SQGNE Software Quality Group of New England

Dedicated to Software Quality Professionals

www.sqgne.org

WELCOME TO SEASON 23!

SQGNE is made possible by the support of our sponsors:

Oracle and
Sun Integrated Systems

CHECKPOINT Technologies

casenet

ASTQB American Software Testing Qualifications Board, Inc.

FLATBREAD COMPANY

WELCOME TO SEASON 23!

- All-volunteer **non-profit** with no membership dues!
- Supported entirely by our sponsors…
- Over **1,384** members on LinkedIn, **945** Constant Contact
- **196** members have joined our Meetup Group
- Monthly meetings - Sept to June on 2nd Wed of month
- **SQGNE Web site: www.sqgne.org**

LinkedIn SQGNE Software Quality Group of New England

Officers / Hosts / Mission

Current Officers:

- John Pustaver – Founder
- Stan Wrobel– President
- Robin Goldsmith – Vice President
- Barbara Wioncek – Treasurer
- David Sullivan– Clerk

At-large Directors:

- Candace Murphy
- Marge Shinkle
- Jim Turner

Our Gracious Hosts:

- Paul Ratty
- Marge Shinkle
- Jack Guilderson

Mission

- To promote use of engineering and management techniques that lead to delivery of high quality software
- To disseminate concepts and techniques related to software quality engineering and software engineering process
- To provide a forum for discussion of concepts and techniques related to software quality engineering and the software engineering process
- To provide networking opportunities for software quality professionals

SQGNE 2016-17 Schedule

Speaker	Affiliation	Date	Topic
Bob Crews	Checkpoint Technologies	Sept 14	Shift Left Testing: What the heck does that mean?
Michael Durrant	Everquote	Oct 12	Web Automation Fundamentals
Derek Kozikowski	**ZoomInfo**	**Nov 9**	**Using Selenium For Web Application Testing**
Dave Todaro	Ascendle Technology LLC	Dec 14	Testers and Developers are Best Friends
Robin Goldsmith	Go Pro Management, Inc.	Jan 11 2017	YOU Don't Need No Stinking Test Cases
Alex Seriy	IBM	Feb 8	Data-Driven Function, Integration and End-to-End Testing in the API Economy
Joe Zec	Shire Pharmaceuticals	Mar 8	How to build quality into software in 6 easy steps
Nikhil Kaul	SmartBear	Apr 12	Mobile App Testing: What and How to Test?
Mark Holland	Applause	May 10	In Their Shoes: Understanding Your Mobile User's Point of View
SmartBear and Checkpoint Tech	SmartBear and Checkpoint Tech	June 14	Test Tool Bakeoff *Annual Election of Officers*

Welcome Software Quality Group of New England

casenet

WHO WE ARE

Bedford, MA based company founded in 2003

93% customer satisfaction

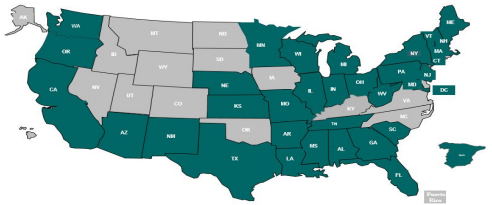
100% customer retention

175 Employees

99% on-time, on-budget implementations

Fortune 500 wholly-owned subsidiary of Centene

WHAT WE DO



33 states & Spain with TruCare presence

40% Commercial

60% Medicare/Medicaid, ACO & carve-outs

43 Health Plans representing 20 U.S. clients plus 1 in Spain

31 million members live or under contract

Extensible, scalable platform:

8,000 concurrent users

14M members

supported on a single instance

casenet

Tonight's Topic

Using Selenium for Web Application Testing

Speaker: Derek Kozikowski, ZoomInfo

Meeting Abstract:

- With Selenium WebDriver now available for all of the major browsers it has become the go to open source tool for web application automated testing. It provides a way to create robust, browser-based regression automation suites and tests that can scale across multiple test environments. Perhaps you've been using Selenium for a while now, and would like to find out more about it, or you would like a sense of WebDriver's capabilities. This talk will dig into the details of techniques for using WebDriver, discuss key best practices to apply when using the tool, and call out some of the challenges to success. You will come out of the session with a better understanding of how WebDriver works, learn some techniques to address commonly encountered issues, and a list of tools and references that will help you to take full advantage of WebDriver in your automated test environment.

