TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Veebirakenduse regressioontestimise automatiseerimine Protractor raamistiku abil

Bakalaureusetöö

Üliõpilane: Vera Bezdušnaja
Üliõpilaskood: 083068IABB
Juhendaja: lektor Karin Rava

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

....

(*kuupäev*) (*allkiri*)

Аннотация

Цель этой бакалаврской диссертации - исследование автоматического веб тестирования на примере использования фреймворка Protractor.

Главная проблема связана с большой затратой времени на регрессионное тестирование.

Автор работы покажет эффективность использования тест фреймворков на примере использования фреймворка Protractor.

Бакалаврской диссертация написана на русском языке и содержит 35 страниц текста, 3 заголовка, 10 картинок и 1 таблицу.

Annotatsioon

Bakalaurusetöö eesmärk on veebirakenduse regressioontestimise automatiseerimise analüüsimine Protractor raamistiku näitel.

Tähtsam probleem on seotud suure ajakaotusega, mis on kulutatud regressioontestimisele.

Töö autor näitab testimise automatiseerimise raamistiku kasutamise efektiivsust kasutades raamistik Protractor näidet.

Lõputöö on kirjutatud vene keeles ning sisaldab teksti 35 leheküljel, 3 peatükki, 10 joonist, 1 tabelit.

Список картинок

Список таблиц

Содержание

1.Вступление

Достаточно сложно, даже порой невозможно представить сегодняшние дни без мобильных телефонов, планшетов, компьютеров, интернета, которые призваны делать нашу повседневную жизнь проще, удобнее, информативнее. Практически любую информацию можно найти за пару минут в интернете. Уже нет необходимости тратить время на дорогу до магазина, пару кликов и ваша покупка в онлайн магазине прошла успешно. Информационные технологии проникли практически во все сферы нашей жизни (медицина, политика, пресса, строительство), это одно из наиболее бурно развивающихся областей современности. Эта область формирует наше сознание, формирует наше будущее, каким будет мир, что будет возможным ещё через пару десятков лет. Информационная эра получила своё начало в конце 1970-х годов после выпуска первых персональных компьютеров и приобрела грандиозный размах в наши дни (“*Смотрите: Annex 1. Статистика интернет пользователей* ”).

Разработка программного обеспечения очень актуальная тема на сегодняшний день, но и не стоит забывать о такой важной составляющей процесса разработки, как тестирование. Никакая разработка не может проходить без проверки качества. Если в некоторых случаях ошибки могут совсем не значительно откликаться для окружающих, то в некоторых случаях ошибки в программном обеспечении (ПО) могут стоить людям жизни.

Как уже было сказано разработка новых приложений, сайтов не обходится без тестирования. Тестирование программного обеспечения – это неотъемлемая часть цикла разработки, которая обеспечивает улучшения качества ПО и отвечает за обнаружение ошибок. Требования современных систем всё-время меняются, их сложность увеличивается. Бывают такие ограничения, как финансирование, время, необходимость определённого технического устройства и т.д. У любого проекта есть временные и финансовые рамки и очень часто эти рамки плохо влияют на качество продукта.

В этой бакалаврской диссертации автор будет исследовать автоматическое веб тестирование на примере использования фреймворка Protractor.

Цель этой работы выявить проблемы, связанные с регрессионным тестированием, предложить путь для их разрешения и проанализировать его.

Работа состоит из трёх частей. В первой части автор рассмотрит текущие проблемы, связанные с веб тестированием. Опишет плюсы и минусы использования фреймворка Protractor. Так же расскажет о критериях выбора программы для написания подобного типа тестов.

Во второй части работы автор напишет руководство по использованию Protractor: как именно необходимо настраивать среду для автоматических тестов и на реальном примере покажет написание реальных тестов. В этой части работы автор произведёт тестирование на примере сайта https://www.elion.ee, так как на работе встал вопрос об оптимизировании процесса тестирования именно этого сайта.

В третьей части работы автор подведёт итоги и покажет достигнутый результат.

1.1 Тестирование

Тестирование - это процесс, которые должен подвердить правльную работу, эффективность, безопасность, а также удобсвто продукта. Тестирование не делается один раз, оно происходит какждый раз после изменения кода. Тестировщики находят не только саму ошибку, но и причину её появления, что позволяет предотвартить такие же ошибки в дальнейшем. От того насколько качественно был протестирован продукт зависит насколько конечные пользователи будут удовлетворены надёжностью, безопасностью и функционалом продукта.

Можно ли без тестирования, зачем оно нужно?

Во-первых, тестирование даёт надежность и безопасность продукта, тестирование должно выявить все проблемы, связанные с доступом к закрытой информации.

Во-вторых, тестирование обеспечивает уверенность в том, что при разработке были реализованы все функциональные требования.

В-третьих, оно гарантирует, что ваш продукт удобен и не содержит функций, которые могут быть не понятны пользователям.

В-четвёртых, данный процесс позволяет убедиться в том, что продукт работает не только как отдельно взятая система, но и вкупе с другими программами.

Как и для разработки, так и для тестирования количество различных программ и фреймворков всё-время растёт, что позволяет сэкономить много времени и помочь при разработке и тестировании ПО.

1.2 Сложность тестирования

Сложность тестирования заключается в том, что любое маленькое изменение может привести к большим последствиям. Прежде, чем начинать любое тестирование всегда нужно составить тест план или политику тестирования. Обслуживание регрессионного тестирования может быть очень сложным заданием, так как какой-то функционал добавляется, какой-то убирается и регрессионный лист также всё время должен изменяться.

Это обычная практика, что когда баг исправлен, то проводятся две формы тестирования. Первая для того, чтобы потвердить, что проблема была действительно решена и вторая для того, чтобы определить, что ничего нового не сломалось. То же самое происходит и при добавлении нового функционала.

Со временем, так как приложение постоянно увеличивается, нужно разработать регрессионный пакет, включающий в себя все тесты для запуска при каждой новой версии приложения.

К сожалению, очень часто многие организации создают большие, сложные приложения без какой-либо документации, что противоречит нормальному тестированию – тестирование должно быть основано на документации, иначе не с чем сравнивать то как приложение работает и как должно работать.

Разработка без документации означает, что со временем в системе может появиться код, который уже не используется, но может приводить к каким-то ошибкам. Любой кто когда-либо был связан с разработкой знает, что изменение чужого кода всегда не лёгкое занятие, так как на первый взгляд может показаться, что код делает одну вещь, а на самом деле из-за других частей кода, результат оказывается совсем другим.

