Software Project Management - Laboratory

Lecture n° 11
A.Y. 2020-2021

Prof. Fabrizio Fornari
fabrizio.fornari@unicam.it

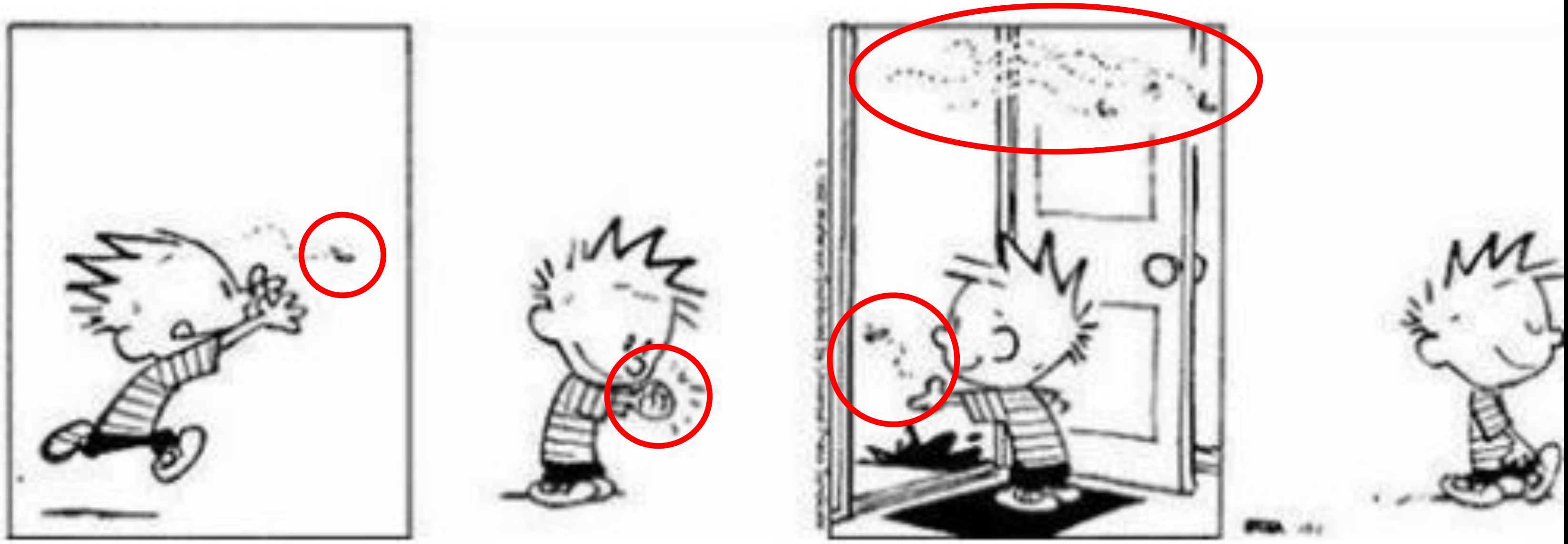
Recap

Types of Testing: Unit Testing, Integration Testing, Regression Testing

Manual vs Automated Testing

Automated Web Testing

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



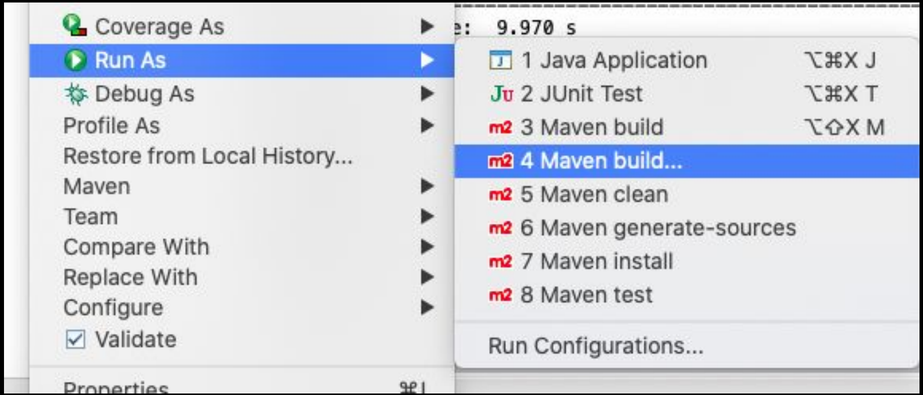
Manual Testing

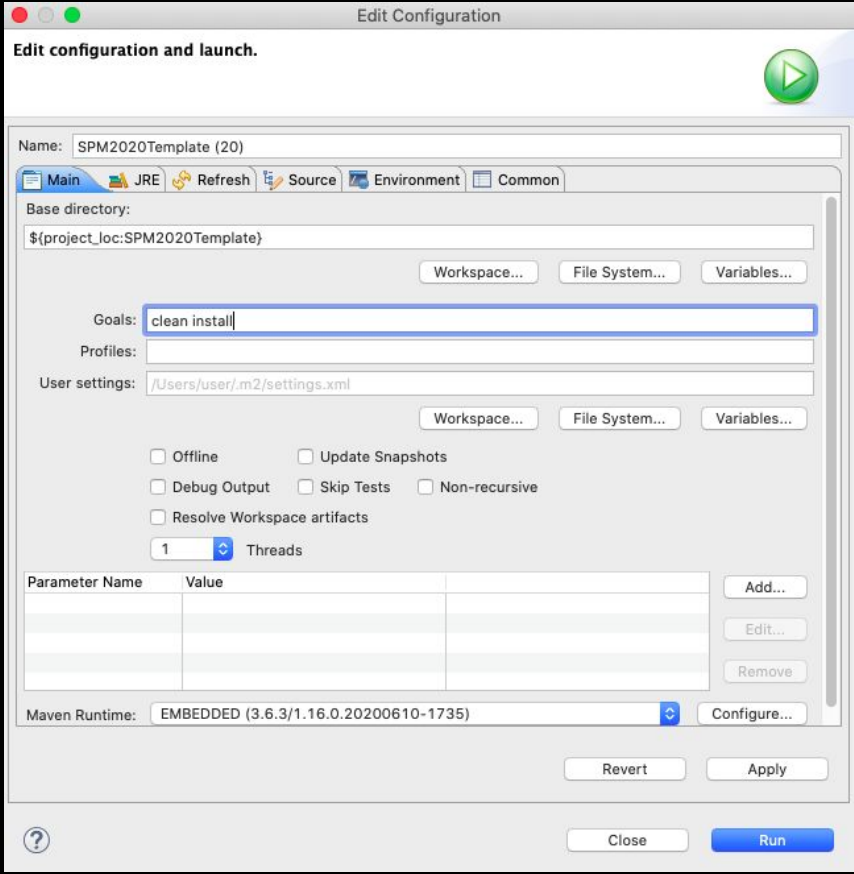
The oldest type of software testing.

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.

Automated Testing with Maven




Edit Configuration

