

Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Баев Дмитрий Олегович

**Разработка системы построения отчетов автотестов,
написанных на разных языках программирования**

Научный руководитель: Ерошенко Артем Михайлович, старший инженер
по автоматизации тестирования, компания Яндекс

Санкт-Петербург
2014

Содержание

Введение	4
Глава 1. Постановка задачи	6
1.1 Термины и понятия	6
1.1.1 Тестирование	6
1.1.2 Сокращения	8
1.2 Обоснование актуальности	8
1.3 Обзор существующих систем	9
1.3.1 Surefire	9
1.3.2 Thucydides	9
1.4 Технический и организационный контекст	10
1.5 Уточненные требования к работе	10
Глава 2. Теоретические результаты	12
2.1 Общая схема работы	12
2.1.1 Listener	12
2.1.2 Programming language API	12
2.1.3 XML model	14
Глава 3. Результаты	15
Заключение	16

Введение

Бывают модульные тесты, а бывают высокоуровневые. И когда их количество начинает расти, анализ результатов тестов становится проблемой. Дело в том, что высокоуровневые тесты сильно отличаются от модульных, и обладают рядом особенностей:

- они затрагивают гораздо больше функциональности, что затрудняет локализацию проблемы;
- такие тесты воздействуют на систему через посредников, например, браузер;
- таких тестов очень много, и зачастую приходится вводить дополнительную категоризацию. Это могут быть компоненты, области функциональности, критичность.

В рамках стандартной модели xUnit анализировать результаты таких тестов достаточно проблематично. Например, в ошибке «Can not click on element «Search Button»» тесте на web-интерфейс может произойти по следующим причинам:

- сервис не отвечает;
- на странице нет элемента «Search Button»;
- элемент «Search Button» есть, но не получается на него кликнуть;

А имея дополнительную информацию о ходе выполнения теста, например, лог работы сервиса и скриншот страницы, локализовать проблему гораздо легче.

Отсюда возникает следующая задача: разработать такую систему, которая позволяет агрегировать дополнительную информацию о ходе выполнения тестов и строить отчет.

В данной работе будет описан процесс разработки такой системы.

Во второй главе, на основании анализа различных систем построения отчетов автотестов, а также опыта написания тестирования, сформулированы основные принципы для организации системы.

В третьей главе приведена подробная архитектура Allure, позволяющая легко интегрироваться с любыми существующими тестовыми фреймворками и расширять имеющийся функционал. Подробно описана интеграция новых фреймворков, и новых систем сборки.

В заключении дано описание текущего состояния разработки и перспективы ее развития.

Глава 1. Постановка задачи

1.1. ТЕРМИНЫ И ПОНЯТИЯ

В данном разделе описаны термины, используемые других частях представленной работы. При этом смысл многих терминов сужен, по сравнению, с их обычным смыслом. Это связано, с тем, что данная работа ориентирована в первую очередь, разработку системы построения отчетов автотестов. В дальнейшем приведенные термины будут использоваться в указанных значениях, если не оговорено обратное.

1.1.1. Тестирование

Аттачмент (attachment) — любая информация, например, скриншот или лог, которую надо сохранить вместе с результатами теста.

История (user story, story) — модуль, часть функциональности, из которых может состоять требование.

Контекст теста (test context, test fixture) — все, что нужно тестируемой системе чтобы мы могли ее протестировать. Например, наглядно понятно, что такое контекст теста в тестовом фреймворке RSpec:

- контекст — множество фруктов содержащих = яблоко, апельсин, грушу;
- экспертиза — удалим апельсин из множества фруктов;
- проверка — множество фруктов содержит = яблоко, груша.

Ошибка теста (test error) — ошибка, возникающая в ходе выполнения теста. Например, ошибка может возникнуть в проверяемой системе, или в самом тесте. Также ошибка может возникнуть в окружении (например, в операционной системе, виртуальной машине). Как правило, ошибка в самом тесте, а не в проверяемой системе.

Падение теста (test failure) — тест падает, когда в проверке утверждений актуальное значение не совпадает с ожидаемым. Обычно означает наличие ошибки в проверяемой системе.

