Автоматическое обновление сценариев нагрузочных тестов на основе отслеживания сетевой активности функциональных тестов веб-приложения

Колобов Роман Евгеньевич

научный руководитель: С. М. Андреев

СПб АУ НОЦНТ РАН

14 июня 2016 г.

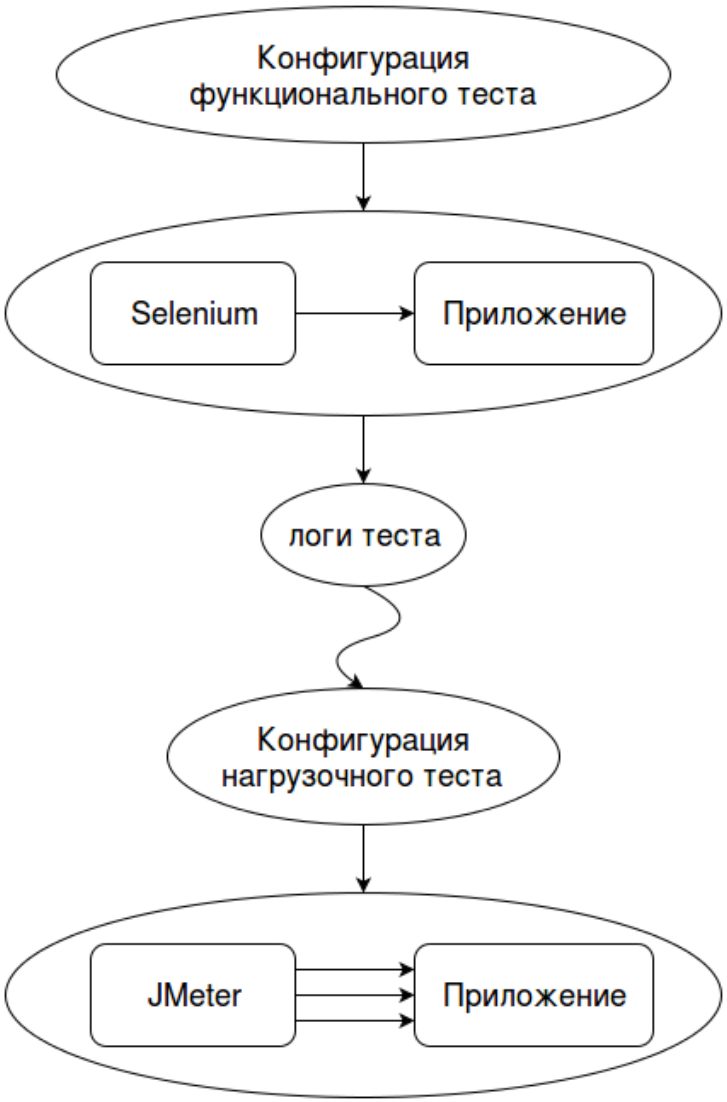
Введение

- Функциональное тестирование веб-приложений; Selenium WebDriver
- Нагрузочное тестирование веб-приложений; JMeter
- Основной принцип создания нагрузочных тестов: мониторинг сетевой активности тестировщика / пользователей / функциональных тестов
- Continuous delivery, автоматизация тестирования
- Сложность актуализации нагрузочных автотестов