1.3 Возможность автоматизации

Теперь взглянем на это со стороны тестера. Как уже было сказано - даже маленькое изменения может иметь большие последствия. Перед тестером стоит задача, чтобы протестировать систему так, чтобы быть увереным что изменения были правильны и ничего не сломалось. Представьте, что потеницональное изменение в системе должно быть минимально, с одной стороны какждый раз тестировать всю систему будет большой тратой времени, с другой стороны если не

будет протестировано всё, то есть шанс ошибки. Тут на помощь и приходит автоматическое регрессионное тестирование: проверяется весь базовый функционал и за короткое время.

Чтобы понять, что такое регрессионное тестирование, лучше всего сперва понять, что такое регрессионные ошибки. Это такие изменения в коде, при которых перестаёт работать, то, что уже работало до этого и было протестировано, т.е. по сути любое изменение может повлиять на работу любой части системы. “Регрессионное тестиование проводится при любом изменении, которое может затронуть любой работающий до этого функционал программного обеспечения.”[1] Очень сложно предсказать последствия любого изменения, поэтому по сути после каждого изменения системы нужно проводить регрессионное тестирование всей системы. Если система очень большая, то ручное тестирование всего продукта может занимать часы или дни, поэтому так важно автоматическое тестирование, которое позволяет экономить много времени.

Вот так выглядит обычный тест план с результатами:

Таблица 1: Тест план. Часть регрессионного листа содержащего результаты тестов в двух версиях

		Total Test	**16**	**16**	
		Total Pass	15	14	
		Total Pass Percentage	93.75%	87.50%	
		Total Fail	6.25%	12.50%	
Test Name	Target name	OS Platform	0.83.2	0.84.0	History
Изменение языка контента	WEB3-1023	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности кнопки "TV"	WEB3-1024	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности кнопки "Internet"	WEB3-1025	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности кнопки "Internet"	WEB3-1026	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности кнопки "Telefon"	WEB3-1027	Windows 7	Fail	pass	Mixed

Проверка корректной функциональности кнопки "Pakkumised"	WEB3-1028	Windows 7	Pass	pass	Mixed
Проверка корректной функциональности кнопки "Järelmaks"	WEB3-1029	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности кнопки "Iseteenindus"	WEB3-1030	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности компонента "Pea bänner"	WEB3-1031	Windows 7	Pass	Fail	Mixed
Проверка корректной функциональности компонента "minuTV-parim TV app"	WEB3-1032	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности компонента "Onlain Abi"	WEB3-1033	Windows 7	Pass	pass	All Passed
Проверка корректной функциональности компонента "Elioni Teenus äriklientidele"	WEB3-1034	Windows 7	Pass	pass	All Passed

1.4 Критерии выбора фреймворка для тестирования

На данный момент существует более 10 фреймворков, которые стоит рассматривать, как инструмент для написания тестов. Часть фреймворков подходят больше для юнит тестирования, часть для регрессионого. На что стоит обратить внимание при выборе фреймворка?

- Синтаксис. От этого будет зависеть насколько легко и интуитивно вы сможете писать тесты на этом фреймворке. Не стоит выбирать вреймворк если его синтаксис кажется вам странным и сложным для использования. Например, qUnit[2] больше декларативный фреймворк. Его API состоит из таких функций как test, equals, strictEqual и т.д. Другой тест фреймворк, mocha[3], больше тестирует по тексту, чем по структуре. Его API состоит из таких функций как describe, it, assert.
- Функционал. Очень часто при тестировании вы оказываетесь в ситуации где вам нужные какие-то специальные условия и хорошо если бы фреймворк мог их предоставить. Например, fake сервер – если ваш код отсылает AJAX запросы на сервер и вы хотите fake response от сервера, то такой функционал

необходим в вашем фремворке. Fake часы – если ваш код ведёт себя по-разному в зависимости от времени суток, то fake часы это как раз то чт овам нужно. Очень хороший фреймворк, который поддерживает большое количество вспомогательных инструментов – Sinon.JS.[4]

- Хорошая документация и большая аудитория. Чем подробнее написана документация, тем легче будет решать проблемы, возникающие при написании тестов. Большое количество пользователей показывает, что фреймворк действительно стоит того, чтобы ознакомится с ним и то что с большей вероятностью кто-то ответит на ваш вопрос по фреймворку.

Стоит ли пробывать каждый фреймворк?

Нет, достаточно будет просто рассмотреть примеры на сайте, это уже даст вам представление о фремворке.

Ниже автор расскажет о наиболее популярных фреймворков для тестирования: Protractor, Karma, TestSwarm.

Protractor – разработан командой AngularJS. [5]

Преимущества:

- Использует node.js, так что можно запускать под Windows/OS X/Linux
- Позволяет запускать все виды бразуеров
- Одновременно поддерживает запуск многих клиентов
- Большое количество опций для запуска
- Есть возможность запсука тестов с командрой строки, а значит лёгкая интеграция для автоматического запуска.
- Создаёт репорт по окончанию.
- Поддерживает Jasmine, mocha, Cucumber.

Недостатки:

- Сложность отладки

Архитектура фреймворка Protractor:

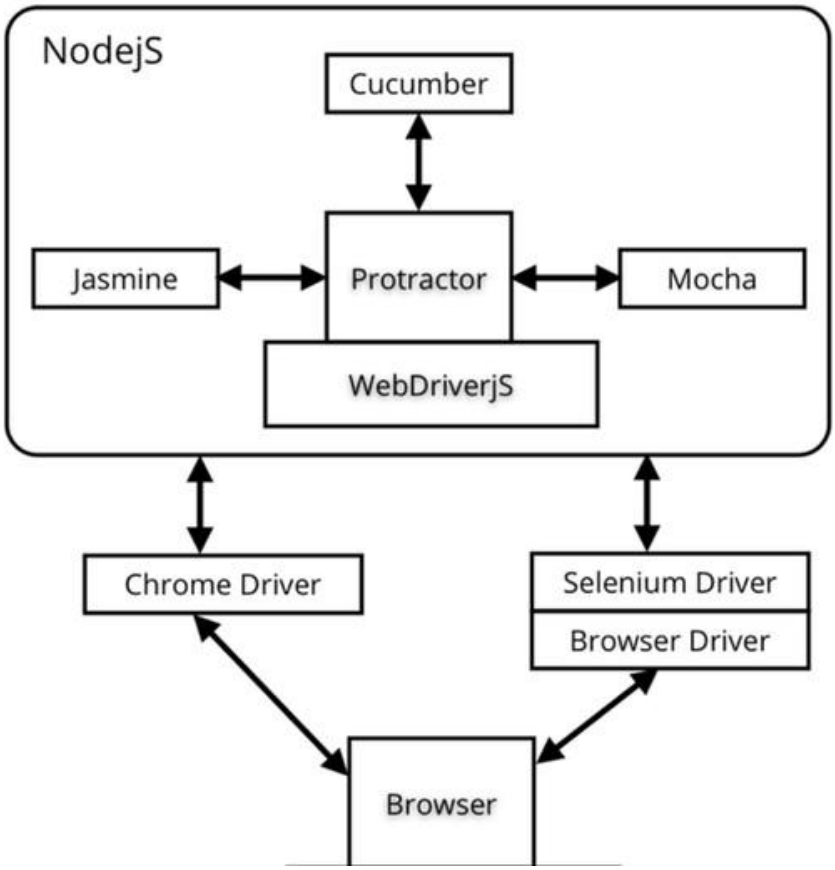


Рисунок 1: Protractor. Архитектура Protractor.
[Источник: http://www.thoughtworks.com//insights/blog/testing-angularjs-apps-protractor]

Karma – разработан той же командой, что и Protractor. Больше используется для юнит тестирования. [6]

Преимущества:

- Использует node.js, так что можно запускать под Windows/OS X/Linux
- Позволяет запускать все виды бразуеров
- Одновременно поддерживает запуск многих клиентов
- Большое количество опций для запуска
- Есть возможность запсука тестов с командрой строка, а значит лёгкая интеграция для автоматического запуска.
- Создаёт репорт по окончанию.
- Есть плагин для многих IDE.

Недостатки:

- Не поддердживает Node.JS тестирование

- Нет истории предыдущих тест результатов

TestSwarm – разработан командой jQuery. Подходит для любого типа веб страниц. [7]

Преимущества:

- Использует интеграционный сервер на JavaScript
- Поддерживает большое количество бразуеров и операциооных систем
- Одновременно поддерживает запуск многих клиентов
- Автоматический перезапуск тестов при изменении.
- Показывает историю тестирования
- Показывает рультат группирую по операционной сиситеме и версии бразуера

Недостатки:

- Нельзя запустить с помощью ant
- Нет плагина для IDE

Это не полный список фреймворков для написания тестов, но достаточный для ознакомления с различными возможностями тестирования.
Автор остановился на фремфорке Protractor, так как уже был знаком с написанием тестов на Javascript и часть сайтов, которые нуждались в тестировании были написаны на AngularJS, а Protractor лучше всего подходит для этих тестов.

2. Автоматическое регрессионное тестирование на примере фреймворка Protractor

В этой части работы автор напишет руководство по использованию Protractor: как именно необходимо настраивать среду для автоматических тестов и на реальном примере покажет написание реальных тестов.

2.1. Установка фреймворка для написания автоматических тестов

Любая программа или фреймворк нуждаются в установке и конфигурации. При выборе фреймворка очень важно предварительно узнать на каких устройствах и операционных системах возможно его запустить, чтобы в дальнейшем не возникло проблем с переносом тестирования на другое устройство. Лучше всего выбирать фреймворки, которые запускаются на всех популярых операционных системах, т.е. на Windows, Mac OS, Linux. Автор будет рассказать про установку на операционную систему Windows 8.1.

2.1.1. Предварительная установка

Прежде чем устанавливать сам фреймворк нужно установить все программы, которые нужны для его запуска.

Так как Protractor является Node.JS программой, то необходимо установка Node.JS.

Скачать её можно отсюда:

https://nodejs.org/download/

После установки удостоверьтесь, что версия Node.JS выше, чем 0.10.0, с помощью команды в командной строке:

```
node --version
```

Рисунок 2: Проверка Node.JS версии

Для запуска и управления браузерами в режиме тестирования, нам понадобится Selenium Server, который работает на Java. Установите её отсюда:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Эта команда покажет версию, установленной Java, если установка прошла успешно.

```
java -version
```

2.1.2. Установка Protractor

Когда все предварительные программы установлены, можно начинать установку фреймворка.
Для начала создайте папку где вы собираетесь писать тесты (например, C:\protractor) и перейдите в неё с помощью командной строки.
Для установки Protractor используйте следующую команду:

```
npm install -g protractor
```

Эта команда установит две программы: protractor и webdriver-manager. Для проверки, что всё установлено правильно запустите

```
protractor --version
```

webdriver-manager – программа для работы с Selenium Server. Для скачивания необходимых компонентов запустите

```
webdriver-manager update --ie
```

Параметр –ie нужен для установки драйвера для Internet Explorer. По умолчанию webdriver-manager скачивает только дравйвера для Chrome и Firefox. Дравйвера для других браузеров также доступны. [9]

После установки запустите сам сервер:

```
webdriver-manager start
```

Ваш тест скрипт будет посылать запрос на этот сервер для запуска и контролирования локальных браузеров. Оставьте эту командную строку открытой для обзора логов.
Информацию касательно статуса сервера можете посмотреть тут:

http://localhost:4444/wd/hub/static/resource/hub.html

Protractor поддерживает тестирование всех популярных браузеров: Chrome, Firefox, Internet Explorer, а также мобильных браузеров Android и iOS.

2.1.3. Конфигурация Protractor

Создайте файл conf.js и положите его в папку где будут храниться ваши тесты, в моём примере это будет C:\protractor.

Содержимое файла должно быть следующим:

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['test.js'],
  suites: {
    homepage: 'tests/homepage/*_spec.js'
  },
  multiCapabilities: [{
    browserName: 'firefox'
  }, {
    browserName: 'chrome'
  }, {
    browserName: 'internet explorer'
  }]
};
```

seleniumAddress – URL для Selenium Server

suites – путь для тестов. Возможен как запуск всех suites, так и одной.

multiCapabilities – список браузеров на которых запускать тестирование

2.1.4. Синтаксис тестов

Для написания тестов автор использует фреймворк Jasmine 2.0.

Вообще Protractor поддерживает 4 фреймворка Jasmine 1.3, Jasmine 2.0, Mocha, and Cucumber. Был выбран Jasmine 2.0, так как на нём уже был опыт работы.

Почитать про синтаксис можно тут:

http://jasmine.github.io/2.0/introduction.html

Для написания первого теста создадим файл first_spec.js в папке tests/homepage:

```
describe('Elion homepage', function() {

  it('should have a title', function() {

    browser.get('http://elion.ee');

    expect(browser.getTitle()).toEqual('Eraklient - Elion');