Edit configuration and launch.

Name: SPM2020Template (20)

Main | JRE | Refresh | Source | Environment | Common

Base directory: ${project_loc:SPM2020Template}

Goals: clean install

Profiles:

User settings: /Users/user/.m2/settings.xml

☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts

1 Threads

Parameter Name	Value

Maven Runtime: EMBEDDED (3.6.3/1.16.0.20200610-1735)

Automated Web Testing

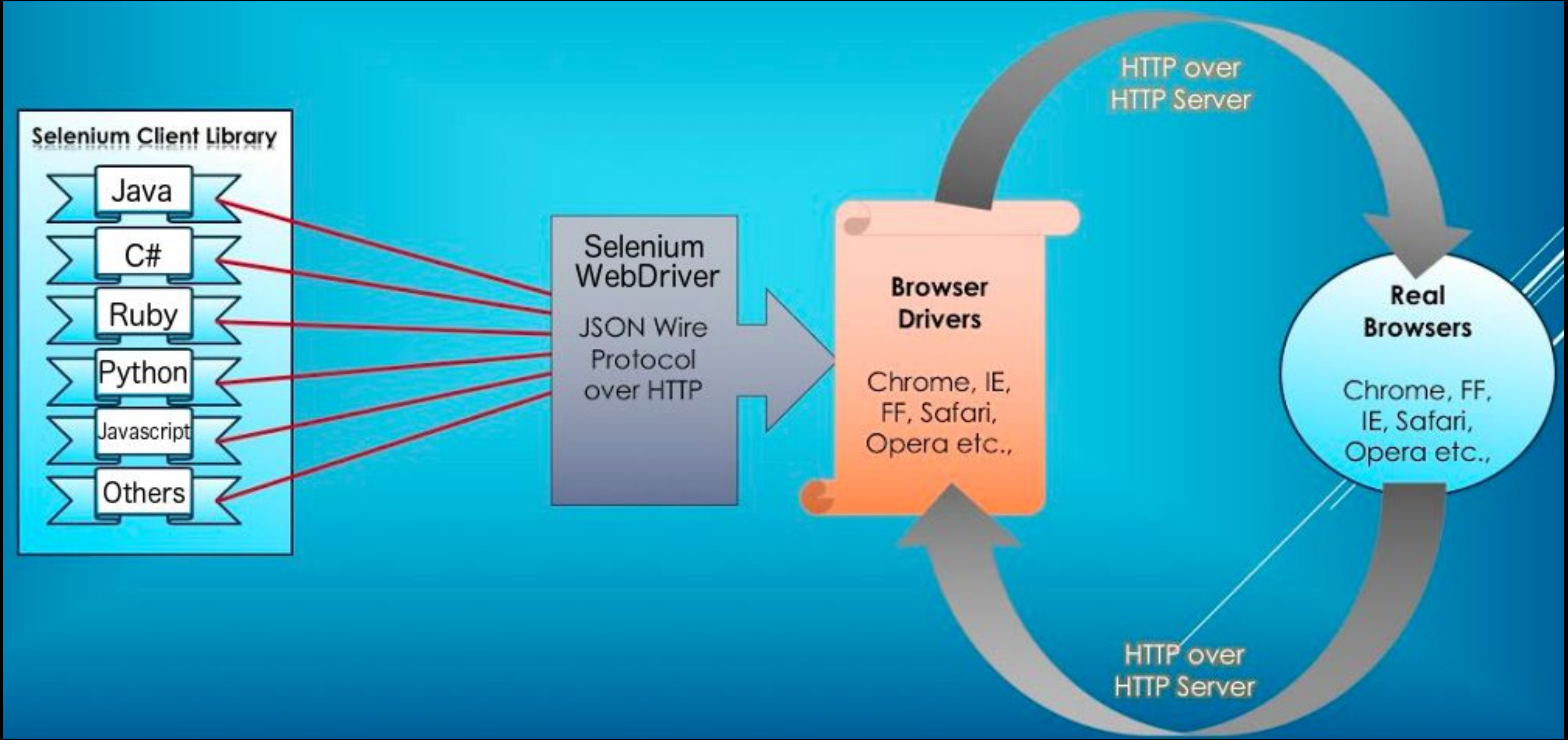
To remotely control browsers so that we can do things like write automated tests for the content they run or tests for the browser UI itself.

