SeU Certified Selenium Engineer (CSE) Syllabus

Released
Version 1.0 2019

Selenium United

Copyright Notice

Thank you to the main collaborators

- Rahul Verma, Vipul Kocher, Chandra Mouli Maddala and Jayapradeep Jiothis

Revision History

Version	Date	Remarks
SeU 2018	August 2018	First official release
SeU 1.0 2019	Jan 2019	Tech-Neutral Version

Table of Contents

Purpose of this document

This syllabus forms the basis of Selenium Untied in regards to its Certified Selenium Engineer (CSE) certification. This document defines what you need to know in order to pass the certification test for Certified Selenium Engineer (CSE), and is copyright of Selenium United. The certification test will only cover concepts and knowledge that are described in this document.

Resources of the SeU

An overview of SeU resources as well as all relevant information about the SeU certification and other types of SeU certifications is available on www.selenium-united.com — the official website of Selenium United. The information on www.selenium-united.org includes:

- A complete list of recognized SeU training providers and available courses. Note that training is recommended but not required in order to take the SeU CSE certification exam.
- SeU Syllabus (this document) for download.
- A complete sample exam set of 40 SeU CSE questions with answers for training purposes.
- We aim to have the documents available in further languages as soon as possible. For currently available language versions, please check www.selenium-united.com

What is Selenium?

Selenium is a suite of different tools meant for browser automation. With its comprehensive library for testing web applications, it has become the most sought-after tool by software testers. With support for all of the widely used browsers, it has also grown beyond just being another software testing library. It is the backbone of countless browser automation tools, APIs and frameworks. The WebDriver interface introduced by Selenium WebDriver is recommended to become a W3C standard, which would further increase its importance in the space of test automation.

About SeU Certified Selenium Engineer (CSE)

SeU Certified Selenium Engineer (CSE) is a practitioner level course for testers involved in web test automation. The course covers Selenium as a browser

automation library from the ground up. Test automation constructs and design are kept to a minimum to better focus on code constructs that enable usage of Selenium in a properly designed manner. This design-focused course provides an un-diluted Selenium experience, focusing more on the critical Selenium concepts, which enables better real-life implementation in the participants' daily work.

Business Outcomes

Business Outcomes (BOs)

BO 1	Adapt existing testing & test automation experience and knowledge to develop automated tests for web applications using Selenium.
BO 2	Use Selenium to create automation tests for Web Applications.
BO 3	Debug Selenium-based automated tests for correct functionality.

Learning Objectives/Cognitive Levels of Knowledge

Learning objectives (LOs) are brief statements that describe what you are expected to know after studying each chapter. The LOs are defined as follows:

- K1: Remember
- K2: Understand
- K3: Apply

Following table lists the main LOs for the SeU CSE certification:

LO1	Understand the importance of browser coverage and distinguish among various options for testing UI of web applications. (K2)
LO2	Understand the relationship of Web UI with the underlying HTML, JavaScript and CSS with DOM Inspection. (K2)
LO3	Recall the history of Selenium and various tools in its suite along with their purpose. (K1)
LO4	Understand the Selenium Architecture in terms of Language Bindings, Communication Protocols and Drivers. (K2)
LO5	Recognize the purpose and API of Web UI Automation at Browser,

	Page and Element levels. (K2)
LO6	Understand and Explain the purpose of test automation engine constructs for defining tests, fixtures and assertions. (K3)
LO7	Apply different identification strategies for UI elements. (K3)
LO8	Apply different inquiry and interaction strategies for UI elements. (K3)
LO9	Apply various deeper Selenium automation constructs, which set the basis for more advanced automation. (K3)
LO10	Automate end-to-end user scenarios by using good coding practices and Object-Oriented principles for placing Selenium code across different classes and methods. (K3)
LO11	Troubleshoot and describe scope of improvement for automation implementation depicted in short code samples. (K3)

Programming Language Prerequisites

The CSE workshop/content can be delivered in any programming language that has Selenium bindings available. As per the programming language chosen for delivery of a workshop, the facilitator would brief upon the relevant language concepts as and when they are used in the course. However, such concepts would only be referred to and not explained, unless a day is added to the workshop to account for this additional coverage.

Participants who are well versed with core programming concepts in the language used as the medium of delivery would be able to focus on the Selenium concepts in a much better manner, without diluting their attention to understand programming constructs.

The CSE exam would include code snippets relevant to the CSE syllabus. Currently CSE is available in Java and Python.