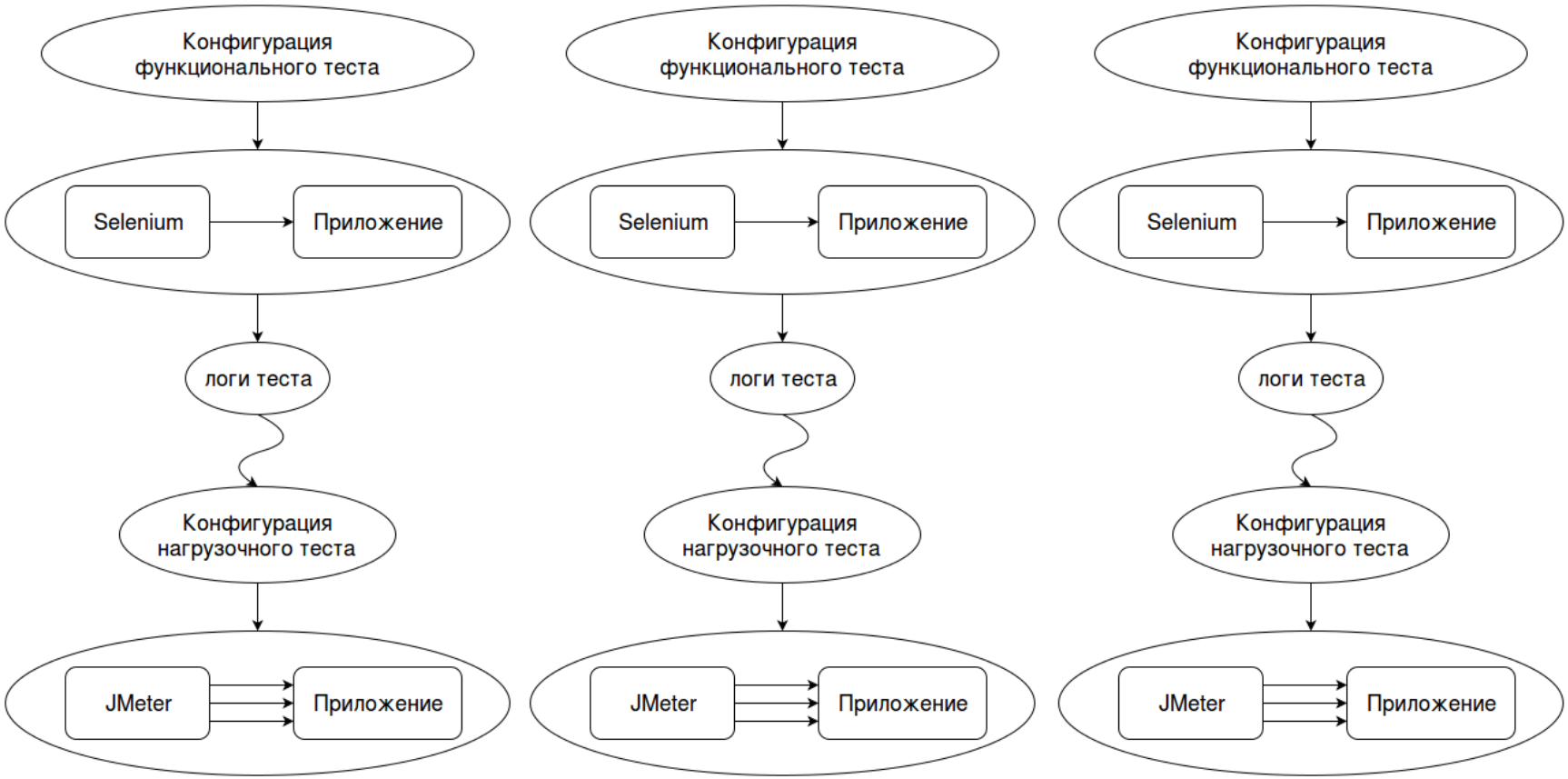
Актуализация нагрузочных тестов

- Selenium Grid
  - ресурсоёмко
  - тестируется не только время отклика сервера
  - страдает воспроизводимость результатов
- Генерация каждого нагрузочного сценария из собственного функционального теста
  - при изменении сценария теста требуется перепрогон функционального теста
  - чем больше набор тестов, тем больше времени требуется на их генерацию