Разработка через тестирование (TDD, test-driven development) — техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов:

- написание теста на новую/изменяемую функциональность;
- имплементация функциональности. Тест должен пройти;
- рефакторинг кода под соответствующие стандарты разработки.

Разработка через требования (BDD, behavior-driven development) — Разновидность разработки через тестирование, сфокусированная на тестах в которых четко описаны ожидаемые требования к тестируемой системе. Упор делается на то, что тесты используются как документация работы системы.

Результат теста (test result) — тест, или тест суит могут быть запущены несколько раз, и каждый раз возвращать различные результаты проверок.

Тест — некоторая процедура, которая может быть выполнена вручную или автоматически, и может быть использована для проверки ожидаемых требований к тестируемой системе. Тест часто называют тесткейсом.

Тест кейс (test case) — обычно синоним для понятия "тест". В xUnit это также может обозначать тестовый класс, как `testCase` в котором содержит тестовые методы.

Тест прошел (test success) — ситуация, в которой проверка каждого утверждения в тесте прошла успешно (актуальные значения совпали с ожидаемыми), и в процессе выполнения теста не произошло никаких ошибок теста.

Тест ран (test run) — запуск некоторого числа тестов или тестсуитов. После выполнения тестов из тестрана, мы можем получить их результаты.

Тест суит (test suite) — способ наименования некоторого числа тестов, которые могут быть запущены вместе.

Тестируемая система (System Under Test) — любая вещь, которую мы проверяем, например, метод, класс, объект, приложение.

Требование (feature) — часть функциональности развивающейся системы, которая может быть протестирована.

Шаг (step) — некоторая логическая часть теста. Каждый тест может состоять из одного или нескольких шагов. Как правило, шаги отображают сценарий теста.

Шаг теста (test step) — смотри "Шаг".

xUnit — под этим термином подразумевается любой член семейства инфраструктур автоматизации тестов (Test Automation Framework), применяемых для автоматизации созданных вручную сценариев тестов. Для большинства современных языков программирования существует как минимум одна реализация xUnit. Обычно для автоматизации применяется тот же язык, который использовался для написания тестируемой системы. Хотя это не всегда так, использовать подобную стратегию проще, поскольку тесты легко получают доступ к программному интерфейсу тестируемой системы.

WebDriver — Утилита, позволяющая эмулировать действия пользователя в различных браузерах.

Большинство членов xUnit реализованы с использованием объектно-ориентированной парадигмы.

1.1.2. Сокращения

SUT — System Under Test, смотри "Тестируемая система".

1.2. ОБОСНОВАНИЕ АКТУАЛЬНОСТИ

На данный момент, практически отсутствуют системы, которые позволяют решать поставленную задачу. Для xUnit тестов можно построить

стандартный отчет, но он подходит только для модульного тестирования, то есть в нем нет возможности сохранения дополнительной информации о ходе теста, сценария теста. Также у некоторых тестовых фреймворков есть свои отчеты (например, Thucydides), но они узконаправленные, и позволяют действовать только в рамках соответствующих фреймворков. Для большинства членов семейства xUnit систем построения отчетов, кроме стандартного, нет.

Из-за отсутствия универсального решения, и наличия большого числа высокоуровневых тестов, анализ результатов отнимает очень много времени и сил. И часто узкое причина длинного цикла разработки именно затянувшийся процесс тестирования.

1.3. ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ

1.3.1. Surefire

Семейство xUnit предлагает стандартный отчет, который называется surefire. Это простой отчет, содержащий список тестов, и для каждого теста информацию о его статусе, времени работы и сообщения об ошибке (если имеется). Данное решение подходит для анализа результатов модульных тестов, но не годится в рамках поставленной задачи.

1.3.2. Thucydides

Thucydides — тестовый фреймворк на основе junit. Он предоставляет возможность писать WebDriver'ные тесты, есть возможность разбивать тесты на шаги, и сохранять скриншоты. Основным недостатком данной системы является то, что она слишком узконаправленная, и накладывает слишком много ограничений на как тесты и тестируемые системы.