  });

});
```

Тест готов к запуску:

```
protractor conf.js --suite homepage
```

При правильной настроке вы увидите, что тест на всех трёх браузерах прошёл успешно:

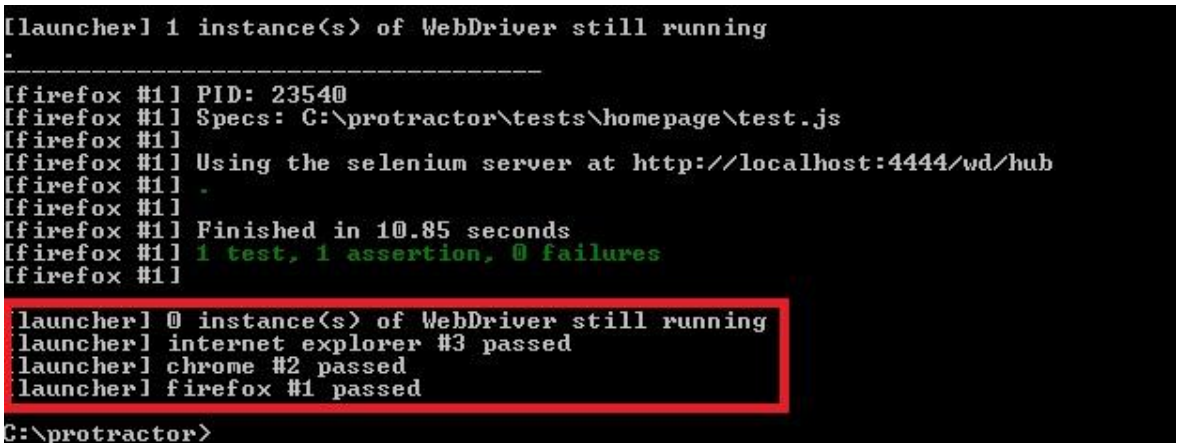


Рисунок 3: Вывод теста. Результат проверки на наличие заголовка страницы.

Очень часто при тестировании веб приложений изменения происходят несколько раз за день, а значит регрессионное тестирование должно проходить очень быстро. На реальном рабочем примере будет показано как за несколько минут можно пройти регрессионное тестирование на трёх самых популяных бразуерах: Google Chrome, Firefox и Internet Explorer. (“ *Смотрите: Аппех 2. Топ 5 популярных браузеров* ”)

2.2. Написание тестов

В данной части работы автор произведёт тестирование на примере сайта https://www.elion.ee, так как на работе встал вопрос об оптимизировании процесса тестирования именно этого сайта. Ручное регресионное тестирование занимало до трёх дней, поэтому и была поставлена задача проанализировать и выбрать для автоматического тестирования фреймворк, позволяющий значительно сократить время тестирования.

Тесты функциональные, в реальном проекте их было создано более 100, в данной работе автор покажет лишь их малую часть:

- Изменение языка контента
- Переход на страницу
- Проверка главного меню на содержание
- Проверка работы поиска
- Проверка перехода на главную страницу с любой страницы

2.2.1. Изменение языка контента

Тестируемая система:

Веб браузер Google Chrome, Firefox, Internet Explorer

Название:

Изменение языка контента

Шаги:

1. Зайти на страницу www.elion.ee
2. Поменять язык
3. Убедиться что язык контента поменялся

Тест:

```
describe('Elion homepage', function() {

  it('should change language to russian', function() {

    browser.get('http://elion.ee');

    element(by.css('li.languages a')).click();

    expect(element(by.id('private')).getText()).toEqual('Домой');

  });

});
```

Результат:

Все 3 бразуера прошли проверку.

```
[launcher] Running 3 instances of WebDriver
.
-------------------------------------
[chrome #2] PID: 29948
[chrome #2] Specs: C:\protractor\test.js
[chrome #2]
[chrome #2] Using the selenium server at http://localhost:4444/wd/hub
[chrome #2] Elion homepage should change language to russian
[chrome #2] .
[chrome #2]
[chrome #2] Finished in 4.171 seconds
[chrome #2] 1 test, 1 assertion, 0 failures
[chrome #2]

[launcher] 2 instance(s) of WebDriver still running
.
-------------------------------------
[internet explorer #3] PID: 33352
[internet explorer #3] Specs: C:\protractor\test.js
[internet explorer #3]
[internet explorer #3] Using the selenium server at http://localhost:4444/wd/hub

[internet explorer #3] Elion homepage should change language to russian
[internet explorer #3] .
[internet explorer #3]
[internet explorer #3] Finished in 4.462 seconds
[internet explorer #3] 1 test, 1 assertion, 0 failures
[internet explorer #3]

[launcher] 1 instance(s) of WebDriver still running
.
-------------------------------------
[firefox #1] PID: 33320
[firefox #1] Specs: C:\protractor\test.js
[firefox #1]
[firefox #1] Using the selenium server at http://localhost:4444/wd/hub
[firefox #1] Elion homepage should change language to russian
[firefox #1] .
[firefox #1]
[firefox #1] Finished in 10.472 seconds
[firefox #1] 1 test, 1 assertion, 0 failures
[firefox #1]

[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #2 passed
[launcher] internet explorer #3 passed
[launcher] firefox #1 passed
```

Рисунок 4: Вывод теста. Результат проверки на изменение языка контента

2.2.1.1 Изменение языка контента – тест на провал

Для этого теста мы специально изменим текст на неправильный, чтобы проверить что браузеры действительно работают.

Тестируемая система:

Веб браузер Google Chrome, Firefox, Internet Explorer

Название:

Изменение языка контента – тест на провал

Шаги:

1. Зайти на страницу www.elion.ee
2. Поменять язык
3. Подставить не правильное значение для проверки языка контента

Тест:

NB! Текст был изменён с “Домой” на “Домо”

