит. После этого разработчик пишет код, который выполняет действия, требуемые для прохождения теста. После того, как тест успешно пройден, по необходимости осуществляется доработка написанного кода, причём под контролем прохождения тестов.

Рисунок 2. Test Driven Development подход

Разработка через тестирование предлагает больше, чем просто проверку корректности, она также влияет на дизайн программы. Изначально сфокусировавшись на тестах, проще представить, какая функциональность необходима пользователю. Таким образом, разработчик продумывает детали интерфейса до реализации. Все это помогает сократить время на разработку и отладку программы.

Однако стоит отметить, что разработку через тестирование сложно применять в тех случаях, когда для тестирования необходимо прохождение функциональных тестов. Примерами может быть: разработка интерфейсов пользователя, программ, работающих с базами данных, а также того, что зависит от специфической конфигурации сети. Разработка через тестирование не предполагает большого объёма работы по тестированию такого рода вещей. Она сосредотачивается на тестировании отдельно взятых модулей, используя mock-объекты (заглушки) для представления внешнего мира.

BDD — разработка, основанная на поведении. BDD, это разновидность TDD, разница в том, что BDD ориентирован на поведение сущности, которую вы тестируете, в то время как в TDD весь фокус идет на сам код. Суть BDD — в описании системы архитектуры приложения в терминах эксперта предметной области, а не программиста, что позволяет ускорить процесс получения обратной связи и убрать традиционные языковые барьеры между создателями ПО и его пользователями.

Одним из инструментов программной поддержки BDD является Cucumber. Тесты пишутся на простом языке управляемой поведением разработки (BDD) в стиле Given, When, Then

(условия, операция, результат), которой понятен любому пользователю. Затем контрольные тесты записываются в файлы функций, охватывающие один или несколько сценариев тестирования. Cucumber интерпретирует тесты на указанном языке программирования и использует Selenium для управления тестами в браузере.

BDD-синтаксис Given, When, Then интуитивно понятен. Элементы синтаксиса:

- Given предоставляет контекст выполнения сценария тестирования, например, точки вызова сценария в приложении, а также любые необходимые данные.

- When определяет набор операций, инициирующих тестирование, таких как действия пользователей или подсистем.

- Then описывает ожидаемый результат тестирования.

На рисунке 3 представлен тест входа в систему для простого веб-приложения.

```
@run
Feature: Test Login

Scenario Outline: Login Success and Failure
    Given I navigate to the mock application
    When I try to login with '<type>' credentials
    Then I should see that I logged in '<status>'

Examples:
    | type      | status          |
    | valid     | successfully    |
    | invalid   | unsuccessfully  |
```

Рисунок 3. Листинг тестового сценария

Как мы видим, Cucumber предоставляет возможность повторно использовать этот тест путем выбора значений из таблицы. Cucumber читает указанные файлы функций и выполняет тесты, используя указанные теги (@run). Для интерпретации файлов функции создается класс, где каждый метод имеет аннотации Given, When или Then, содержащие регулярные выражения, соответствующие строкам файла функций [3].

Основные преимущества Cucumber:

1) Тесты понятны всем - самим тестировщикам, программистам, менеджерам и заказчикам. Можно смело отправлять их на анализ команде разработки или заказчику и получить ценные рекомендации по их улучшению.

2) Очень высокая скорость разработки новых тестов. Даже начинающие инженеры по тестированию, посмотрев примеры уже созданных тестов, могут быстро и качественно создать множество подобных тестов, тестирующих уже другой функционал. В среднем во время интенсивной разработки возможно написание несколько десятков автоматизированных тестов в день, при этом не написав ни строчки кода. Возможно создание тестовых сценариев из готовых строчек текста.

3) Ненадобность логирования при написании тестов – каждый степ (действие пользователя) по сути своей является логированием.

4) При описании дефектов не нужно придумывать шаги для воспроизведения, т.к. необходимые шаги берутся из отчета.

В процессе разработки новых тестов так правило не изменяется исходный код уже созданных шагов, а значит, теоретически сразу же снижается возможность внесения дефектов в ранее разработанные и проверенные тесты [4].

Список литературы:

1. Jazzteam [Электронный ресурс]. – Электронные данные. – Режим доступа: https://jazzteam.org/ru/technical-articles/data-driven-testing/.

2. Guru99 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.guru99.com/keyword-driven-testing.html.

3. IBM [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.ibm.com/developerworks/library/a-automating-ria/.

4. Quora [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.quora.com/What-are-the-disadvantages-of-BDD.