Biography:

- Derek has a history of doing software testing and test automation at both large and small companies. Testing a broad range of applications including operating systems, security software, and enterprise web applications, has given him the opportunity to work with a wide variety of testing tools, techniques, and processes. He has presented talks at local software quality conferences as well as to the SQGNE over the past fifteen years on related topics, and has previously served on the Editorial Review Board for the ASQ's Software Quality Professional journal.

Using Selenium For Web Application Testing

by Derek Kozikowski

Agenda

- Configuration challenges
- Interacting with the Browser
- Interacting with the DOM
- Page Object Model
- Finding Elements
- Interacting with Elements
- Handling Exceptions
- Waiting
- Working outside of Selenium
- Wrap up

About Selenium WebDriver

From the documentation; “Selenium-WebDriver makes direct calls to the browser using each browser’s native support for automation. How these direct calls are made, and the features they support depends on the browser you are using.”

Uses browser-specific driver.

Programming language specific packages/modules also required.

Important! Keep browser and driver versions in sync and up to date

Discussion is limited to Selenium 2

WebDriver W3C Recommendation https://www.w3.org/TR/webdriver/

Configuration: Desired Capabilities

```
ChromeOptions chromeOpts = new ChromeOptions();
Map<String, Object> prefs = new HashMap<String, Object>();
prefs.put("profile.default_content_settings.popups", 0);
prefs.put("download.default_directory", "/abspath/for/downloads");

chromeOpts.setExperimentalOption("prefs", prefs);

chromeOpts.addExtensions(new File("/path/to/extension.crx"));

DesiredCapabilities capabilities = DesiredCapabilities.chrome();
capabilities.setCapability(ChromeOptions.CAPABILITY, chromeOpts);
WebDriver driver = new ChromeDriver(capabilities);
```

Defines properties for the WebDriver describing which features should be used for a session.

Shared and browser-specific properties.

You typically set session-specific properties via your test execution framework.

May retrieve critical info as well, e.g. webdriver.remote.sessionid

Configuration: WebDriverFactory

Use Configuration files to store driver and browser settings.

WebDriverFactory class configures WebDrivers using the settings based on the arguments to the constructor that typically indicate which browser is the target (e.g. Chrome), which returns a WebDriver instance.

A test setup method would pass the appropriate configuration to WebDriverFactory, which returns the configured WebDriver.

Lots of examples on the web, including FriendlyTester's WebDriverFactoryExample on GitHub

Interacting with the Browser

```
get('www.google.com')
getCurrentUrl()
getPageSource()
getWindowHandle()
manage()
navigate()
switchTo()
findElement(), findElements()
close(), quit()
...
```

IWebDriver interface for;

- controlling the browser directly,
- selecting WebElements in the DOM,
- Debugging tools.

Implementations for;

- Chrome
- Firefox
- Edge
- Safari
- RemoteWebDriver (Grid)
- ...

Interacting with the DOM: Standard methods

```
// find elements
WebElement q =
driver.findElement(By.name("q"));
List<WebElement> li =
    driver.findElements(By.tagName(“a”));

// click on an element
q.click();

// Collect information about an element
String firstLinkText = li[0].text;
```

Normally, most operations in the DOM are covered with existing methods.