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.

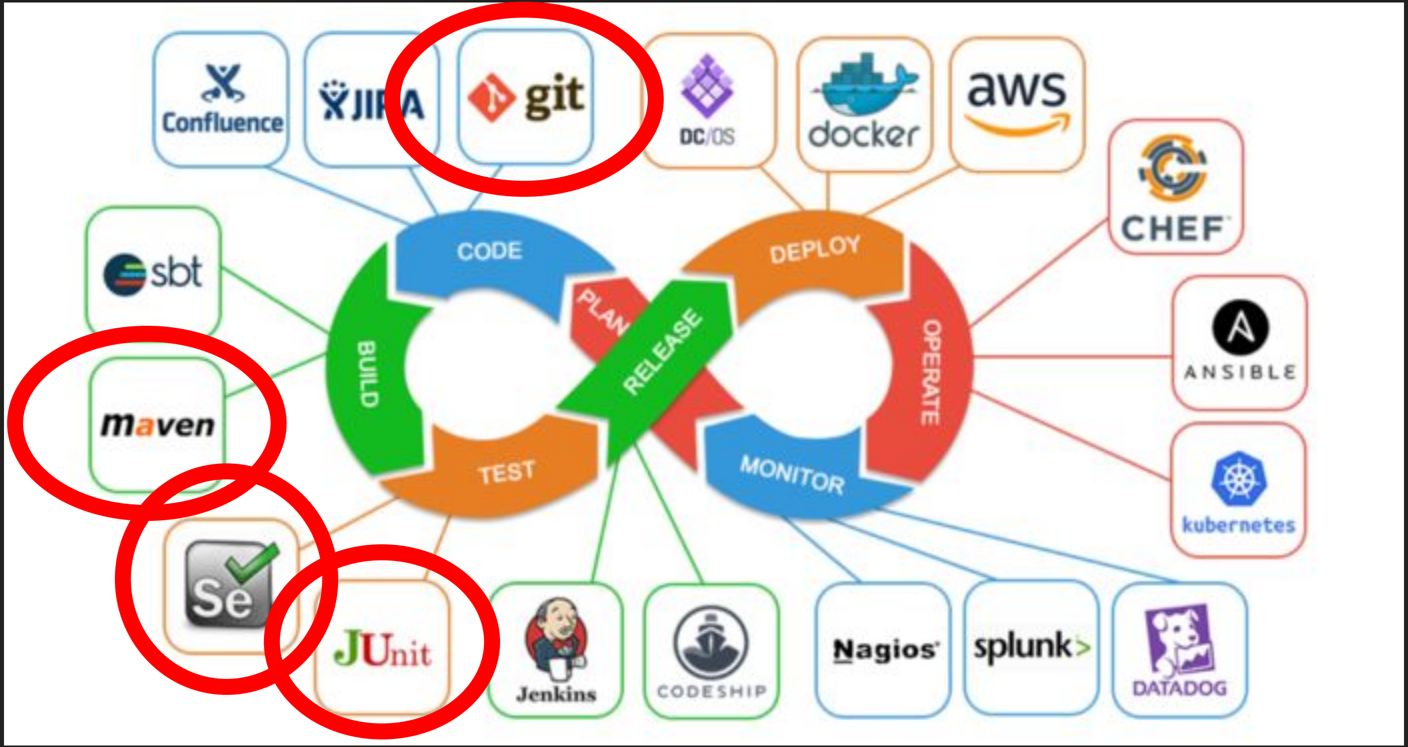
https://w3c.github.io/webdriver/

Selenium is a suite of tools for automating web browsers
https://www.selenium.dev/

Selenium Architecture

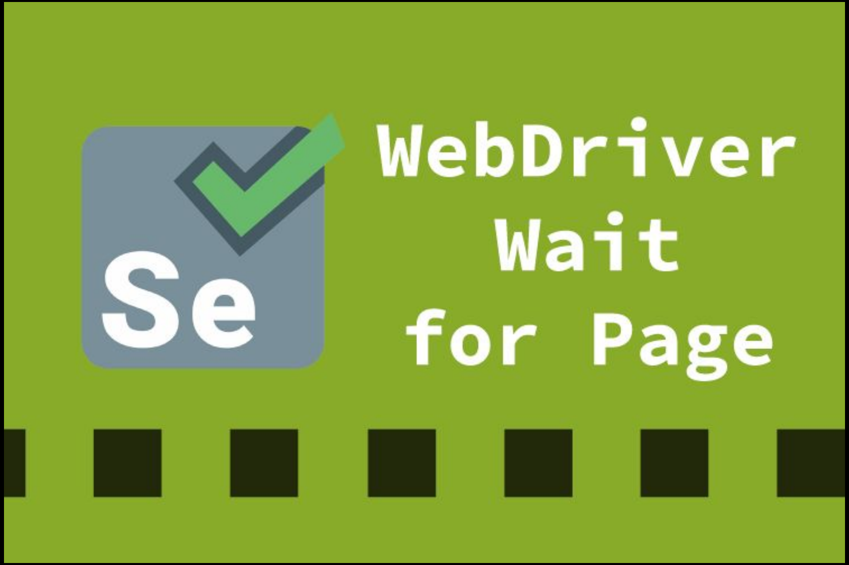


DevOps


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

Selenium

Something more to say about it…



Selenium

To make Selenium tests resilient, we need to make them wait for certain elements to load. Elements that we want to interact with. This is especially true with JavaScript heavy pages.

Implicit waits vs Explicit waits

And the standard advice from the Selenium Core Committers is to use explicit waits.

Note: Explore AdvancedSeleniumTest.java

Implicit Wait

An implicit wait requires setting a default amount of time for Selenium to wait if it can't perform an action immediately, and/or setting static sleeps.

Static sleeps:

```
Thread.sleep(ms);                    // To avoid!
```

It forces your tests to wait a hard-coded amount of time to perform an action

Implicit sleeps:

```
driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);
```

By implicitly waiting, WebDriver polls the DOM for a certain duration when trying to find any element. It means that if the element is not located on the web page within that time frame, it will throw an exception.

Explicit Waits

An explicit wait is code you define to **wait for a certain condition to occur** before proceeding further in the code.

The **condition** is **called with a certain frequency until the timeout of the wait is elapsed**. This means that for as long as the condition returns a falsy value, it will keep trying and waiting.

Since explicit waits allow you to wait for a condition to occur, they make a good fit for synchronising the state between the browser and its DOM, and your WebDriver script.

```
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
...
WebDriver driver = new FirefoxDriver();
driver.get("http://somedomain/url_that_delays_loading");
...
WebDriverWait wait = (new WebDriverWait(driver, NumberOfSeconds));
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("ElementId")));
```

Screenshot

```
public void takeSnapShot(WebDriver webdriver,String fileWithPath){

        TakesScreenshot scrShot =((TakesScreenshot)webdriver);

        //Call getScreenshotAs method to create image file
        File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);

        //Move image file to new destination
        File DestFile=new File(fileWithPath);
        Files.copy(SrcFile.toPath(), DestFile.toPath(),StandardCopyOption.REPLACE_EXISTING);

}
```

How to check the status of HTTP Request?