```
describe('Elion homepage', function() {
  it('should change language to russian', function() {
    browser.get('http://elion.ee');
    element(by.css('li.languages a')).click();
    expect(element(by.id('private')).getText()).toEqual('Домо');
  });
});
```

Результат:

Как и ожидалось тест выдал ошибку:

Expected “Домой” to equal “Домо”.

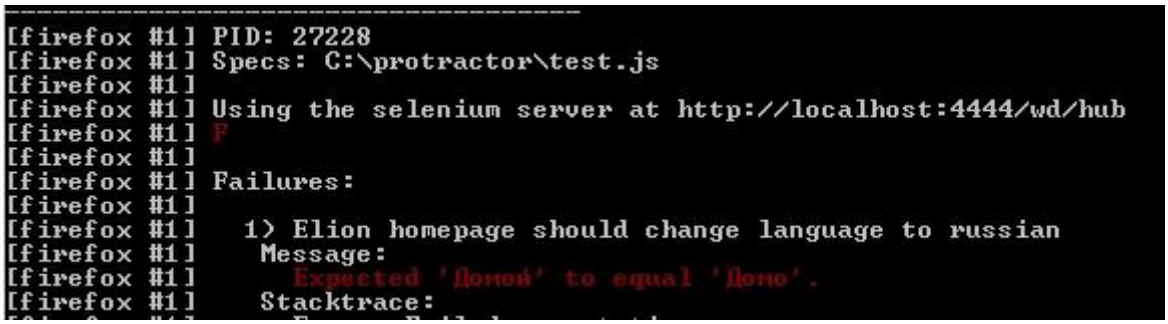


Рисунок 5: Вывод теста. Вывод ошибки при проверке на изменение языка

2.2.2. Переход на страницу

Тестируемая система:

Веб браузер Google Chrome, Firefox, Internet Explorer

Название:

Переход на страницу

Шаги:

1. Зайти на страницу www.elion.ee
2. Нажать на “TV”
3. Нажать на “nutiTV teenused”
4. Убедиться что открылась правильная страница

Тест:

```
describe('Elion homepage', function() {

  it('should switch to nutiv page', function() {

    browser.get('http://elion.ee');

    element(by.id('eraklient_nutitv')).click();

    element(by.id('eraklient_nutitv_nutitv-teenused')).click();

    expect(element(by.id('element-hero')).isPresent()).toBe(true);  });

});
```

Результат:

Только Google Chrome прошёл тест, остальные бразуеры тест не прошли.

```
[launcher] chrome #2 passed
[launcher] internet explorer #3 failed 1 test(s)
[launcher] firefox #1 failed 1 test(s)
[launcher] overall: 2 failed spec(s)
[launcher] Process exited with error code 1
```

Рисунок 6: Вывод теста. Результат проверки при переходе на страницу

Рассмотрим вывод ошибки более детально:

```
[firefox #1] Using the selenium server at http://localhost:4444/wd/hub
[firefox #1] Elion homepage should switch to nutiv page
[firefox #1] WARNING - more than one element found for locator By.id("eraklient_
nutitv_nutitv-teenused") - the first result will be used
[firefox #1] F
```

Рисунок 7: Вывод теста. Ошибка при запуске теста.

Protractor утверждает, что на странице находятся два элемента с одним id: *eraklient_nutitv_nutitv-teenused*

Проверим это используя Google Chrome Developer Tools:

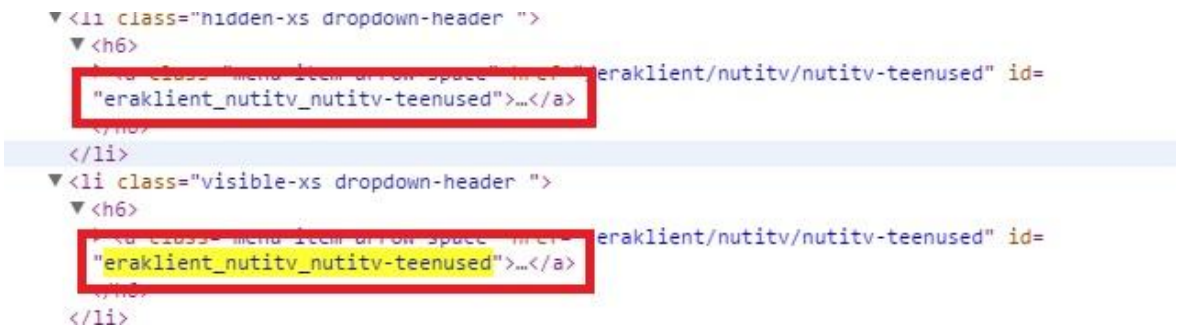


Рисунок 8: Chrome Developer tools. Нахождений одиаковых id.

Действительно на странице два элемента с одним id, что по веб стандартам не допустимо. [8]

2.2.3. Проверка главного меню на содержание

Тестируемая система:

Веб браузер Google Chrome, Firefox, Internet Explorer

Название:

Проверка главного меню на содержание

Шаги:

1. Зайти на страницу www.elion.ee

2. Проверить что все элементы меню присутствуют на странице

Тест:

```
describe('Elion homepage', function() {

  it('should have all header menus', function() {

   browser.get('http://elion.ee');

   expect(element(by.id('eraklient_nutitv')).getText()).toEqual('TV');

   expect(element(by.id('eraklient_internet')).getText()).toEqual('Internet');

   expect(element(by.id('eraklient_telefon')).getText()).toEqual('Telefon');

   expect(element(by.id('eraklient_kodujuhtimine')).getText()).toEqual('Kodujuhtimine');

   expect(element(by.id('eraklient_pakkumised')).getText()).toEqual('Pakkumised');

   expect(element(by.id('eraklient_jarelmaks')).getText()).toEqual('Järelmaks');

   expect(element(by.id('private_self-service')).getText()).toEqual('Iseteenindus');  });