Interacting with the DOM: Actions

```
# drag and drop
source_element = driver.find_element_by_id(
        'element_to_drag')
dest_element = driver.find_element_by_id(
        'element_to_drag_to')
ActionChains(driver).drag_and_drop(
        source_element, dest_element).perform()

# menu selections
menu = driver.find_element_by_css_selector("div.nav")
hidden_submenu = menu
        .find_element_by_css_selector("span#submenu1")
ActionChains(driver).move_to_element(menu)
        .click(hidden_submenu).perform()
```

Automate low level interactions with the DOM. Typical use cases are drag and drop, or complex menu selections.

E.g.: click_and_hold(), double_click(), key_down(), key_up(), release(), send_keys()

https://seleniumhq.github.io/selenium/docs/api/py/webdriver/selenium.webdriver.common.action_chains.html

Interacting with the DOM: Using JavaScript

```
// click on an element
((JavascriptExecutor) driver)
    .executeScript(
        "arguments[0].click();",
        element);
// Hide an element
js.executeScript(
    "document.getElementsByName('foo')[0]" +
    ".style.display='none'");
// Show an element
js.executeScript(
    "document.getElementsByName('bar')[0]" +
    ".style.display='block'");
// get computed style on an element (e.g. color)
js.executeScript(
     "Return window.getComputedStyle(" +
     "document.querySelector(' '), " +
     "':before').getPropertyValue('content')");
```

Ultimate workaround for limitations of WebDriver, or unique behavior of the web application under test.

Development Project with Page Object Model

Directory Structure

- Packaged according to dev language
- Organized by application function

File Structure

- Follow standards of dev language

Page Objects:

- Page of application
- “Dialog” built from HTML
- Row in a data table

Page object tips:

- Use inheritance for application pages, use composition for components within the page (e.g. dialogs, navigation menus, etc.).
- No validation (e.g. assert) statements
- Provide direct access to controls on the page
- Provide “helper” methods to easily accomplish typical actions (e.g. LoginPage.login(username, password))
- Relevant methods returns the page object instead of “void”, to allow chaining for fluent programming

Finding elements: Basics

Java: driver.findElement(By.name("q"));

C#: driver.FindElement(By.Name("q"));

Python: driver.find_element_by_name("q")

Ruby: driver.find_element :name => "q"

Javascript:
driver.findElement(webdriver.By.name('q'));

Perl: $driver->find_element("q", "name");

How best to target by: id, name, css, xpath

Learn how to use CSS selectors and xPath

Retrieve path suggestions using FirePath extension to Firebug for Firefox, or Chrome developer tools context menu option.

The By object is critical: (Java) ByClassName, ByCssSelector, ById, ByLinkText, ByName, ByPartialLinkText, ByTagName, ByXPath

Finding elements: finding links

<p>
The data presented in this plot is taken from the
<a href=
"https://github.com/hannorein/open_exoplanet_catalogue">Open Exoplanet Catalogue. All information on this page is directly generated from the XML files in the catalogue. If you are interested in how, look at the source code. You can download the entire catalogue on
<a href=
"https://github.com/hannorein/open_exoplanet_catalogue/">github.

...

xPath: //a[contains(@href, '/open_exoplanet_catalogue')]

CSS: a[href*='/open_exoplanet_catalogue']

The above would find multiple links on the page, let's be specific about parent and instance:

xPath:
//**p/**a[contains(@href,'/open_exoplanet_catalogue')]**[2]**

CSS: **p >**
a[href*='/open_exoplanet_catalogue']**:nth-of-type(2)**

Finding elements: by element attributes

```
<div id="percent-loaded"
        role="progressbar"
        aria-valuenow="0"
        aria-valuemin="0"
        aria-valuemax="100" />
```

When the only thing that makes an element unique is a specific attribute.

