Software Project Management - Laboratory

Lecture n° 9
A.Y. 2020-2021

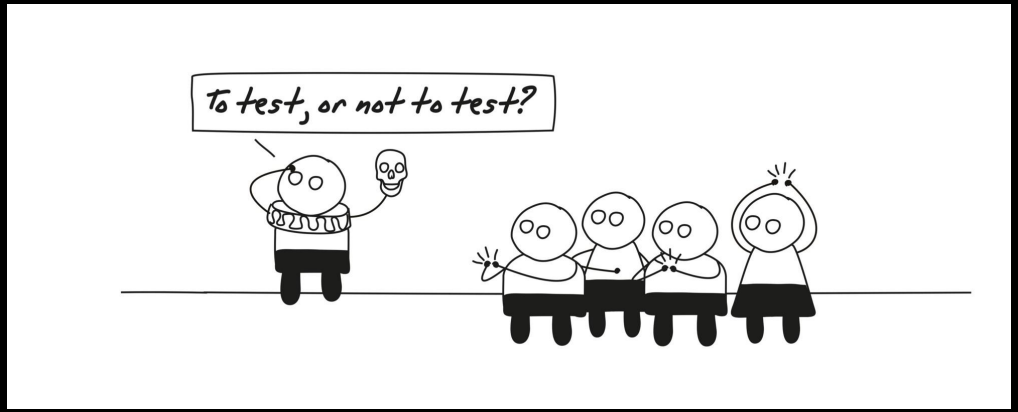
Prof. Fabrizio Fornari
fabrizio.fornari@unicam.it

Unit Testing - Recap

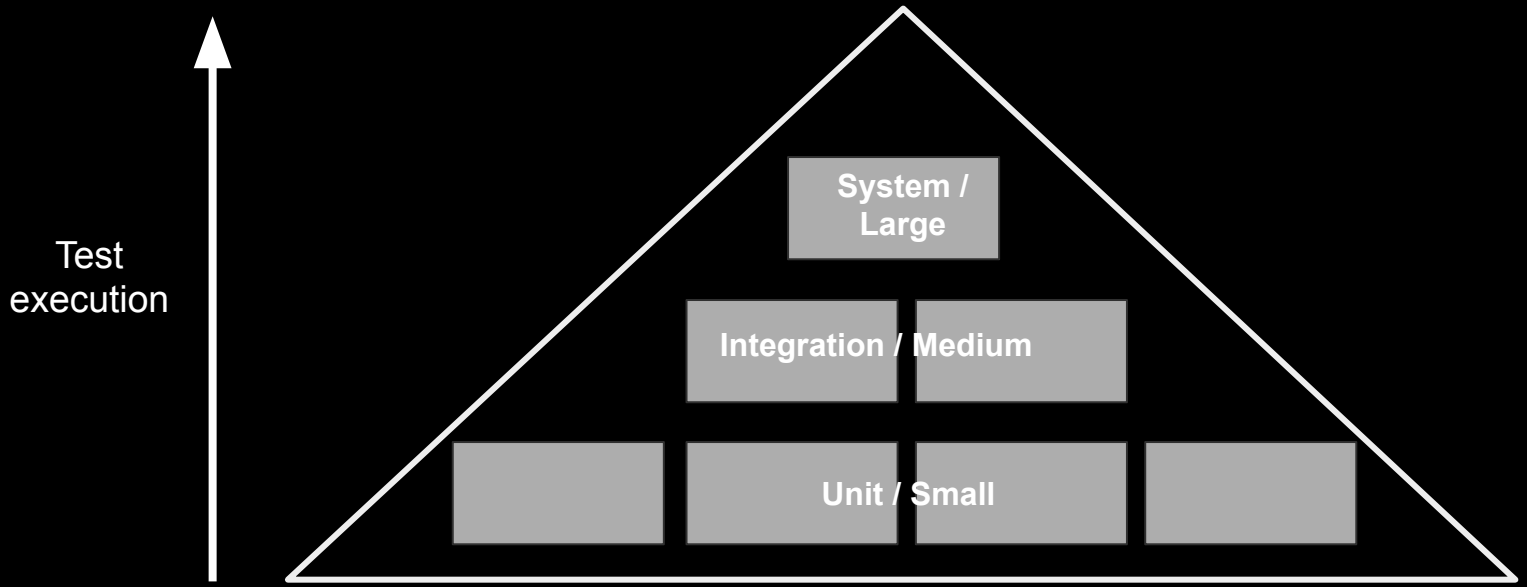
with JUnit

Unit Testing

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently analysed for proper operation.



The Test Stack



Types of Testing

- Unit Testing
- Integration Testing
- Regression Testing
- ...

https://www.softwaretestinghelp.com/types-of-software-testing/

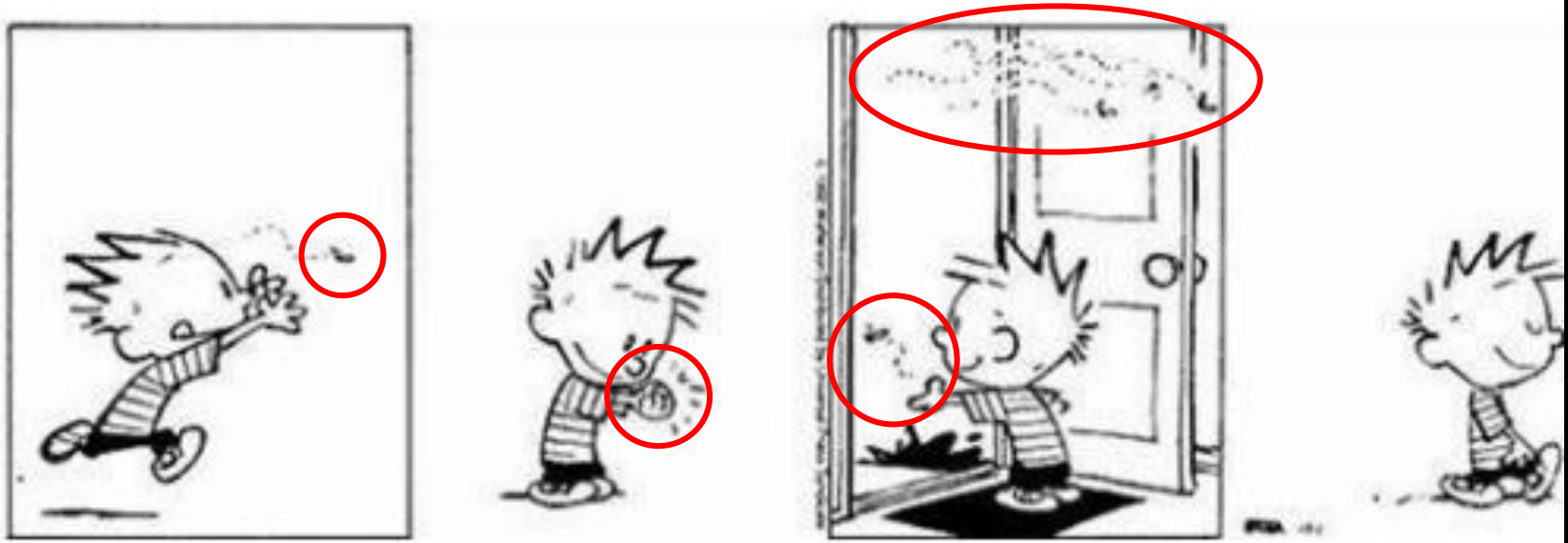
Integration Testing

Individual modules are combined and tested as a group.
Data transfer between the modules is tested as well.