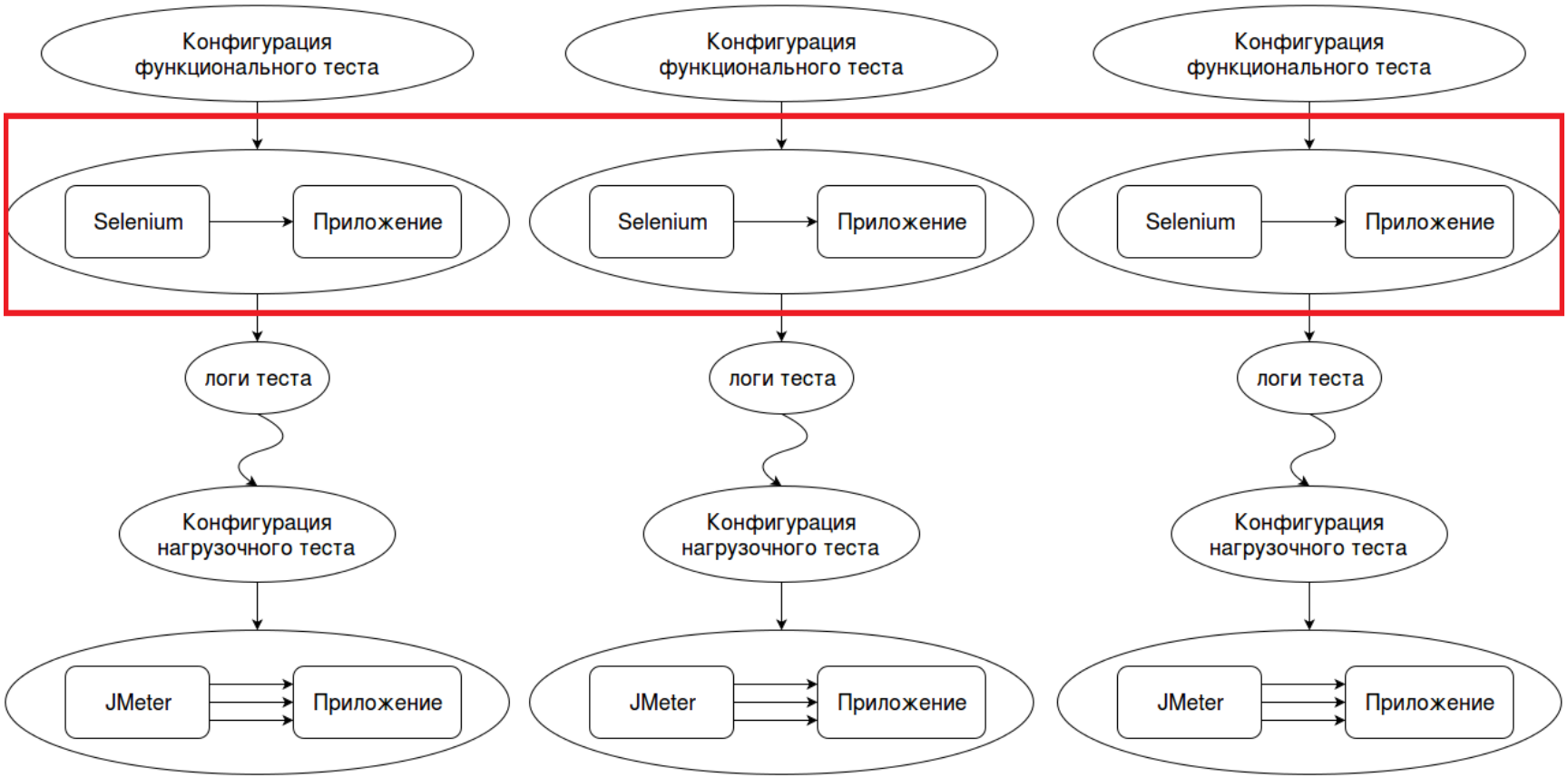
Автоматическая генерация тестов



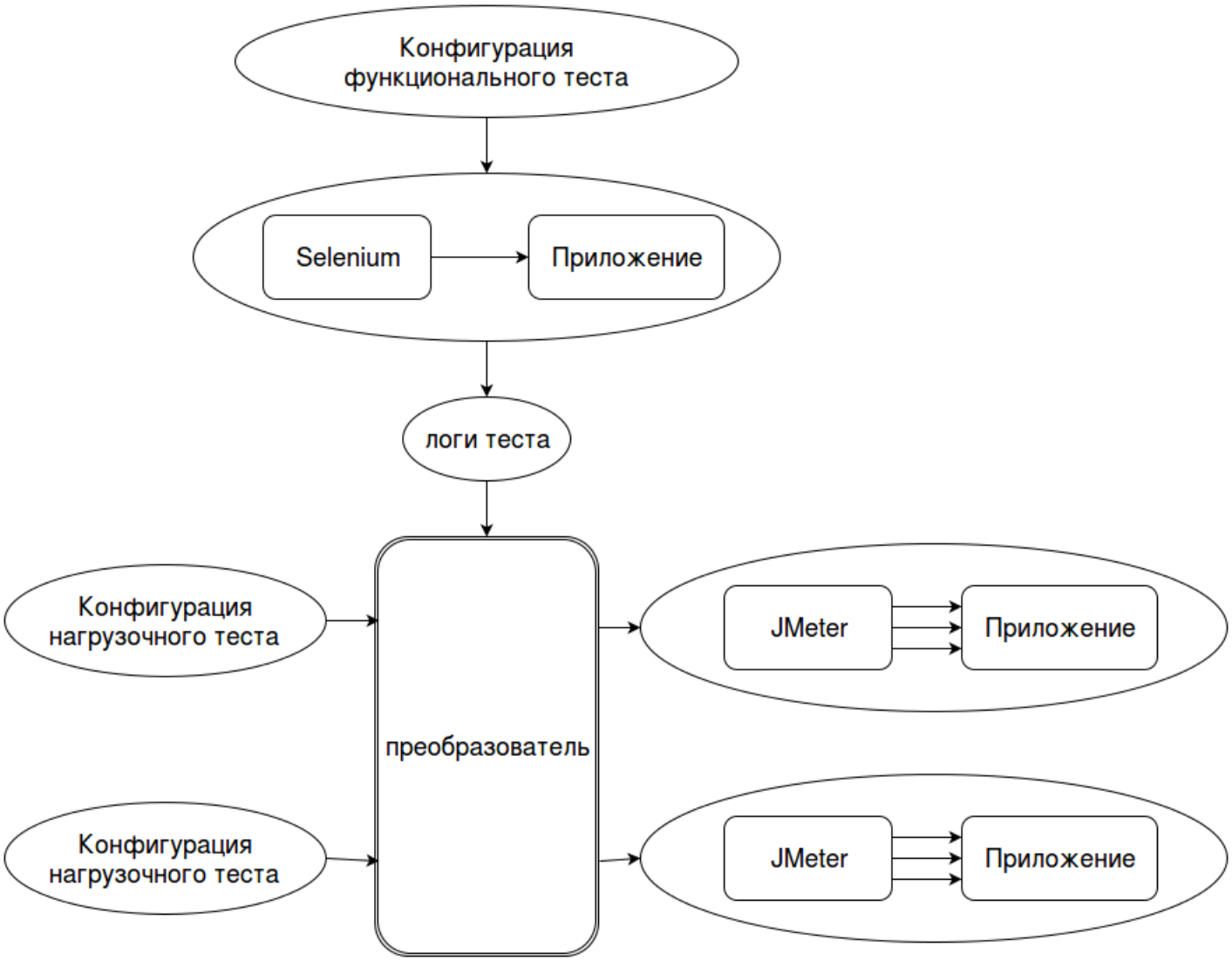
Автоматическая генерация тестов



Автоматическая генерация тестов



Автоматическая генерация тестов



Цель и задачи

- Цель: сократить время и ресурсы, затрачиваемые на исполнение ”генерационных” функциональных тестов
- Задачи
  - предложить подход, в котором для генерации нагрузочных тестов будет использоваться единственный функциональный тест
  - разработать формат функционального теста
  - разработать формат нагрузочного теста
  - предложить и реализовать схему преобразования логов функционального теста в нагрузочные сценарии
  - провести апробацию программного решения

Предложенный подход

- Группировка запросов в логические действия
- Запись активности функционального теста сниффером и детектирование отдельных действий в логах сниффера
- Построение нагрузочного теста перед запуском из каркаса (запрограммированной логики) и записанных действий
- Организация функционального теста:
  - исполняется jmeter’ом
  - каждая логическая группа действий хранится в отдельном скриптовом файле
  - параллельное исполнение групп действий
  - все запросы из одного файла снабжаются уникальным HTTP-заголовком

Формат функционального теста

Формат нагрузочного теста

```
1  - type: login
2    vars_from_file:
3    - file: credentials_${userNum}.tsv
4      varnames:
5      - username
6      - password
7
8  - type: dashboard
9    wait_for: 10m
10   wait_in_background: true
11
12 - type: dashboard_add_issue_widget
13   vars_from_file:
14   - file: search_queries_${userNum}.tsv
15     varnames:
16     - query
17     - type_delay
18   wait_for: 10m
19   wait_in_background: false
```