How to tell when the progress bar has completed loading? When aria-valuenow=100, the aria-valuemax value.

xPath:
//*[@id='percent-loaded'][@aria-valuenow='100']

CSS:
div#percent-loaded[aria-valuenow="100"]

Finding elements: Relative positioning

```
<fieldset>
   <div class="grid-3-12 form-no-lbl">
         <label class="form-lbl">CNPJ:
         </label>011234560083
   </div>
   <div class="grid-3-12 form-no-lbl">
         <label class="form-lbl">CIDADE:
         </label>TAUBATE
   </div>
   <div class="grid-3-12 form-no-lbl">
         <label class="form-lbl">ESTADO:</label>SP
   </div>
   <div class="grid-3-12 form-no-lbl">
         <label class="form-lbl">TOTAL BRUTO:
         </label>2.407,09
   </div>
</fieldset>
```

Sometimes CSS works, but this is where xPath shines.

xPath:

```
target = driver.find_element_by_xpath(
        "//fieldset/div/label[starts-with(., 'CIDADE:')]/..")
        .text
print(target.split(":")[-1].strip())
```

CSS:

```
targetdivs = driver.find_elements_by_css_selector(
        "fieldset div.form-no-lbl")
for target in targets:
        if target.find_element_by_css_selector("label")
                .text == 'CIDADE:':
                print(target.text.split(":")[-1].strip())
```

Finding elements: Absence of something

```
<ul class="nav nav-pills nav-stacked ng-hide"
ng-show="false">
<li>first</li>
</ul>

<br>

<ul class="nav nav-pills nav-stacked">
<li>first</li>
</ul>
```

Looking for an element that doesn't include a specific thing:

xPath:

```
//ul[contains(@class,'nav-pills') and
        not(contains(@class, 'ng-hide'))]
```

CSS:

```
ul.nav.nav-pills:not(.ng-hide)
```

http://stackoverflow.com/questions/36164807/how-to-avoid-the-hidden-classes-using-selenium-webdriver

Finding elements: Containing Text

```
<table id="systemtable" class="tablesorter
tablesorter-default" summary="List of
exoplanets" role="grid" cellspacing="0">
<tbody id="systemtablebody"
aria-live="polite" aria-relevant="all">
...
<tr class="odd" role="row">
<td>
<a href="/planet/Kepler-450%20c/">
<span class="numericvalue">
        Kepler-450 c
</span>
</a>
</td>
```

Avoid this when you can, especially for localized applications!

To get an element by it's text value, use the locator:

xPath: //span[contains(text(), '**Kepler-450 c**')]

CSS: you can't do this w/ CSS now.

Interacting with Elements: Menus

```
def choose_menu(main, sub):
        main_menu = wait.until(
                EC.visibility_of_element_located(
                        (By.CSS_SELECTOR,
                        "li.item a#" + main)))
        # hover over main_menu
        ActionChains(driver).move_to_element(
                main_menu ).perform()

        sub_menu = wait.until(
                EC.visibility_of_element_located(
                        (By.CSS_SELECTOR,
                        "li.item a#" + sub)))
        # click on sub menu
        ActionChains(driver).move_to_element(
                sub_menu).click().perform()
```

Menus are a common control in web applications. To automate interactions with these menus, code must dynamically toggle state, while waiting for that state to change.

The Page Object Model calls for methods that interact with main navigation menus to be defined in a parent class so that they are available to all child page objects.

Debugging NoSuchElementException

- Is the path correct?
  - Test it with a browser debugging tool
- Is the path unique?
  - Make the path more specific by including a nesting element
  - If it is one of many, get the n-th one, or use a findElements() method to grab a list to process further in your code
- Is it inside of an iframe?
  - Use switchTo() to shift Selenium's context
- Did you wait for it long enough?

Waiting, waiting, waiting, ...

Learn more about race conditions and Selenium at
https://bocoup.com/weblog/a-day-at-the-races

No Thread.sleep()!!!

Implicit vs Explicit

- stackoverflow discussion about combining them
- stackoverflow discussion about where to put waits

Have a rule for waiting... E.g method waits until stability of required state for its action. Be consistent.