1.4. ТЕХНИЧЕСКИЙ И ОРГАНИЗАЦИОННЫЙ КОНТЕКСТ

Рассмотрим структуру разработки системы отчетов автотестов. Автор работы является основным разработчиком фреймворка (Allure), активно взаимодействует с остальными участниками разработки. Основным заказчиком является отдел тестирования компании Яндекс, в лице Ерошенко Артема Михайловича, который также является основным идейным вдохновителем и руководителем разработки.

Первый прототип, а так же первые две версии были спроектированы и разработаны автором данной работы, совместно с Артемом Михайловичем. Дальше к разработке присоединился профессиональный front-end разработчик Сердюк Борис Дмитриевич, который переработал "морду" отчета, и до сих пор активно участвует в поддержке и развитии фреймворка.

Изначально перед автором стояла задача предложить структуру фреймворка, разработать прототип и адаптировать фреймворк под работу с junit и pytest.

Общую схему работы фреймворка можно увидеть на рисунке 2.1.

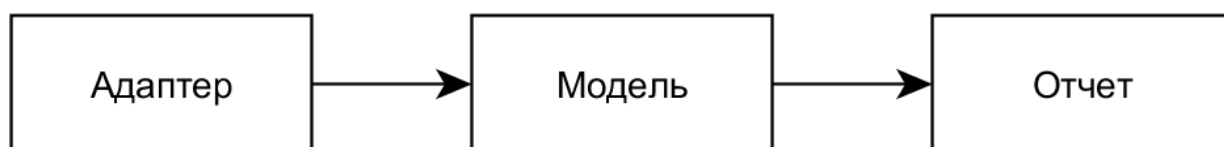


Рис. 1.1: Общая схема работы Allure

Фактически, работа фреймворка состоит из двух частей. Сначала надо собрать данные о ходе выполнения тестов, а затем сгенерировать из них отчет.

1.5. УТОЧНЕННЫЕ ТРЕБОВАНИЯ К РАБОТЕ

Обобщая вышесказанное, выведем следующие основные цели данной работы:

- разработать систему, позволяющую собирать дополнительную информацию о тестах и отображать ее в виде отчета:
 - система должна легко подключаться к большому числу уже написанных тестов;
 - система должна легко адаптироваться под разные языки программирования;
 - система должна быть модульной, легко расширяться;
- написать первый прототип системы;
- реализовать первую версию программы;
- протестировать систему в реальных условиях;
- проанализировать результаты работы системы.

Результатом данной работы будет являться готовый роботоспособный фреймворк, активно используемый в тестировании в качестве универсального способа отображения результатов работы автотестов.

Глава 2. Теоретические результаты

В данной главе рассматриваются теоретические аспекты разработки фреймворка. Будет приведена общая схема фреймворка, описаны основные модули. Также будет рассказано о ходе выполнения тестов и приведены примеры использования работы на примере JUnit тестов.

2.1. ОБЩАЯ СХЕМА РАБОТЫ

Общая схема работы фреймворка показана на рисунке 2.1. Рассмотрим подробнее назначение отдельных частей.

2.1.1. Listener

Для большинства тестовых фреймворков xUnit есть возможность подключить листенер для сбора информации о ходе тестов. Мало того, подключение листенера, как правило, вынесено на уровень конфигурации запуска, что полностью удовлетворяет требованиям работы. Для адаптации тестового фреймворка достаточно реализовать тест листенер используя соответствующее API языка программирования.

Однако стоит заметить, что не всю необходимую информацию о ходе теста можно собрать используя листенер, так как он оперирует терминологией xUnit. Сбор остальной информации о тестах, например информацию о пройденных шагах и сделанных аттачментах, будет реализован на уровне API языка программирования.