});
```

Результат:

Все 3 бразуера прошли проверку.

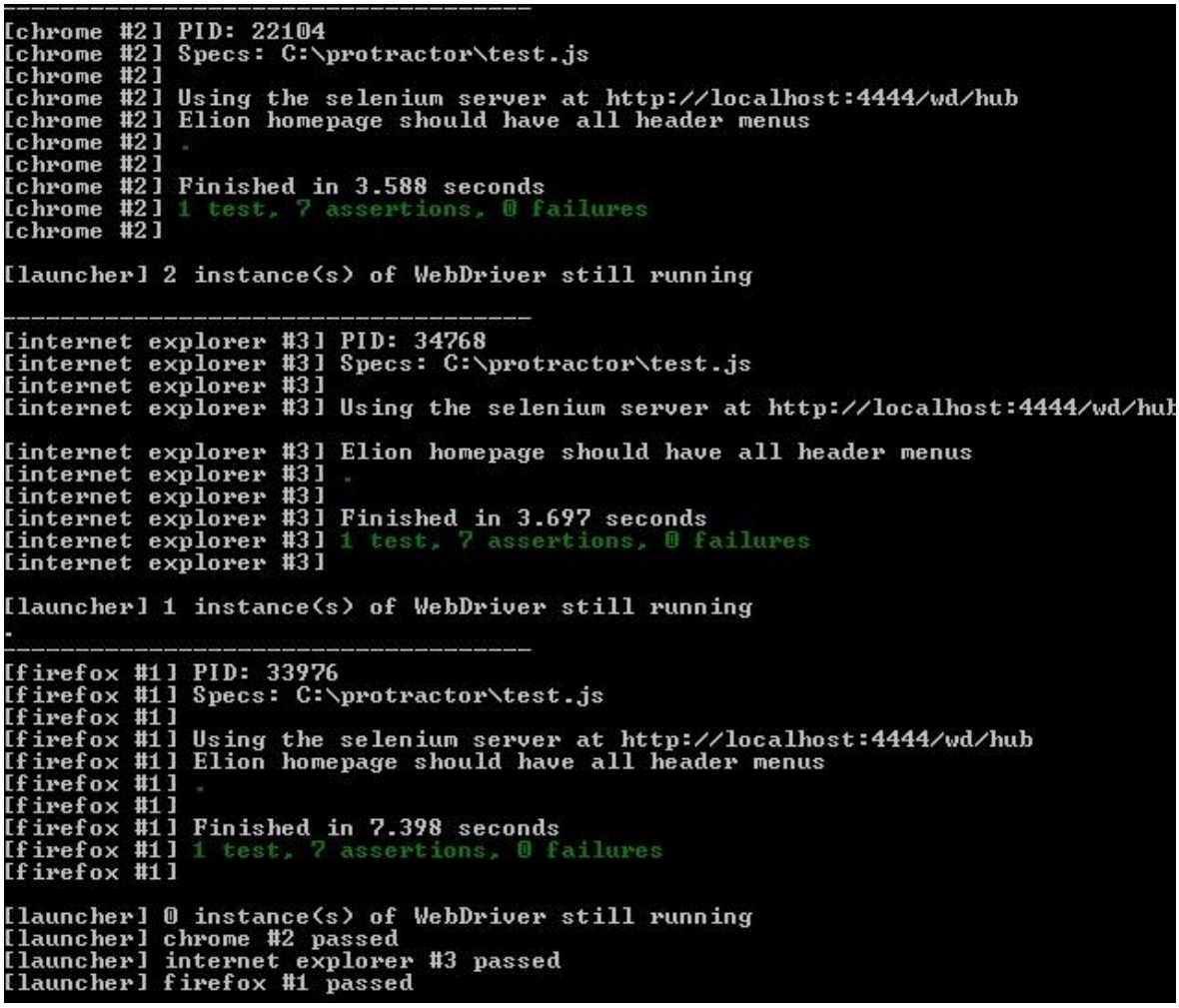


Рисунок 9: Вывод теста. Результат проверки на наличие главного меню.

2.2.4. Проверка работы поиска

Тестируемая система:

Веб браузер Google Chrome, Firefox, Internet Explorer

Название:

Проверка работы поиска

Шаги:

1. Зайти на страницу www.elion.ee
2. Включить поиск
3. Найти первое совпадение в поиске по слову “Telefon”
4. Зайти на страницу по первому совпадению в поиске и удостовериться, что это главная страница об услугах телефона

Тест:

```
describe('Elion homepage', function() {

  it('should show search results', function() {

   browser.get('http://elion.ee');

   element.all(by.css('.navbar-collapse
ul')).get(1).all(by.tagName('li')).get(1).element(by.tagName('a')).click();

   element(by.css('form.search input.query')).sendKeys('Telefon');

   element(by.css('form.search li')).get(0).click();

   expect(browser.getTitle()).toEqual('Telefon - Elion');

});
```

Результат:

Все 3 бразуера прошли проверку.

2.2.5. Проверка перехода на главную страницу с любой страницы

Тестируемая система:

Веб браузер Google Chrome, Firefox, Internet Explorer

Название:

Проверка перехода на главную страницу с любой страницы

Шаги:

1. Зайти на страницу www.elion.ee
2. Перейти на любуюу страницу
3. При клике на лого вернуться на главную и проверить что эта страница действительно главная
4. Проделать с 2-3 разными страницами

Тест:

```
describe('Elion homepage', function() {

  it('should switch between pages', function() {
```

```
    browser.get('http://elion.ee');

    expect(browser.getTitle()).toEqual('Eraklient - Elion');

   element.all(by.css('.mainfooter-bottom
ul')).get(1).all(by.tagName('li')).get(0).element(by.css('a')).click();

   expect(browser.getTitle()).toEqual('Uudised ja artiklid - Elion');

   element(by.id('private')).click();

   expect(browser.getTitle()).toEqual('Eraklient - Elion');

});

describe('Elion homepage', function() {

  it('should switch between pages', function() {

    browser.get('http://elion.ee');

    expect(browser.getTitle()).toEqual('Eraklient - Elion');

          element.all(by.css('.mainfooter-bottom
ul')).get(0).all(by.tagName('li')).get(2).element(by.css('a')).click();

          expect(browser.getTitle()).toEqual('Lepingud ja tingimused - Elion');

          element(by.id('private')).click();

          expect(browser.getTitle()).toEqual('Eraklient - Elion');

});

describe('Elion homepage', function() {

  it('should switch between pages', function() {

    browser.get('http://elion.ee');

    expect(browser.getTitle()).toEqual('Eraklient - Elion');

          element.all(by.css('.mainfooter-bottom
ul')).get(2).all(by.tagName('li')).get(2).element(by.css('a')).click();

          expect(browser.getTitle()).toEqual('Projektide kooskõlastamine - Elion');

          element(by.id('private')).click();

          expect(browser.getTitle()).toEqual('Eraklient - Elion');