Regression Testing

Regression means retesting the unchanged parts of the application.

Regression:
"when you fix one bug, you introduce several newer bugs."



Regression Testing

Test cases are re-executed in order to check whether the previous functionality of the application is working fine and the new changes have not introduced any new bugs.

This test can be performed on a new build when there is a significant change in the original functionality that too even in a single bug fix.

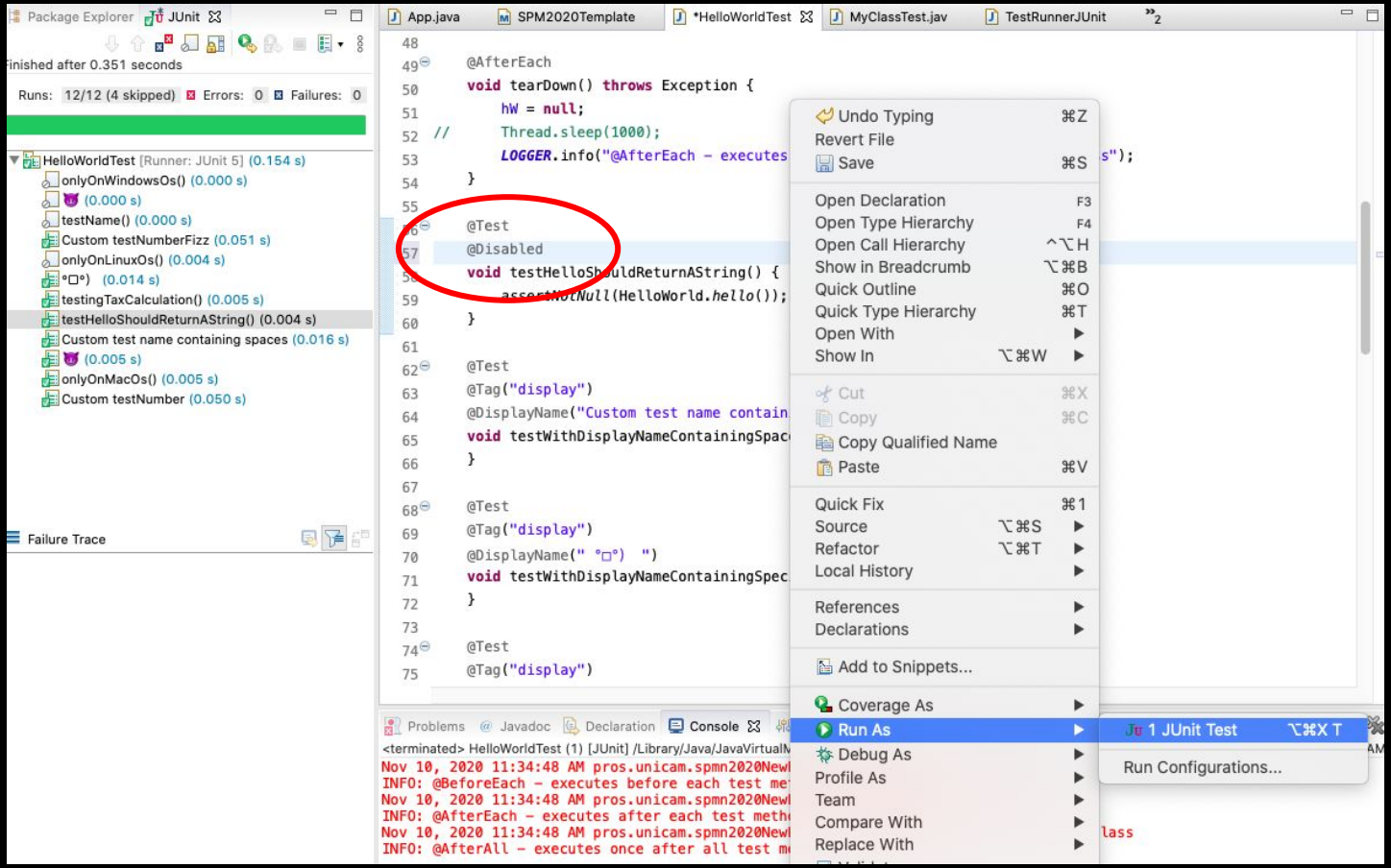
Manual Testing

The oldest type of software testing.

It requires a tester to perform manual test operations on the test software without automation scripts.

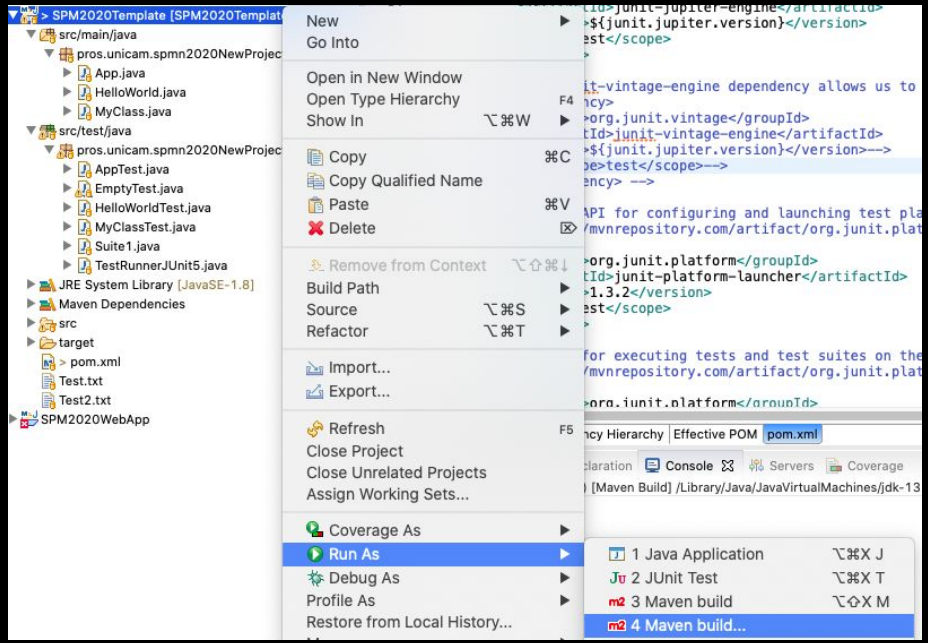
The tester choose which tests to run, when to run them, and how many times.