2.1.2. Programming language API

API для языка программирования представляет из себя набор обработчиков событий и сами события, используя которые можно полностью описать жизненный цикл теста. Программный интерфейс содержит в себе следующие события:

- начало/конец тестового запуска;

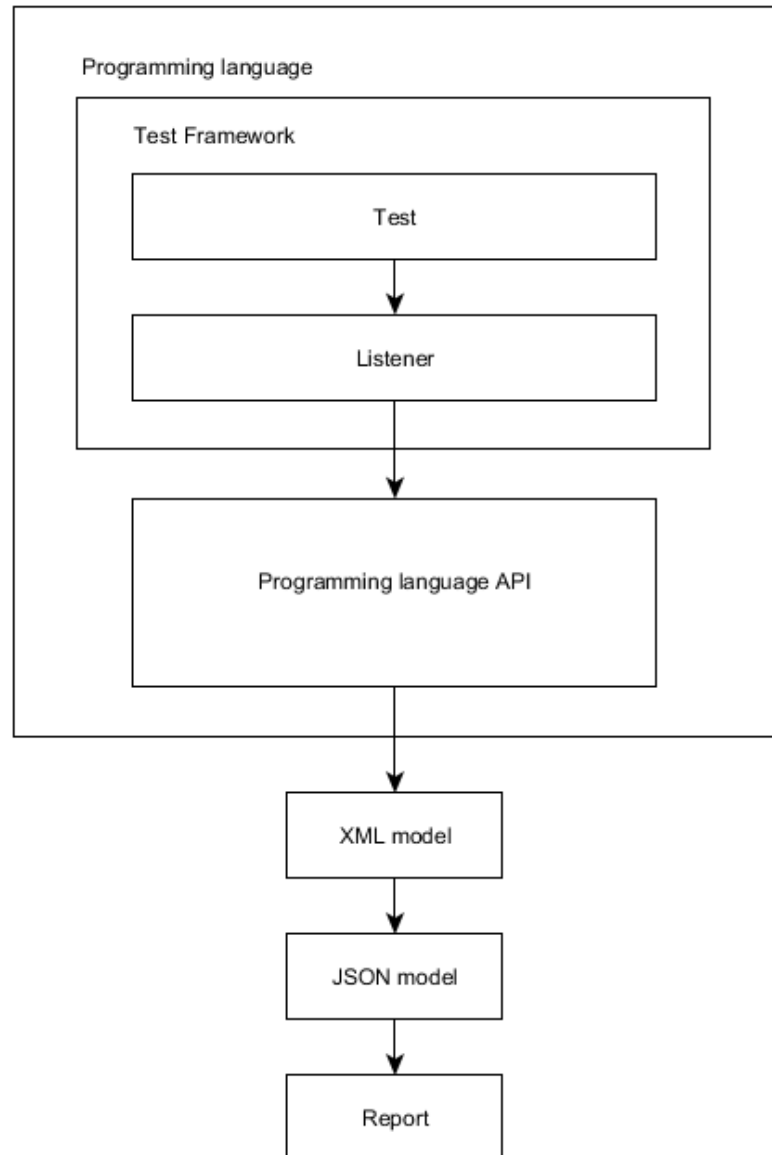


Рис. 2.1: Общая схема фреймворка Allure

- начало/конец тест суита;
- начало/конец тест кейса;
- начало/конец шага;
- сохранение аттачмента;
- добавление параметров запуска/тест суита/тест кейса;
- изменение статуса теста/шага;
- добавление пометок к тесту.

С использованием API для языка программирования сильно упрощается написание и поддержка листнеров для тестовых фреймворков. Вся

собранная информация о ходе тестов сохраняется в XML модель.

2.1.3. XML model

Собранная о тесте информация серелизуется в виде XML файлов. Для каждого теста создается свой файл. Сохраняются только те данные, которые нельзя синтезировать, что упрощает реализацию и поддержку интерфейса для языка программирования. Простейший пример сохраненной информации об одном тесте:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:test-suite xmlns:ns2="urn:model.allure.qatools.yandex.ru"
  start="1400681607876" stop="1400681627123">
  <name>my.company.SampleTest</name>
  <test-cases>
    <test-case start="1400681608883" stop="1400681608891"
      status="passed">
      <name>test_pass</name>
    </test-case>
  </test-cases>
  <labels/>
</ns2:test-suite>
```

Глава 3. Результаты

Заключение