We can use Rest Assured

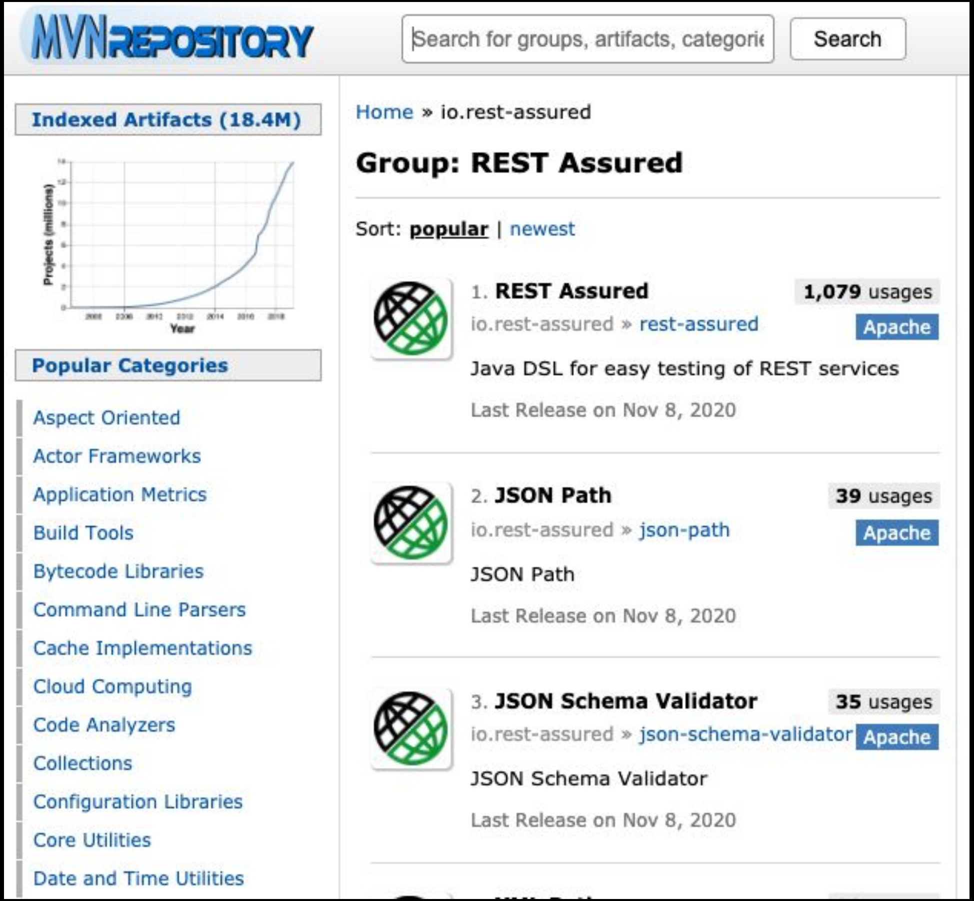
REST Assured is a Java DSL for simplifying testing of REST based services built on top of HTTP Builder.

It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests.

https://rest-assured.io/

https://github.com/rest-assured/rest-assured/wiki/Usage

Rest Assured



MVNREPOSITORY

Search for groups, artifacts, categories | Search

Indexed Artifacts (18.4M)

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities

Home » io.rest-assured

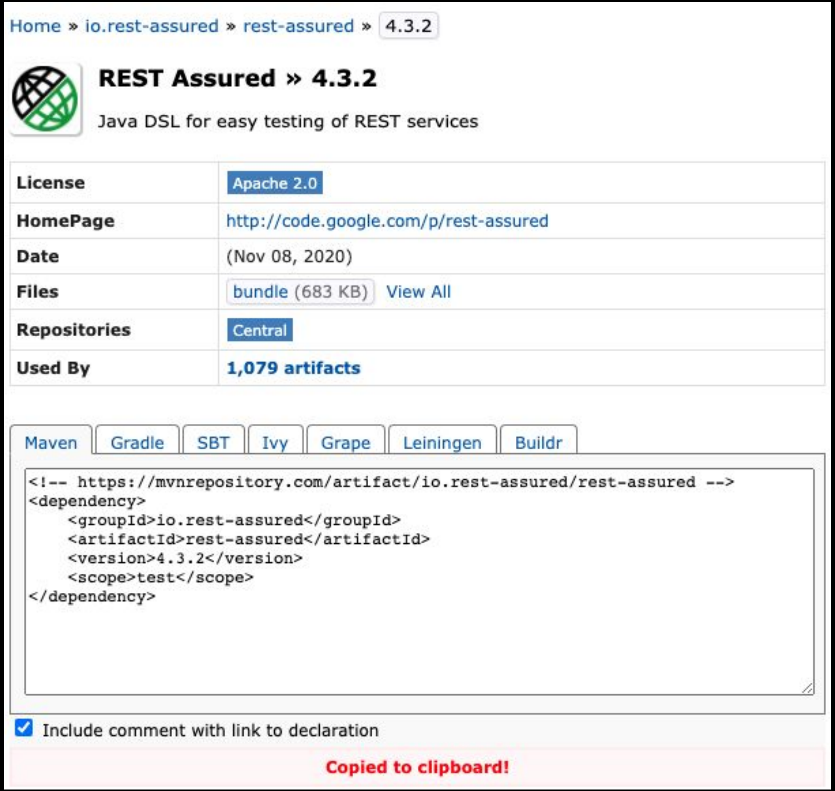
Group: REST Assured

Sort: **popular** | newest

1. **REST Assured** — **1,079** usages
   io.rest-assured » rest-assured — Apache
   Java DSL for easy testing of REST services
   Last Release on Nov 8, 2020