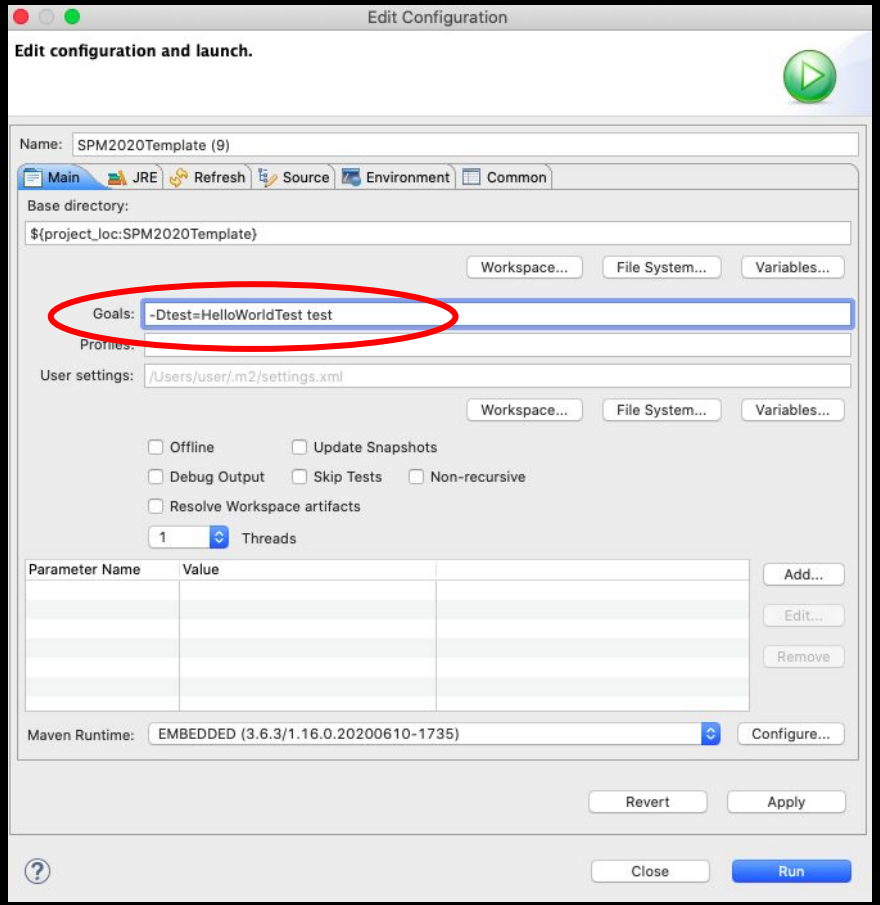
Manual Testing in Eclipse



Manual Testing with Maven

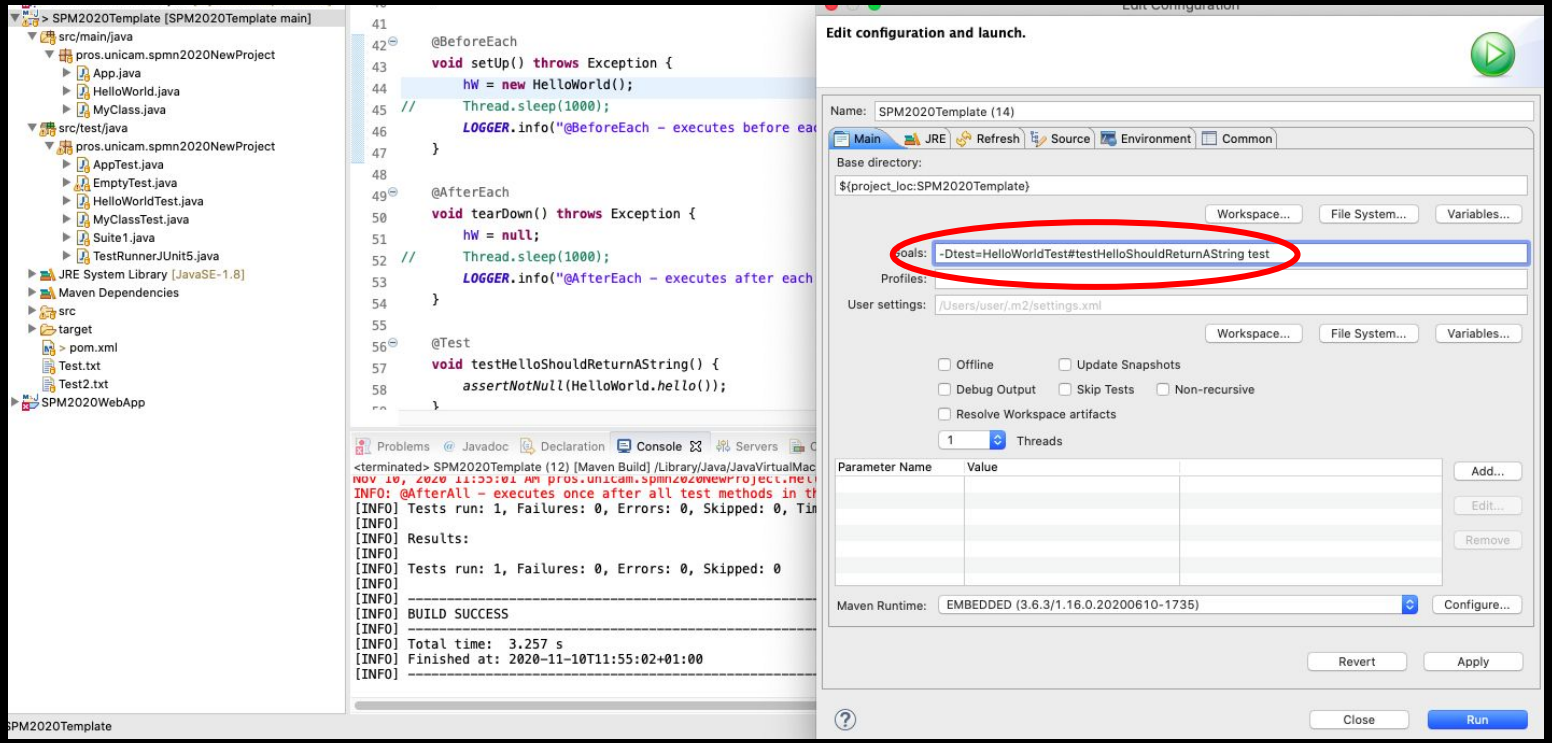
- Run a single test class:
  -Dtest=<NameOfTheTestClass> test





Manual Testing with Maven

- Run a single test method from a test class:
  -Dtest=<NameOfTheTestClass>#<NameOfTheTestMethod> test