});
```

Результат:

Все 3 бразуера прошли проверку.

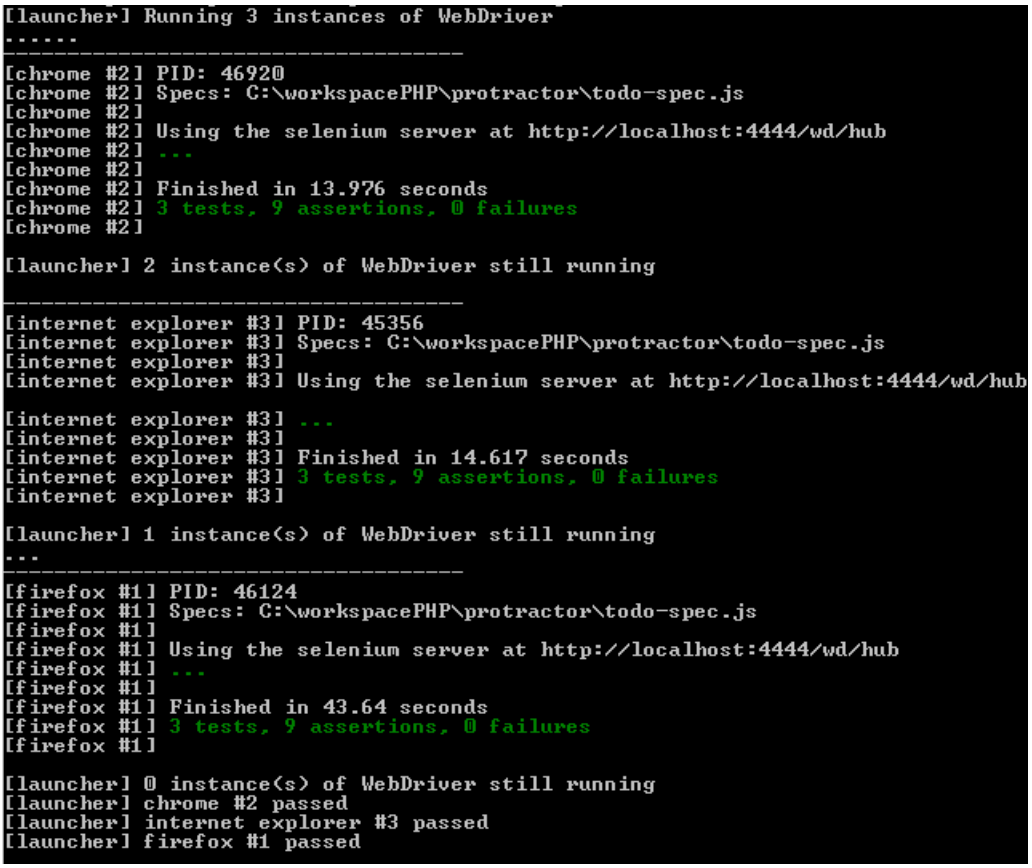


Рисунок 10: Вывод теста. Результат проверки на переход на главну страницу с любой другой.

3. Заключение

Написание тестов, конечно, не быстрый процесс, так как, часто веб-страницы сделаны не по веб-стандартам и на поиск решения в интеренете уходит время, но обычно всегда можно быстро найти быстрое решение и в конечном итоге вы получите регрессионный лист, который за пару минут проверить функционал веб-страницы.

Регрессионное тестирование и создание регрессионного листа больше нужно для больших, постоянно развивающихся проектов. Для маленьких проектов и проектов, которые не планируются к дальнейшей доработки можно обойтись и обычным тестированием функционала. Можно настроить систему так, что при каждом обновлении сайта тесты запускаются автоматически и система информирует если какой-то из тестов провалился.

Как и было сказано в начале, автор выявил проблемы, связанные с веб тестированием, предложил путь для их разрешения и показал решение проблемы на примере использования фреймворка Protractor.

Проведённый анализ дал мне возможность предложить данное решение в своей фирме, чтобы оптимизировать процесс тестирования.

Основываясь на моём анализе, хочу обозначить преимущества использования Protractor:

- можно тестировать различные браузеры
- можно тестировать браузеры под разным разрешением
- тестирование автоматическое

Есть и стороны требующие улучшения:

- по окончанию должен генерироваться репорт

Нельзя назвать это метод „серебряной пулей“ от всех проблем, связанных с регрессионным тестированием, но так или иначе этот метод позволит вам быстро и эффективно протестировать веб приложение.

Веб разработка, как и веб тестирование имеет одну отличительную особенность – она всегда нуждается в непрерывном обучении, технологии развиваются слишком быстро, даже если вы нашли подходящий вам метод, не переставайте интересоваться новинками – может оказаться, что новый метод будет для вас ещё более полезен.

Kokkuvõte

Tarkvara testimine – see on tarkvara arendamistsüklist üks tähtsam osa, mis lubab parandada tarkvara kvaliteeti ja vastutab vigade ülesotsimise eest. Kaasaegsete süsteemide vajadused muutuvad päevast päeva, kasvab nende keerukus. On sellised piirangud nagu finantseerimine, aeg, teatud tehnilise seadme vajadus jms. Igal projektil on aja ja finanseerimispiirangud ja need piirangud sageli mõjutavad tulemuse kvaliteeti.

Bakalaurusetöös autor uurib veebirakenduse automatiseeritud regressioontestimist kasutades raamistik Protractor näidet.

Selle töö eesmärk on avaldada probleeme, mis on seotud regressioontestimisega, seda analüüsida ja pakkuda lahendusvariandid.

Esimeses osas autor vaatleb olemasolevaid probleeme, mis on veebirakenduse testimisega seotud. Kirjeldab positiivsed ja negatiivsed küljed Protractor raamistiku kasutamisel. Annab ülevaate programmi valikukriteeriumitest sellist tüüpi testide kirjutamisel.