2. **JSON Path** — **39** usages
   io.rest-assured » json-path — Apache
   JSON Path
   Last Release on Nov 8, 2020

3. **JSON Schema Validator** — **35** usages
   io.rest-assured » json-schema-validator — Apache
   JSON Schema Validator
   Last Release on Nov 8, 2020



Home » io.rest-assured » rest-assured » 4.3.2

REST Assured » 4.3.2

Java DSL for easy testing of REST services

License	Apache 2.0
HomePage	http://code.google.com/p/rest-assured
Date	(Nov 08, 2020)
Files	bundle (683 KB) View All
Repositories	Central
Used By	**1,079 artifacts**

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>4.3.2</version>
    <scope>test</scope>
</dependency>
```

[✓] Include comment with link to declaration

Copied to clipboard!

What about complex tests…?

Do we have to write them entirely from scratch?

Fortunately No!

Selenium IDE

Download it from:

https://www.seleniumhq.org/selenium-ide/

and let us see what we can do with it...

However we cannot export tests in a format that we can use for writing tests in our preferred programming language

Katalon Recorder

Katalon Automation Recorder it is an automation recorder that helps to export Selenium WebDriver code.

Download the extension for the browser you want to use

Explore testGuestSearchMetadataTypeChoregraphy method

Katalon https://www.katalon.com/

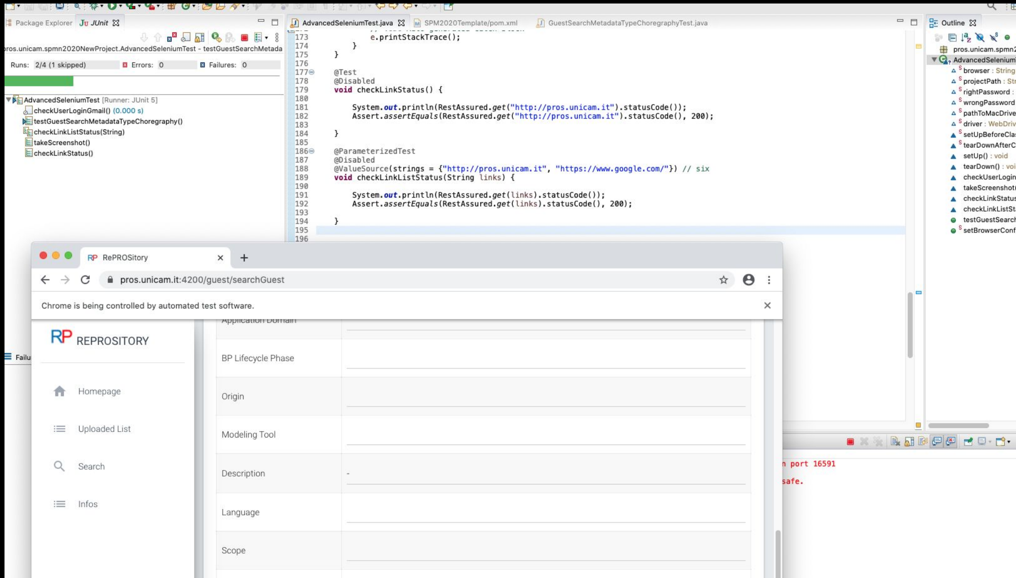
Try to record the some tests

Try to find any difference

Do we really need a browser...?

Or better...do we really need a graphical interface?

Every time we run a test, an instance of a browser is created and the graphical user interface of the chosen browser appears...do we really need it?