Manual Testing with Maven

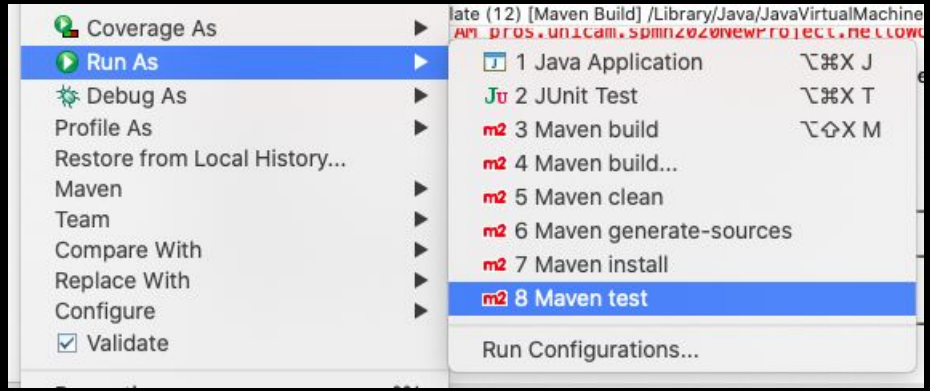
- Run a single test class:
  -Dtest=<NameOfTheTestClass> test
- Run a single test method from a test class:
  -Dtest=<NameOfTheTestClass>#<NameOfTheTestMethod> test
- Run multiple test classes:
  -Dtest=<NameOfTheTestClass>,<NameOfTheTestClass> test
- Run all test methods that match pattern 'testMethod1*' from a test class:
  -Dtest=TestApp1#testMethod1* test
- Run all test methods match pattern 'testMethod1*' and 'testMethod2*' from a test class:
  -Dtest=TestApp1#testMethod1*+testMethod2* test
- Run all tests tagged with a specific tag eg. “display”:
  test -Dgroups=display

Manual Testing with Maven

Use the "Test" suffix at the end of test classes names.

ClassNameTest es. MyClassTest or HelloWorldTest...

The Maven build system automatically includes such classes in its test scope.



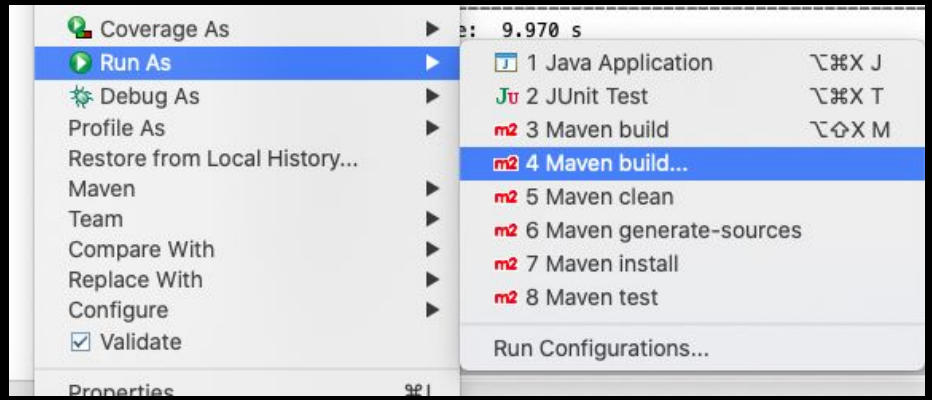
Automated Testing

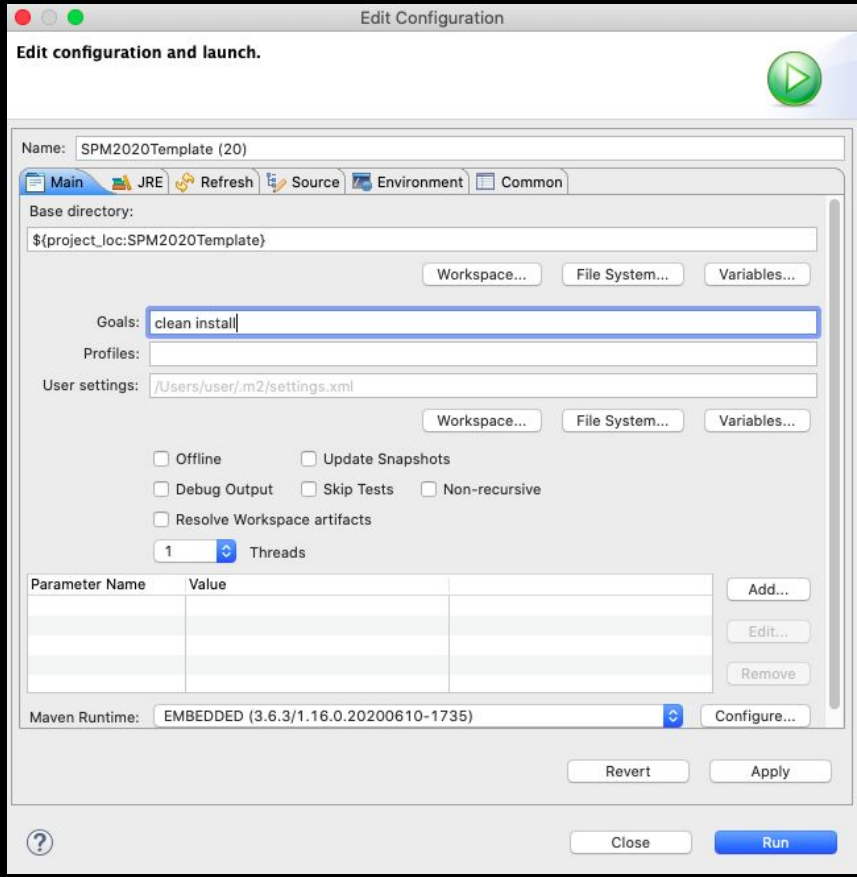
Automated testing

To automatically verify main functionality, ensure new version does not cause new defects, provide regression testing and help the teams to run a large number of tests in a short period of time.