Teises töö osas autor kirjutab Protractor kasutusjuhendit: nimelt, kuidas seadistada automaattestimise keskkonda ja näitab reaalsuses, kuidas neid teste kirjutada. Sellel töö osas autor teeb testimist kasutades https://www.elion.ee veebilehte, kuna selle veebilehega tekkis testimise optimeerimisprotsessi probleem.

Kolmandas osas teeb autor järelduse ja näitab saavutatud tulemust.

Testide kirjutamine ei ole kiire protsess, kuna sageli veebilehed ei ole tehtud veebistandardite järgi, lahenduse otsimine interenetist nõuab aga aega, kuigi seda lahendust on võimalik leida.

Regressioontestimine ning regressioonilehe loomine sobib rohkem suurte ja pidevalt arendavate tarkvarasüsteemide jaoks. Väikeste tarkvarasüsteemide jaoks, mis ei nõua edasist töötlemist, piisab tavalisest funktsionaalsest testimisest.

Tehtud analüüs võimaldas mul pakkuda antud lahendust minu tööandjale, selleks et optimeerida testimise protsessi.

Võttes minu analüüsi põhjaks, tahan loetleda Protractori kasutamise eelised:

- Võimalik erinevate brauserite testimine
- Võimalik brauserite testimine erinevate lahendustega.

- Automaatne testimine

On ka need küljed mis, võiks olla parandatud:

- Testimise lõppfaasis peaks olema raport genereeritud

Ei saa öelda, et Protractor raamistik on sobilik kõikide probleemide lahendamisel, aga see meetod kindlasti lubab kiiresti ja efektiivselt testida veebirakendusi.

Veebirakenduse arendamine ja veebirakenduse testimine on olemas üks eripärane joon - see nõuab pidevat koolitust, tehnoloogiad arenevad väga kiiresti ja isegi kui on olemas sobilik meetod, ei tasu peatuda uute asjade uurimisel – äkki osutub uus meetod veelgi sobilikumaks.

Kõik planeeritud tööd oli tehtud ja püstitatud eesmärgid oli täidetud.

Используемая литература

Книги:

[1] Elfriede Dustin. Automated Testing: Selected Best Practices, 2002.

Интернет:

[2] QUnit: A JavaScript Unit Testing framework [WWW]

http://qunitjs.com/ (07.05.2015)

[3] Mocha JavaScript test framework [WWW]

http://mochajs.org/ (07.05.2015)

[4] Sinon.JS [WWW]

http://sinonjs.org/ (07.05.2015)

[5] Protractor [WWW]
https://github.com/angular/protractor (01.05.2015)

[6] Karma [WWW]
http://karma-runner.github.io/0.12/index.html (12.05.2015)

[7] TestSwarm [WWW]
http://swarm.jquery.org/ (12.05.2015)

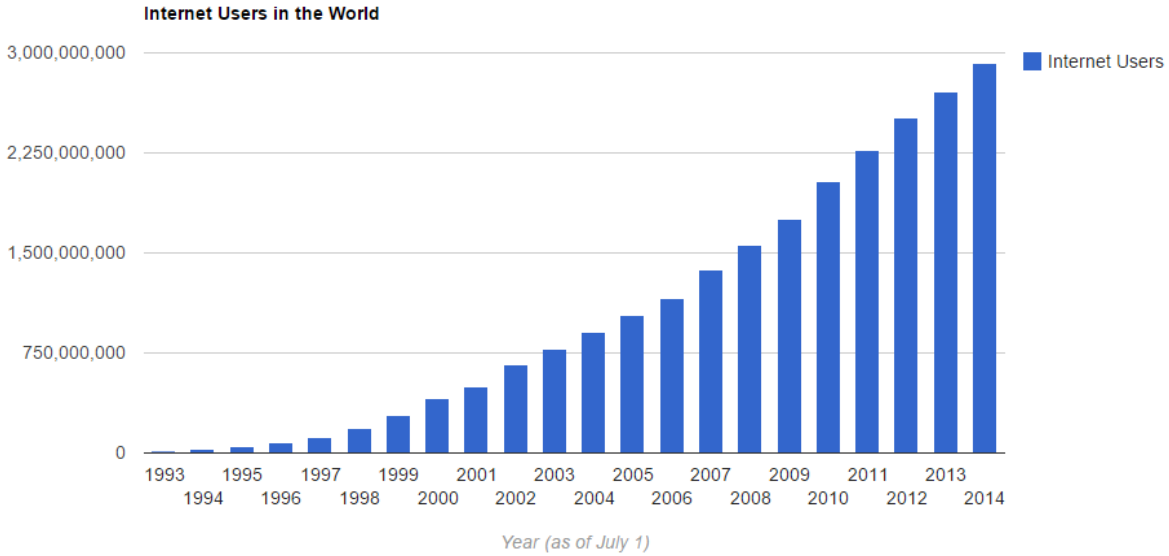
[8] The global structure of an HTML document [WWW]

http://www.w3.org/TR/html401/struct/global.html#h-7.5.2 (28.04.2015)

[9] Selenium WebDriver [WWW]

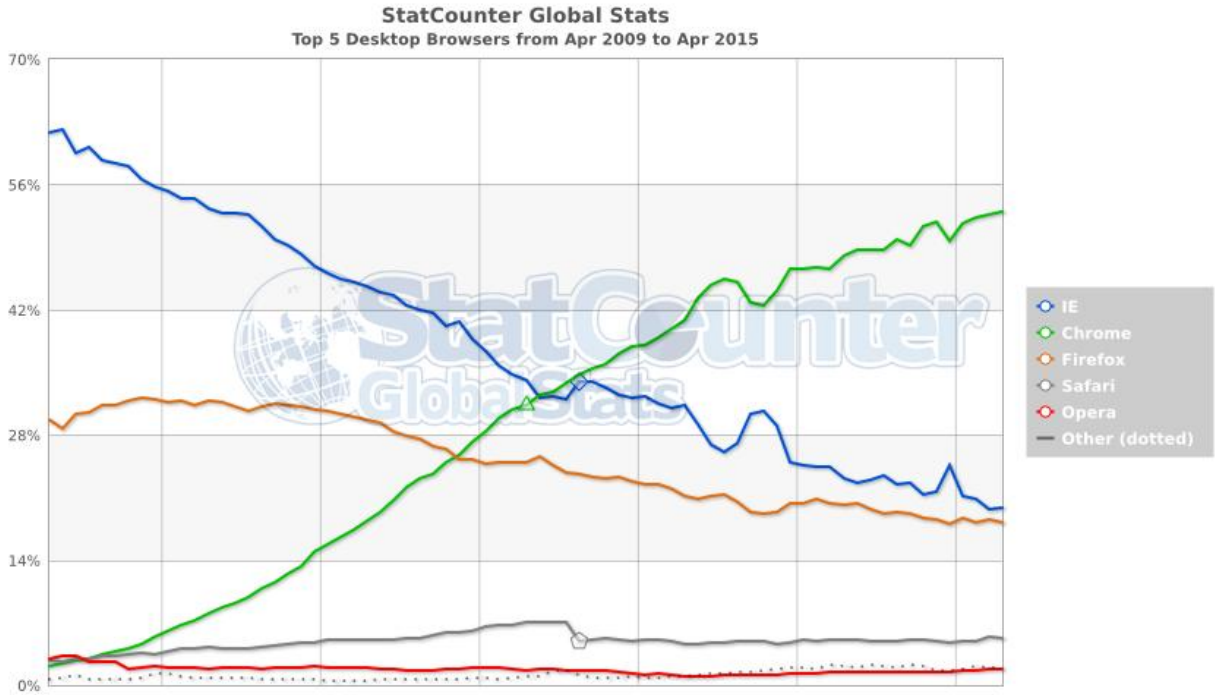
http://docs.seleniumhq.org/projects/webdriver/ (03.05.2015)

ANNEX 1. Статистика интернет пользователей



Источник: http://www.internetlivestats.com/internet-users/#trend (03.05.2015)

ANNEX 2. Топ 5 популярных браузеров



Источник: http://gs.statcounter.com/#desktop-browser-ww-monthly-200904-201504 (03.05.2015)