```
pros.unicam.spmn2020NewProject.AdvancedSeleniumTest - testGuestSearchMetada
Runs: 2/4 (1 skipped)   Errors: 0   Failures: 0

AdvancedSeleniumTest [Runner: JUnit 5]
    checkUserLoginGmail() (0.000 s)
    testGuestSearchMetadataTypeChoregraphy()
    checkLinkListStatus(String)
    takeScreenshot()
    checkLinkStatus()
```

```java
            e.printStackTrace();
        }
    }

    @Test
    @Disabled
    void checkLinkStatus() {

        System.out.println(RestAssured.get("http://pros.unicam.it").statusCode());
        Assert.assertEquals(RestAssured.get("http://pros.unicam.it").statusCode(), 200);

    }

    @ParameterizedTest
    @Disabled
    @ValueSource(strings = {"http://pros.unicam.it", "https://www.google.com/"}) // six
    void checkLinkListStatus(String links) {

        System.out.println(RestAssured.get(links).statusCode());
        Assert.assertEquals(RestAssured.get(links).statusCode(), 200);

    }
```

Headless Browser…

Headless Browser…

- It is a browser without graphical interface
- What is it for?

Headless Browser…

It is a browser without graphical interface

Headless browsers are commonly used for:

- Website and application testing
- JavaScript library testing
- JavaScript simulation and interactions
- Running one or more automated UI tests in the background

Headless Browser…

In a headless testing environment, you can write and execute scripts to:

- Test basic and alternative flows
- Simulate clicks on links and buttons
- Automate form filling and submission
- Test SSL performance
- Experiment with various server loads
- Get reports on page response times
- Scrape useful website code
- Take screenshots of results

Testing these use cases provides you with a solid overview of how a site’s UI performs and gives you essential information for making changes before deployment.

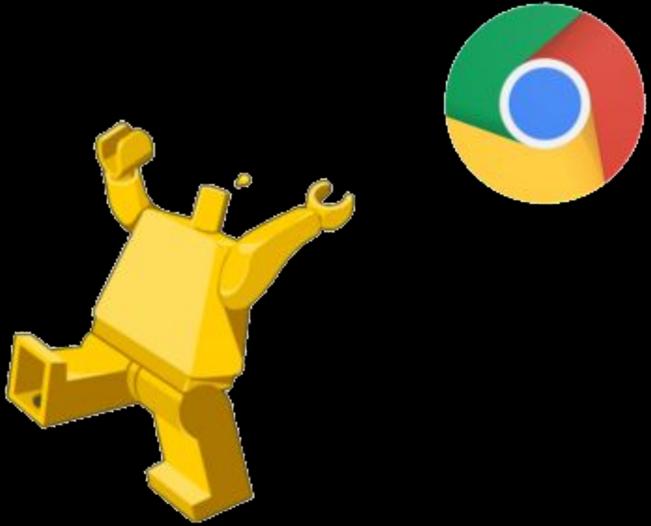
Which Headless Browser…?

Can you name one Headless Browser?

Which Headless Browser…?

- Firefox Headless Mode
- Headless Chrome
- PhantomJS
- Zombie JS
- HtmlUnit
- Splash

Headless Chrome



Headless Chrome

The biggest downside is that you need to be able to install Chrome. You don't need a UI, but installing software is not always possible.

Chrome Driver also requires an executable to be downloaded.