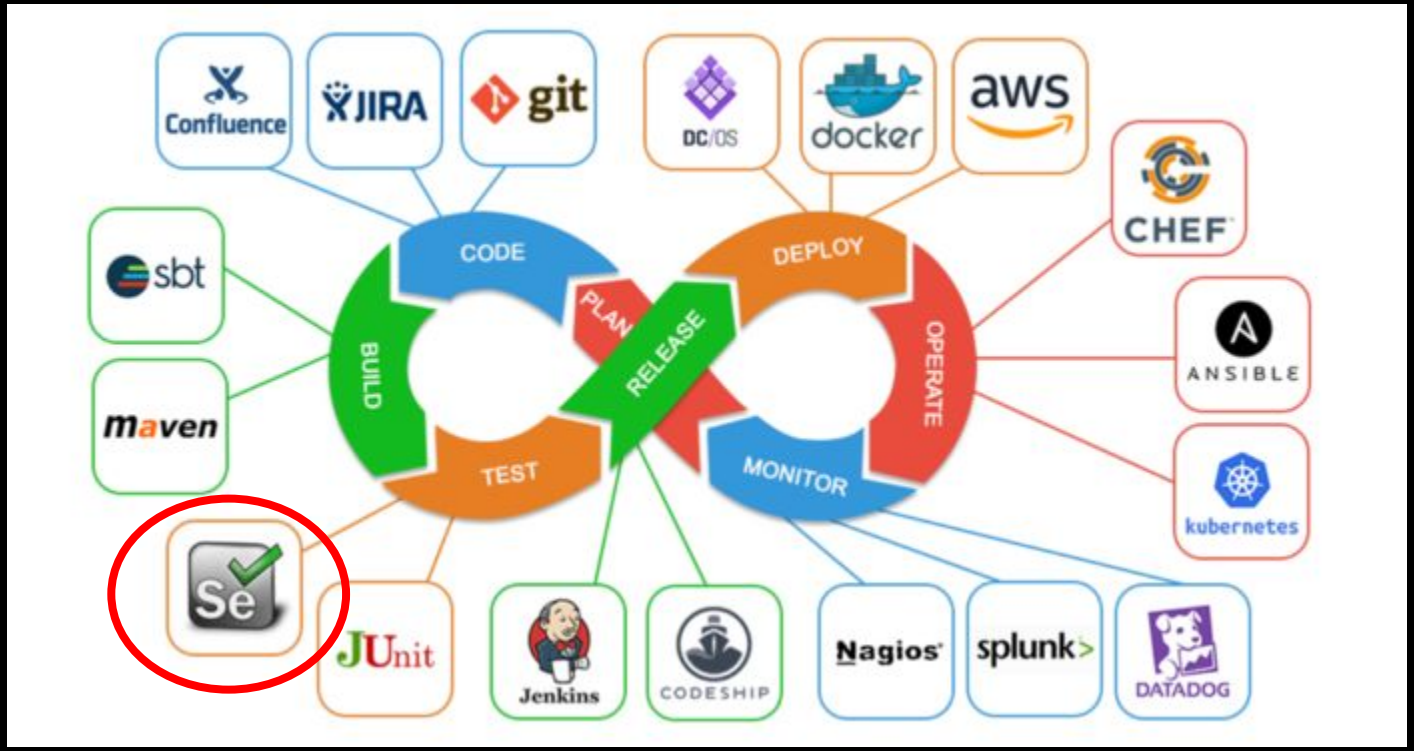
Companies having great number of projects are looking for specialists in the field of automated testing.

Automated Testing with Maven





DevOPs


Confluence
JIRA
git
DC/OS
docker
aws
CHEF
sbt
maven
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
ANSIBLE
kubernetes
Se
JUnit
Jenkins
CODESHIP
Nagios
splunk>
DATADOG

Automated web testing

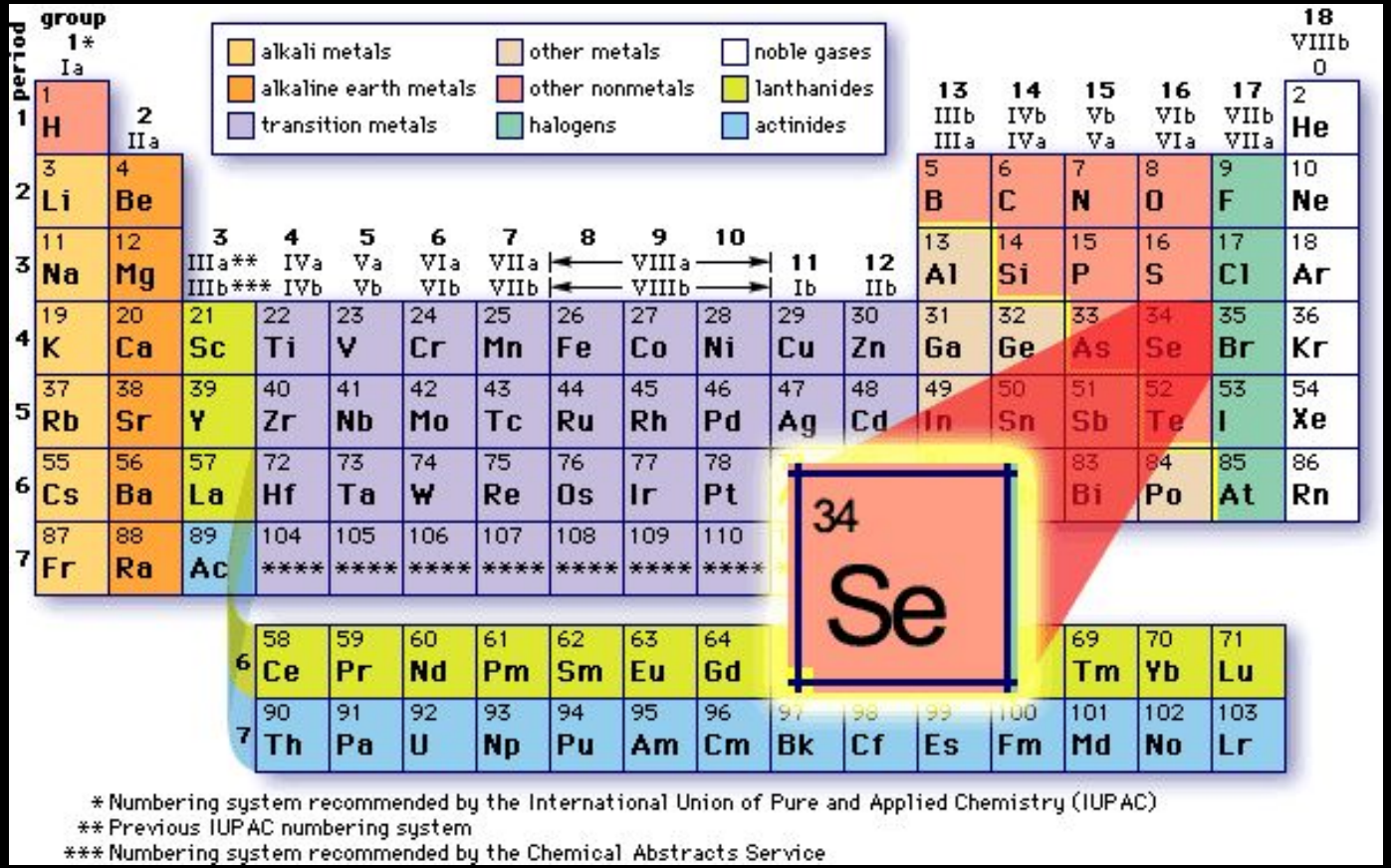
with Selenium

What is Web Testing?

What is Selenium?

Selenium

is a chemical element with symbol **Se** and atomic number **34**



What is Selenium?

Selenium is a suite of tools to automate web browsers across many platforms.

We will use it for automating tests of web applications

Why is it called Selenium then?

Why is it called Selenium then?

During the development of selenium core 2004 there was another competitive product developed by a company called **Mercury Interactive**.

A joke by Jason Huggins saying that ***"mercury poisoning can be cured by taking selenium supplements".***

Selenium is used to remove the toxic content mercury from the human body, so Jason coined the term Selenium for their creative open-source project.