Participants are expected to have working knowledge of following concepts, for the programming language that they have chosen for CSE:

- Concept of “main”/execution entry point
- Compiling and running code
- Primitive data types, corresponding classes as supported by the language and user defined types using Classes
- Arrays
- Basic collections: List, Map/Dictionary, Set
- String formatting and manipulation
- Printing to STDOUT and STDERR
- Conditional control structures
- Looping Control structures
- Exception Handling and Exception hierarchy
- Encapsulation: Access modifiers and access methods
- Object Construction and Initialization
- Class versus Object level access: Methods and Variables
- Enumerations
- Creating Packages and Modules
- Inheritance and Polymorphism
- Abstract classes and abstract methods
- Object Composition
- Adding third-party dependencies (e.g. in Java one can add Jars as direct project configuration or via Maven)

Chapter 1 - Web UI Automation

LO1	Understand the importance of browser coverage and distinguish among various options for testing UI of web applications. (K2)
LO2	Understand the relationship of Web UI with the underlying HTML, JavaScript and CSS with DOM Inspection. (K2)

Summary:

- Introduction
- UI Automation with Actual Browsers
- UI Automation with Actual Browsers with Screen Size Simulation
- Using Headless Browsers
- Web UI: User vs Browser point of view

Background:
Web applications need to be available to a variety of users using a variety of devices. Many of them employ fluid interfaces so that the applications are rendered differently for different browsers. The browsers themselves behave differently in certain situations as they have different canvas sizes and underlying technology.

This makes Browser coverage an essential aspect in testing of web application UI whether it is done by a human or a piece of code. A Selenium test automation engineer must understand how to use Selenium to automate different browsers along with different options to mimic a browser. For speed and efficiency purpose, using a headless browser in automation is another option the engineer must be aware of.

The browsers render UI controls that are represented in HTML to load them into a DOM (Document Object Model) object and visually depict them in their canvas, using browser defaults and style overrides in CSS. JavaScript is used for customized event handling, sending AJAX requests to the server and DOM manipulation. A test automation engineer must be able to associate what is seen on the screen to its definition in HTML/DOM by using DOM inspection. This information is used as input to the test automation code for identification and interaction with such elements.

Web applications have different types of UI elements which support different types of actions that a user can take e.g. a user can click a button, enter text in a text box, select an option from a drop-down list, etc. A user can also visually verify the state of a particular UI element, for example whether it is enabled, what text it contains, etc. A test automation engineer should be able to make all such inquiries and take all types of actions in an automated test.

A Test Automation Engineer must also be aware of CSS styling, and in which parts of HTML it can be specified. This is also true for JavaScript, which is the primary client-side scripting language used by modern web applications.

Chapter 2 - Introduction to Selenium

LO3	Recall the History of Selenium and various tools in its suite along with their purpose. (K1)
LO4	Understand the Selenium Architecture in terms of Language Bindings, Communication Protocols and Drivers. (K2)

Summary:

- History
- Power of Selenium
- Selenium Suite
- Simplified Selenium Architecture

Background:
With its beginning at ThoughtWorks and later contributions by various stalwarts from the community, Selenium has come to the forefront of Web UI automation. The Selenium WebDriver API, touted to become a W3C standard, is used by various non-Selenium tools like Appium, to provide the interface to their underlying implementation.

It is good to know the history of Selenium and to keep up with its development road map so that a test automation engineer knows when and why certain features were introduced and where the future of Selenium and related tooling is headed.

Selenium enables multiple language bindings because of architecture that separates the responsibilities among language-specific client bindings and a browser-specific driver component that employs JSON Wire protocol. Understanding this architecture helps a test automation engineer know which components provide which facilities.

Chapter 3 - Automating the Web UI With Selenium

LO5	Recognize the purpose and API of Web UI Automation at Browser, Page, Element levels. (K2)
LO6	Understand and Explain the purpose of test automation engine constructs for defining tests, fixtures and assertions. (K3)
LO7	Apply different identification strategies for UI elements. (K3)
LO8	Apply different inquiry and interaction strategies for UI elements. (K3)

Summary:

- Introduction
- Browser Level Automation
    - Launching/closing different browsers
    - Navigation
    - Inquire window and URL information
- Page Level Automation
    - Inquire page level information
    - Element Identification in depth
        - ID
        - Name
        - Class Name
        - Link Text
        - Partial Link Text
        - CSS Selectors — coverage of different variants
        - XPath — coverage of different variants
- Element Level Automation
    - State inquiry
    - Basic Actions

Background:

For Web UI Automation, you need automation API at 3 levels:

1. Browser Level — These actions do not depend on a particular web page and are provided by interface in Selenium.
2. Page Level — These actions depend on the current page loaded in the browser and are also provided by the WebDriver object.

3. UI Element level — These actions are related to a particular UI element. These are mostly provided by WebElement object.

A test automation engineer must know which actions Selenium supports via its API across these levels. The engineer is not expected to remember all the API. However, the engineer must be equipped to refer the Selenium API and recognize the purpose of different API calls and should be able to use them while writing a test automation code.

In the process, the engineer should know how to use a test automation engine to utilize different standard features such as test representation, test fixtures, assertions, to use along with the Selenium code in order to write an automated test. For example, in the Java world one could use TestNG or JUnit and learn how to use the corresponding annotations, constructs and run strategies effectively.