Детали реализации: параметры запросов

Проблема: некоторые запросы могут передавать параметры со значениями, полученными из предыдущих запросов

-

Решение:

- обнаружение таких параметров с помощью дублирующего прогона действий
- нахождение источника значения с помощью обхода всех ответов сервера в рамках одной группы действий регулярным выражением

Детали реализации: генерация проверок

Проблема: в нагрузочном тесте помимо проверки кодов ответа необходимо делать простейшие проверки ответов сервера

-

Решение:

- обход всех ответов сервера в пределах одной группы действий
- выделение случайных сегментов ответов и их поиск в ответах для данной группы

Детали реализации: фоновая активность

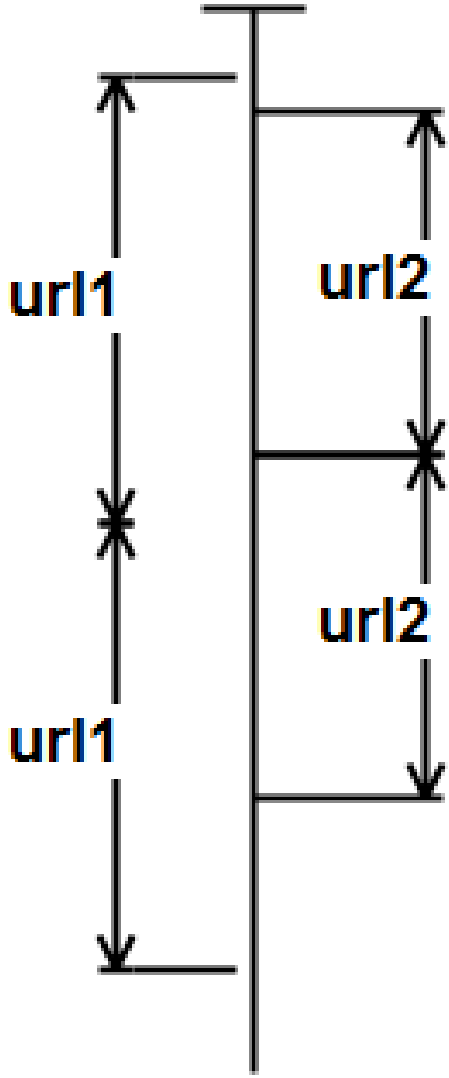
Проблема: многие страницы порождают дополнительную фоновую активность, зачастую периодическую; её необходимо отсеивать от пользовательских действий

-

Решение:

- добавляется длинная пауза по окончании действия
- повторяющийся запрос определяется по постоянному интервалу и URL

Детали реализации: фоновая активность



Формат нагрузочного теста

```
1  - type: login
2    vars_from_file:
3    - file: credentials_${userNum}.tsv
4      varnames:
5      - username
6      - password
7
8  - type: dashboard
9    wait_for: 10m
10   wait_in_background: true
11
12 - type: dashboard_add_issue_widget
13   vars_from_file:
14   - file: search_queries_${userNum}.tsv
15     varnames:
16     - query
17     - type_delay
18   wait_for: 10m
19   wait_in_background: false
```

Генерация итогового нагрузочного теста

- каждое действие преобразовывается в набор размеченных таймстемпами запросов
- все действия записываются в один файл в порядке возрастания таймстемпа
- на старте теста создаётся заведомо достаточное количество потоков
- каждый поток читает следующий запрос из общего файла и ждёт наступления таймстемпа
- перед исполнением в запрос подставляются значения переменных, которые доступны всем потокам

Апробация

- Написание и отладка одного нагрузочного сценария занимает в среднем 6-8 часов, против 8-12 часов при использовании метода с написанием отдельных функциональных тестов
- Объём кода теста уменьшился в 2-3 раза
- Время исполнения функциональных тестов сократилось в 4 раза при не изменившемся времени исполнения нагрузочных тестов

Результаты

- Предложен подход к автоматизации нагрузочного тестирования с относительно низкими накладными расходами на генерацию тестов
- Разработаны форматы нагрузочных и функционального теста
- Предложена и реализована схема преобразования логов функционального теста в нагрузочные сценарии
- Проведена апробация решения