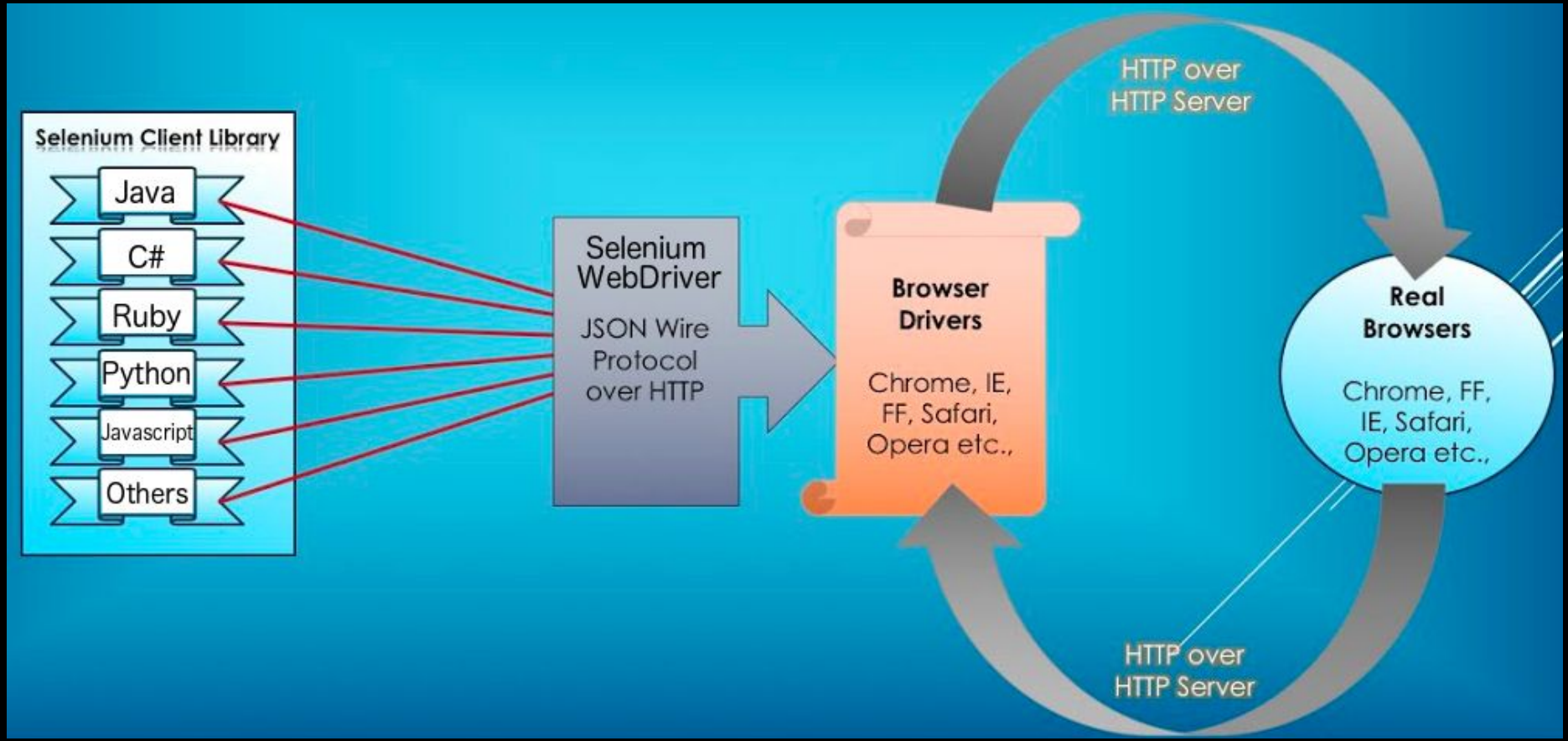
Which is the idea?

To remotely control browsers so that we can do things like write automated tests for the content they run or tests for the browser UI itself.

Should I write a test in a different way per each browser that is out there? No, to this end, a group of people from several organizations is working on the WebDriver Specification.

https://w3c.github.io/webdriver/

Selenium Architecture



Selenium WebDriver

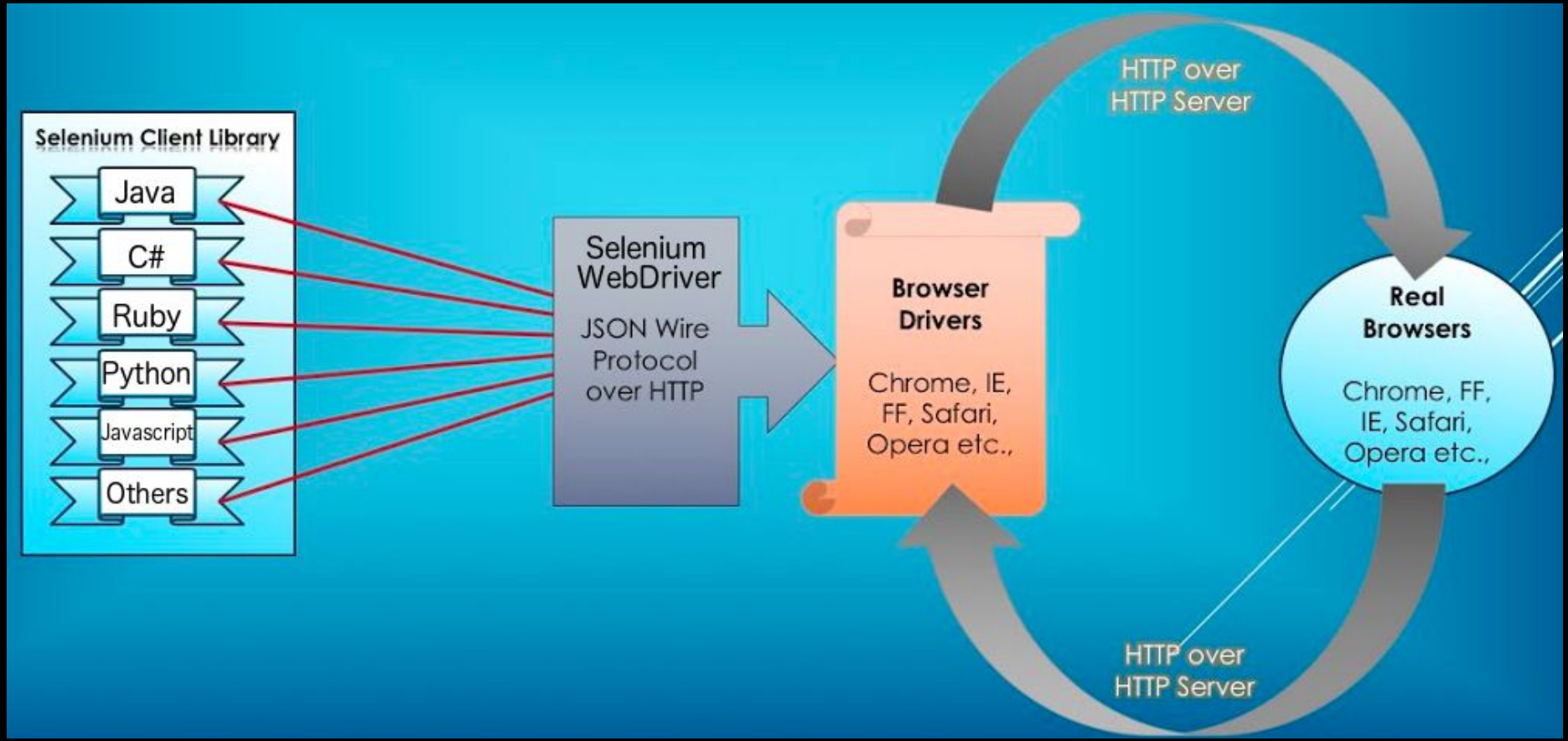
Selenium WebDriver provides APIs so that you can write code in your favourite language to simulate user actions like this:

```
client.get("http://pros.unicam.it/")
link = client.find_element_by_id("participate")
link.click()
```

Underneath that API, commands are transmitted via JSON over HTTP.