For the identification of elements, Selenium supports various locator strategies via its By object:

- By Name
- By ID
- By Class Name
- By Tag Name
- By Link Text and Partial Link Text
- By CSS Selector
- By XPath

A test automation engineer must know the purpose of each of these locator strategies. S/he should be able to choose the right locator strategy for a given situation. CSS Selectors and XPath are extremely powerful and provide multiple ways of defining a location strategy. The test automation engineer must understand CSS Selector and XPath in depth, as well as understand the flexibility and performance implications of different strategies.

Interactions with UI elements also need pre and post element inquiries to check whether an action can be taken and to verify whether an action has been successfully taken respectively.

Chapter 4 - Beyond Simple Selenium Code Constructs

LO9	Apply various deeper Selenium automation constructs, which set the basis for more advanced automation. (K3)

Summary:

- Better Waiting
- Handling Drop-down Lists
- Matching Multiple Elements
- Handling Nested Elements
- Uploading a File
- JavaScript Execution
- Handling Windows/Tabs
- Handling Alerts
- Handling Frames
- Taking Screenshots
- Action Chains
- Keyboard Actions
- Handling Cookies
- Headless Browsing

Background:
Because of network latency and other client-side operations, the web UI might not be in a state where a desired action can be taken. For example, one might want to click a button, however, that button has not yet been rendered, or the button is not yet clickable. An automated test, in practice, contains code to wait for such a desired state before taking an action. Such a wait is also necessary before an element can be identified to wait for page loading. Rather than using hard-coded, static waits, tests should use a polling-based wait mechanism (dynamic waits) provided by Selenium.

Selenium provides a Select object to enable higher-level API for drop-down lists.

There are situations in which one would need to identify multiple elements as per a locator strategy and act on them. There are other situations where

elements are found not from the root of DOM, rather as child elements within an element. Selenium supports automation of such scenarios.

Uploading a file is a pretty common feature in web applications and the automation code should be able to simulate this feature.

Selenium supports execution of raw JavaScript in the context of a browser. This is a very handy feature, using which a test automation engineer can inject any kind of custom JavaScript in the browser and execute it. One can also pass objects to such JavaScript from Selenium code as well as receive an object from the executed JavaScript.

Other facilities needed in automated tests are for handling windows/tabs, alerts and IFrames.

When a failure occurs, an automated test can take a screenshot of the contents of the browser so that this information assists in troubleshooting. Selenium supports taking such screenshots in different formats.

Some actions are chained together to represent a single action. For example, one could hover on a navigation bar, which in turn reveals a child sub-menu and one might click a sub-menu item. These can be simulated in Selenium by using an action chain. It is also used to simulate complex keyboard actions.

Usually, a web application manages its state as well as user preferences using Cookies. Selenium provides access to cookies and related actions can be implemented in automated test.

Selenium supports headless browsing if a given browser provides way to do so. Chrome and Firefox have a headless option. HtmlUnitDriver is a browser specially developed as a headless browser. This is a very handy feature that can be used by automation engineers to run a lot of browsers in parallel, with reduced resource usage per browser.

Chapter 5 - Putting Together a Basic Framework

LO10	Automate end-to-end user scenarios by using good coding practices and Object-Oriented principles for placing Selenium code across different classes and methods. (K3)
LO11	Troubleshoot and describe scope of improvement for automation implementation depicted in short code samples. (K3)

Summary:

- Long Exercise: Automating End To End Scenarios
- Refactoring the code to create and use a WebAutomator class which wraps construction and interactions with WebDriver
- Next Steps: (High Level Walkthrough and Demo)
    - Implementing Event Listener
    - Page Object Design Pattern
    - Using Page Factories and Pages as Loadable Components (available in some language bindings)
    - Selenium Grid

Background:
Without implementing good programming practices and design patterns, an automated code soon bloats with many duplicate instructions and difficult-to-maintain code. Coding inconsistencies also creep into the code as different engineers code in their own specific style.

There is value in putting all the commonly used facilities in a central framework, which is used to write tests. One can use the Object-Oriented principles to put together a basic framework very easily and continue to enhance its feature set as the project progresses.

Although it is not the direct subject area of this course, it is still important to understand that there are many measures, which a test automation engineer can take to put together a robust framework. So, one must know concepts of Page Object Model (POM), Event Listener, Selenium Grid etc. which can further enhance the features and robustness of a framework. Some language bindings also provide additional facilities like Page Factory and Loadable Component to support easy development of Page Object Model. These concepts are covered at

a high level in CSE so that participants can continue learning beyond this programme. CSE exam does not cover these concepts.

References:

- SeleniumHQ Website: https://www.seleniumhq.org/
- Everything has been designed and created utilizing first-hand experiences gathered from the industry by the SME's involved in creating Selenium United.