I keep the executable in the same directory as the project (or in a binary repository and copy it to the workspace.)

It still requires Chrome itself to be installed.

Html Unit

https://htmlunit.sourceforge.io/

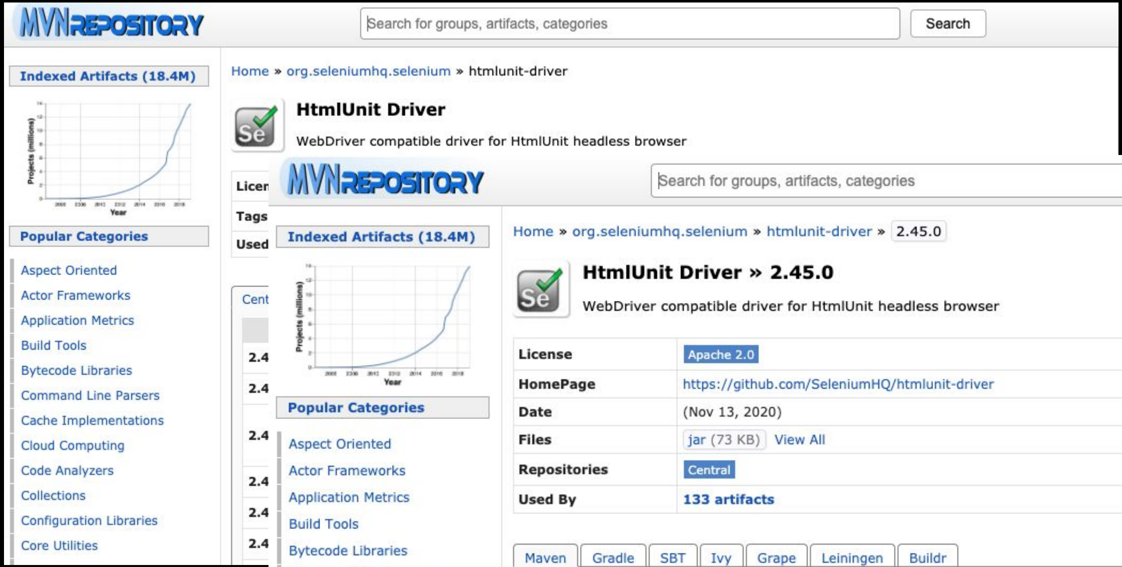
Html Unit

In the past, Selenium came with a built in headless driver called HtmlUnitDriver.

While this driver is still supported, it is now a separate dependency and, unsurprisingly, uses the Html Unit framework.

Prior to Single Page Applications and largely AJAX based pages, this driver was an excellent choice. You have the ability to choose whether to run the page JavaScript, it runs in memory and is very fast. It’s still a good choice for web pages with a good amount of HTML data on them.

HtmlUnit Driver



Search for groups, artifacts, categories | Search

Indexed Artifacts (18.4M)

Home » org.seleniumhq.selenium » htmlunit-driver

HtmlUnit Driver

WebDriver compatible driver for HtmlUnit headless browser

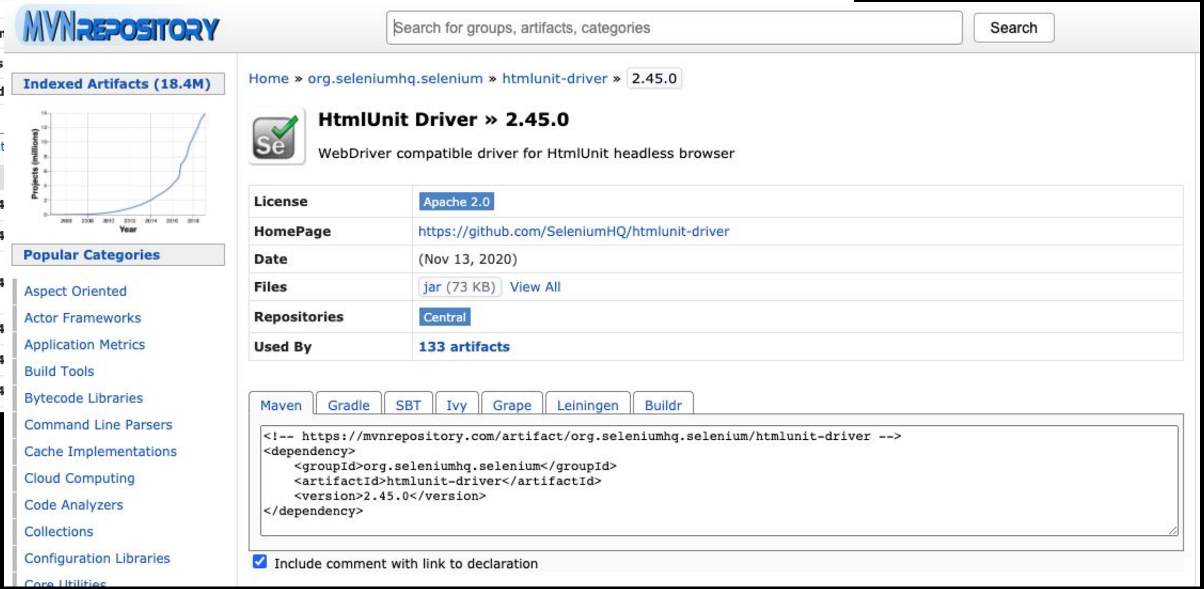
Licen
Tags
Used

Cent

2.4
2.4
2.4
2.4
2.4
2.4

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities



Search for groups, artifacts, categories | Search

Indexed Artifacts (18.4M)

Home » org.seleniumhq.selenium » htmlunit-driver » 2.45.0

HtmlUnit Driver » 2.45.0

WebDriver compatible driver for HtmlUnit headless browser

License	Apache 2.0
HomePage	https://github.com/SeleniumHQ/htmlunit-driver
Date	(Nov 13, 2020)
Files	jar (73 KB) View All
Repositories	Central
Used By	**133 artifacts**

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/htmlunit-driver -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>htmlunit-driver</artifactId>
    <version>2.45.0</version>
</dependency>
```

☑ Include comment with link to declaration

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries

https://htmlunit.sourceforge.io/gettingStarted.html

Phantom JS

PhantomJS is a headless web browser scriptable with JavaScript.
It runs on Windows, macOS, Linux, and FreeBSD.

https://phantomjs.org/
https://github.com/ariya/phantomjs/

Project Suspended - https://github.com/ariya/phantomjs/issues/15344
https://groups.google.com/g/phantomjs/c/9aI5d-LDuNE?pli=1

Keep an eye on:

https://github.com/FabrizioFornari/SPM2020Template.git