For example,
to tell the browser to navigate to a url, a client sends a POST request to the endpoint */session/{session id of the browser instance you're talking to}/url* with body *{"url": "http://pros.unicam.it/"}*.

Selenium Architecture



Browser Driver

At the beginning Selenium had to develop drivers for some browser they wanted to interact with.

Then, browser vendors started implementing the Selenium JSON Wire Protocol themselves!

This makes a lot of sense: they're in the best position to maintain the server side and they can build the necessary behaviour directly into the browser.

It started with OperaDriver in 2009-2011, and then others followed such as ChromeDriver and Mozilla's geckodriver with Marionette. This is where the motivation for a WebDriver standard comes from.

Selenium

GO TO: https://www.seleniumhq.org/

User Guide:
https://www.selenium.dev/documentation/en/getting_started/

Common Steps to follow

Step 0 - Open Eclipse

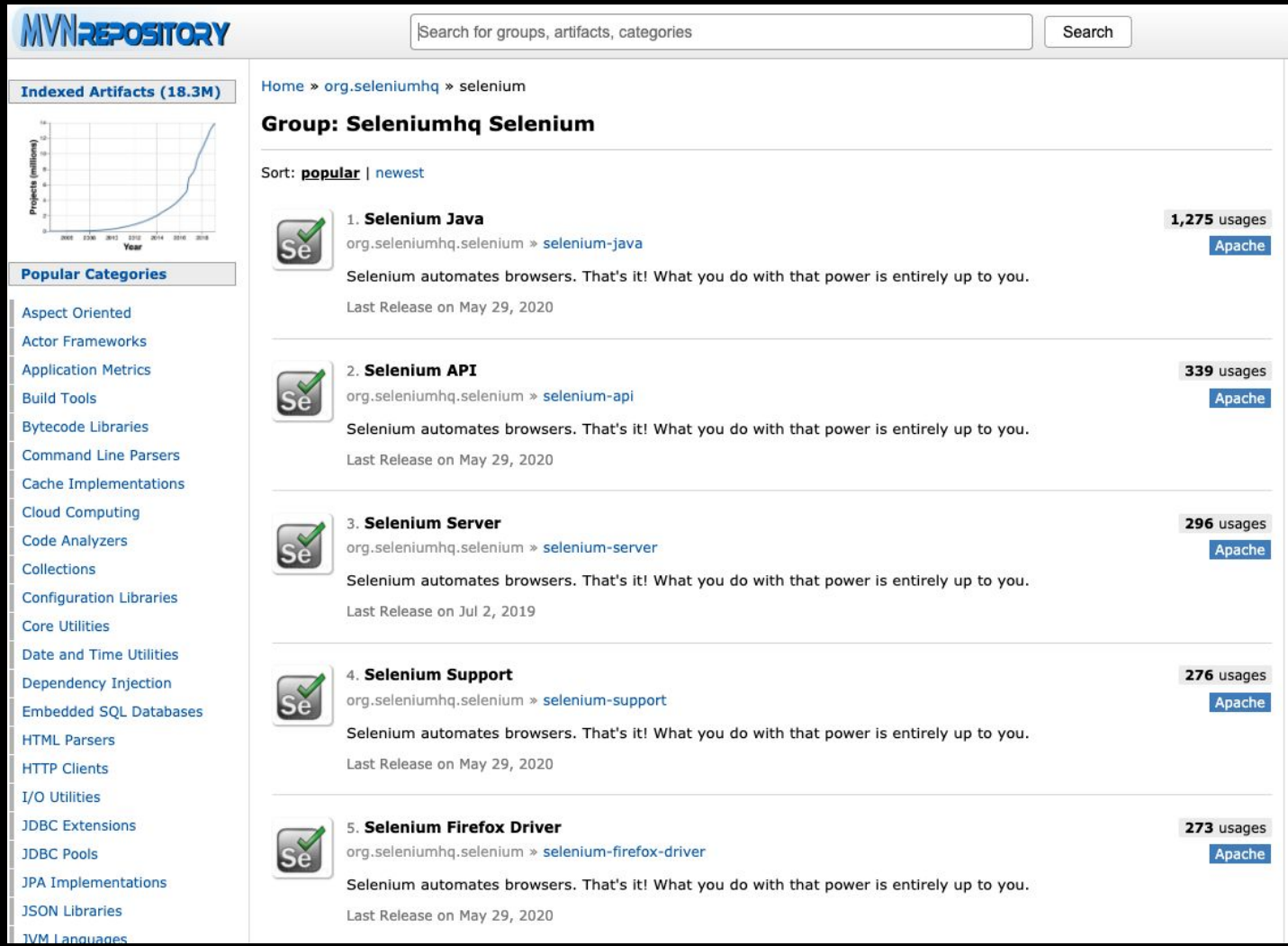
Step 1 - Create a new Maven Project or
use the one you have created during previous lessons

Step 2 - Add Selenium Maven Dependency

Step 3 - Download Third Party Browser Drivers

Add Selenium Maven Dependency

Selenium-java
Selenium-api
Selenium-server
Selenium-firefox-driver
Selenium-chrome-driver



