### **Chapter 1 – A Primer on Selenium (Extended Notes)**

* **Selenium Overview**
  + Open-source suite for browser automation.
  + Composed of three main components: Selenium WebDriver, Selenium Grid, Selenium IDE.
  + Main purpose: automated end-to-end testing of web applications.
  + Other uses: automating web-based administrative tasks, web scraping.

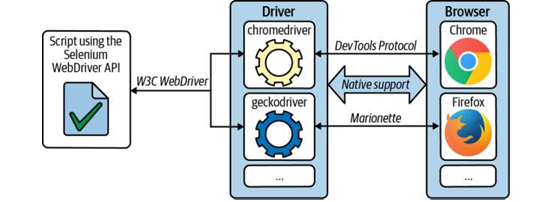
#### **1. History of Selenium**

* Created in 2004 by Jason Huggins and Paul Hammant at ThoughtWorks.
* Selenium Core: JavaScript library executing "Selenese" commands.
* Selenium RC (Remote Control): client-server model, injected Selenium Core into browser.
* RC limitations: could not handle uploads, downloads, popups, dialogs; performance overhead.
* 2007: Simon Stewart created WebDriver using native browser automation.
* 2009: WebDriver merged with RC → Selenium 2.
* Selenium 3 released in 2016.
* Selenium 4 released in 2021, based on W3C WebDriver standard.
* Selenium RC/Core now deprecated (called Selenium 1).

#### 

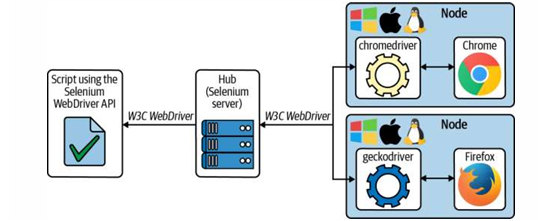
#### 

#### **2. Selenium WebDriver**



* Core library for browser automation.
* Cross-platform API supporting: Java, Python, JavaScript, Ruby, C#.
* Requires **browser-specific drivers** (e.g., chromedriver, geckodriver, msedgedriver).
* Early protocol: JSON Wire Protocol (Selenium 2 and 3).
* Current protocol: W3C WebDriver (Selenium 4).
* Chrome/Edge/Opera use DevTools Protocol; Firefox uses Marionette Protocol.

#### **3. Selenium Grid**



* Introduced in 2008 by Philippe Hanrigou.
* Enables distributed and parallel test execution across OS and browser combinations.
* Hub-Node architecture: Hub manages nodes; nodes provide browsers.
* Limitation: hub bottleneck.
* Selenium 4 Grid is **fully distributed** with load balancing.

#### **4. Selenium IDE**

* Created in 2006 by Shinya Kasatani.
* Record & Playback automation tool.
* Initially Firefox-only, discontinued after Firefox 55.
* Rewritten as cross-browser extension (Chrome, Firefox, Edge).
* Records actions (command, target, value) and plays them back.

#### **5. Language Bindings**

* Selenium provides official client libraries for:  
  + Java
  + Python
  + JavaScript (Node.js)
  + Ruby
  + C# (.NET)
* Bindings allow Selenium to be used in different ecosystems and integrate with various test frameworks.

#### **6. Driver Managers**

* Traditionally, testers had to manually download and manage drivers (e.g., chromedriver, geckodriver).
* Problems: version mismatch, manual updates.
* **Driver Manager tools** automate this process. Examples:  
  + **WebDriverManager** (Java ecosystem).
  + **webdriver-manager** (Python package).
* Benefits:  
  + Automatically downloads correct driver version.
  + Simplifies cross-platform testing.
  + Reduces setup/configuration time.

#### **7. Locator Tools**

* Locators are used to identify web elements on a page.
* Common locator strategies:  
  + ID
  + Name
  + Class Name
  + Tag Name
  + Link Text / Partial Link Text
  + CSS Selector
  + XPath
* Best practice: prefer **ID and Name** for stability, fallback to CSS or XPath when necessary.

#### **8. Frameworks**

* Frameworks provide structure, reusability, and maintainability to Selenium tests.
* Common frameworks in Python:  
  + **unittest** – built-in testing framework.
  + **pytest** – popular with rich plugins and fixtures.
  + **behave** – BDD (Behavior Driven Development).
  + **Robot Framework** – keyword-driven testing.
* In Java ecosystem: JUnit, TestNG, Cucumber.
* Best practices:  
  + Use **Page Object Model (POM)** for maintainable code.
  + Integrate with CI/CD tools (Jenkins, GitHub Actions).

#### **9. Software Testing Fundamentals – Levels of Testing**

* **Unit Testing** – tests individual components in isolation.
* **Integration Testing** – verifies interaction between modules.
* **System Testing** – validates the entire system as a whole.
* **Acceptance Testing** – ensures the system meets business requirements (often UAT).

#### **10. Software Testing Fundamentals – Types of Testing**

* **Functional Testing** – verifies what the system does.
* **Non-Functional Testing** – verifies how the system behaves (performance, security, usability).
* **Smoke Testing** – quick checks to confirm basic functionality.
* **Regression Testing** – ensures existing features work after changes.
* **Sanity Testing** – checks specific functionality after small changes.
* **Performance Testing** – evaluates system speed and scalability.
* **Security Testing** – checks vulnerabilities and data protection.

#### **11. Test Automation Tools**

* Selenium is part of a larger automation ecosystem.
* Popular automation tools:  
  + **Selenium WebDriver** – browser automation.
  + **Playwright** – newer alternative with modern API.
  + **Cypress** – fast testing for front-end JavaScript apps.
  + **Appium** – mobile automation (iOS, Android).
  + **JMeter** – performance testing.
  + **Postman** – API testing.
  + **Robot Framework** – generic automation framework.

✅ **Key Takeaways (Extended)**

* Selenium supports multiple languages, but driver management is crucial for smooth execution.
* Locators are the backbone of stable test scripts.
* Frameworks add structure, scalability, and reporting to automation projects.
* Software testing fundamentals (levels and types) ensure comprehensive test coverage.
* Selenium co-exists with other test automation tools, each serving a specific niche.