WebDriverWait() with ExpectedCondition()

WebDriverWait() and ExpectedCondition()

```
WebElement myDynamicElement = (
        new WebDriverWait(driver, 10)).until(
                ExpectedConditions
                        .presenceOfElementLocated(
                                By.id("theId")));
```

We've seen examples previously. A few useful references:

http://www.seleniumhq.org/docs/04_webdriver_advanced.jsp#explicit-waits

http://www.swtestacademy.com/selenium-9-synchronization/

https://seleniumhq.github.io/selenium/docs/api/py/webdriver_support/selenium.webdriver.support.expected_conditions.html

Custom Expected Conditions

```
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.common.by import By
class completed_progress_bar_located(object):
    def __init__(self, selector, percent):
        """
        :param selector: for the progress bar
        :param percent: desired percent complete
        """
        self.selector = selector
        self.percent = percent
    def __call__(self, driver):
        progress_bar = EC._find_element(driver,
            (By.CSS_SELECTOR, self.selector))
        percent_now =
progress_bar.get_attribute('aria-valuenow')
        return int(percent_now) == self.percent
```

Use a custom ExpectedCondition within a WebDriverWait to wait for the progress bar from our previous example to complete:

```
WebDriverWait(self.driver, 30).until(
        completed_progress_bar_located(
            the_bar, 100))
```

Debugging StaleElementReferenceException

The element you are addressing is simply no longer on the page.

- Has the browser navigated to a new page?
  - Compare your test with the application behavior to figure out if they match
- Is there asynchronous activity that has removed, or updated the target element?
  - Use findElement() to grab it again (avoid race conditions though!)
  - Instead of using @FindBy annotation and initPage() to define the WebElement, define a By object with a selector for your target, then use findElement() to retrieve the WebElement just at the time you want to use it.

See http://www.seleniumhq.org/exceptions/stale_element_reference.jsp for more information!

Working outside of Selenium

File Upload: Type the subject file path into the input field used on the web page form (it may be hidden). (use setFileDetector() method with RemoteWebDriver)

File Download: Besides configuring the browser option pointing to a download location, use httpclient to download files (http://jsystem.org/downloading-files-using-selenium-and-apache-httpclient/).

Date Pickers: Type the properly formatted date string into the field.

Dynamic Tables: Use product API to access data that's presented on the page.

WebDriver Friendly Frameworks

Selenium2Library for Robot Framework (https://github.com/robotframework/Selenium2Library)

Protractor for AngularJS applications (http://www.protractortest.org/#/)

Nightwatch.js with Node.js (http://nightwatchjs.org/)

Geb with Groovy (http://www.gebish.org/)

PhantomJS for headless testing (http://phantomjs.org)

...

Wrap up

- DesiredCapabilities and WebDriverFactory
- Interacting with the Browser and the DOM via driver methods
- Finding different elements with CSS and XPath
- Interacting with elements using WebElement methods and Actions
- Page Object Model
- WebDriverWait and Expected Conditions
- Handling different types of particularly challenging exceptions
- Addressing actions when Selenium can’t help you, like file uploads/downloads

Contact

Derek dot Kozikowski
gmail.com

Some Additional references that you may find helpful:

https://www.w3.org/TR/webdriver/

http://year-2015.seleniumconf.org/

http://elementalselenium.com/tips

https://burdettelamar.wordpress.com/2014/03/21/keep-your-page-objects-dry/

https://saucelabs.com/resources/articles/selenium-tips-css-selectors

http://www.w3schools.com/xml/xpath_intro.asp

http://api.jquery.com/category/selectors/

https://developer.mozilla.org/en-US/docs/Mozilla/QA/Marionette/WebDriver

http://www.seleniumhq.org/docs/03_webdriver.jsp

https://github.com/SeleniumHQ/selenium/wiki

http://stackoverflow.com/questions/37892691/python-selenium-how-extract-text-after-element

Questions?