Download Third Party Browser Drivers

Third party drivers and plugins

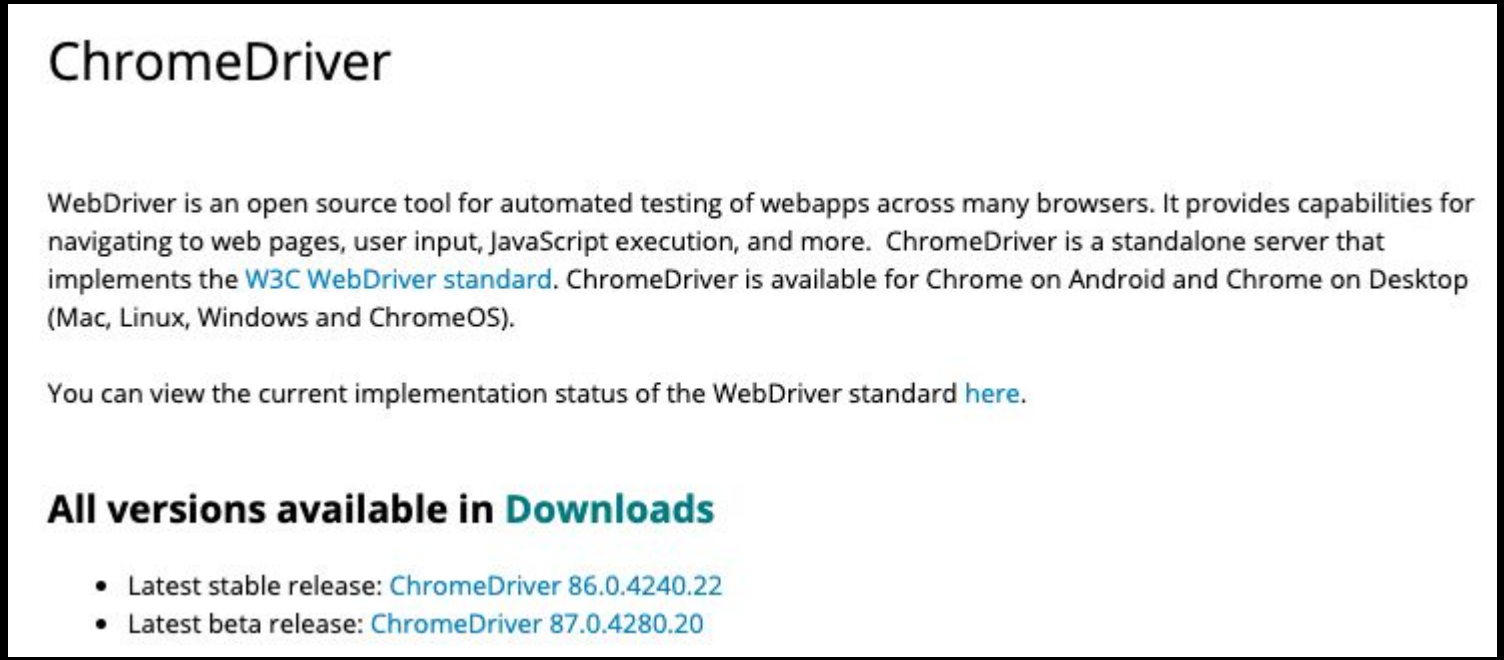
Selenium can be extended through the use of plugins. Here are a number of plugins created and maintained by third parties. For more information on how to create your own plugin or have it listed, consult the docs.

Please note that these plugins are not supported, maintained, hosted, or endorsed by the Selenium project. In addition, be advised that the plugins listed below are not necessarily licensed under the Apache License v.2.0. Some of the plugins are available under another free and open source software license; others are only available under a proprietary license. Any questions about plugins and their license of distribution need to be raised with their respective developer(s).

Browser	Latest	Change log	Issue Tracker
Mozilla GeckoDriver	latest	change log	issue tracker
Google Chrome Driver	latest	change log	issue tracker
Opera	latest	-	issue tracker
Microsoft Edge Driver	latest	-	issue tracker
SafariDriver	Built in	-	issue tracker

https://www.selenium.dev/documentation/en/getting_started_with_webdriver/third_party_drivers_and_plugins/

I Chose Google Chrome Driver



ChromeDriver

WebDriver is an open source tool for automated testing of webapps across many browsers. It provides capabilities for navigating to web pages, user input, JavaScript execution, and more. ChromeDriver is a standalone server that implements the W3C WebDriver standard. ChromeDriver is available for Chrome on Android and Chrome on Desktop (Mac, Linux, Windows and ChromeOS).

You can view the current implementation status of the WebDriver standard here.

All versions available in Downloads

- Latest stable release: ChromeDriver 86.0.4240.22
- Latest beta release: ChromeDriver 87.0.4280.20

My version of Chrome
86.0.4240.111

https://www.selenium.dev/documentation/en/getting_started_with_webdriver/third_party_drivers_and_plugins/

Selenium Locators

Preferred selector order : id > name > css > xpath

HTML

- Html is a standard markup language for creating Web pages. Html elements are the building blocks of HTML pages. HTML tags label pieces of content, such as “heading”, “paragraph”, “table”, and so on.

- HTML elements usually consists of start tag and end tag with content inserted between them.

- For example:
  - <h1> An example for an HTML title </h1>
  - An example for an HTML paragraph

CSS

- CSS is a language that describes the style of an HTML document.
- CSS describes how HTML elements should be displayed.
- A CSS rule-set consists of a selector and a declaration block. For example: selector - h1, declaration – {color:blue;}.

```
<style>

body {
  background-color: lightblue;
}

</style>
```

CSS

- CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

- Some Selectors:

    - Element selector : The element selector selects elements based on the element name. for example: p, h1 etc.

    - ID selector: The id selector uses the id attribute of an HTML element to select a specific element. For example: #id1, #para2 etc.

- A CSS rule-set consists of a selector and a declaration block. For example: selector - h1, declaration – {color:blue;}.

XPath

- XPath stands for XML Path Language, it can be used to navigate through elements and attributes in an XML document.

- XPath uses path expressions to select nodes or node-sets in an XML document

Syntax:

- // - Selects nodes in the document from the current node that match the selection no matter where they are.
- @ - Selects attributes

XPath

- Example

```
<!DOCTYPE html>
<html>
<head>
<title> Page Title</title>
</head>
<body>
<h1 id="h1_id"> This is part of the presentation title</h1>
<p>This is the paragraph.</p>
</body>
</html>
```

The query: //h1[@id='h1_id'] will get the h1 element

Selenium WebDriver

- A Selenium Web driver must be created
- For using Chrome:

```
System.setProperty("webdriver.chrome.driver",projectPath+"/drivers/chromedriver");*
WebDriver driver = new ChromeDriver();
```

- Interaction with the Chrome instance will be made in the code on the driver.

*Note: you need to specify, before instantiating the BrowserDriver, the path to the actual driver that you downloaded following instructions from the selenium website https://www.seleniumhq.org/download/.

Selenium WebDriver

- Navigation using a Selenium WebDriver is very simple, given a defined URL. It can be done in two ways, driver.get(…) or driver.navigate().to(…)

  - *driver.get("https://www.google.com/");*
  - *driver.navigate().to("https://www.google.com/");*

- *The driver.get(…) and driver.navigate().to(…) do exactly the same thing. driver.navigate() supports also driver.navigate().forward() and driver.navigate().backward()*

Finding Web Elements

- Web elements can be defined as each opening and closing tags in the web page. For example: <button>Click Me</button> - a web element.

- Finding a web element and interacting with it can be done in several ways: - ID. - Class. - Name. - Xpath. - Css Selector, etc.

Finding Web Elements

- An example:
  - Assuming that we have the following web page:

```
<html>
    <body>
        <button id= “my_button”> Click Me</button>
    </body>
</html>
```

  - The following lines of code will be used for clicking the button:

```
WebElement button = driver.findElement(By.id(" my_button "));
button.click();
```

Selenium Example

Pull the https://github.com/FabrizioFornari/SPM2020Template repository for the updates related to Selenium.