**Selenium Evolution & Frameworks**

**1. Selenium Core (2004) – ThoughtWorks**

* **Purpose:** Automate Web UI Testing.
* **Working:** Operated by injecting JavaScript into the browser to interact with the web page.
* **Limitations:**
  + Could not handle web applications heavily built with JavaScript.
  + Security restrictions in browsers limited direct interaction.

**2. Selenium RC (Remote Control) (2006)**

* **Purpose:** Introduced a server to bypass same-origin policy and allowed automation in multiple languages.
* **Features:**
  + Supported multiple programming languages: Java, C#, Python, Ruby, PHP, etc.
  + Cross-browser automation support.
* **Disadvantage:**
  + Slow execution due to the server acting as a middle layer between the test script and browser.

**3. Selenium IDE (2006)**

* **Purpose:** Record-and-playback tool for testers without coding skills.
* **Features:**
  + Easy to use for beginners.
  + Suitable for quick test cases.
* **Limitations:**
  + No programming logic support.
  + Not scalable for large applications.
  + Lacked handling for large datasets.

**4. Selenium WebDriver (2009)**

* **Introduced by:** Simon Stewart.
* **Purpose:** Directly controls the browser without JavaScript injection.
* **Advantages:**
  + Faster execution than RC.
  + Full support for programming languages.
  + Supports dynamic web applications.
* **Impact:** Became the **official successor** of Selenium RC.

**5. Selenium Grid (2008, improved in 2011)**

* **Purpose:** Execute tests in parallel on multiple machines and browsers.
* **Advantages:**
  + Reduces execution time.
  + Supports distributed test execution.
* **Progression:**
  + Selenium 3 (2016): Integrated mobile testing with Appium, better browser driver management.
  + Selenium 4 (2021): Enhanced parallel execution, cloud integrations.

**6. Selenium 4 (2021 – Latest)**

* **Improvements:**
  + Modern architecture with W3C WebDriver protocol.
  + Cloud integration with **Sauce Labs**, **BrowserStack**.
  + API integration.
  + AI-powered element locators for dynamic elements.
* **Advantages over previous versions:**
  + More stable element handling.
  + Better debugging with Chrome DevTools Protocol (CDP).

**Selenium Installation Requirements**

* **Selenium WebDriver:**  
  <https://www.selenium.dev/downloads/>
* **ChromeDriver:**  
  <https://googlechromelabs.github.io/chrome-for-testing/>

**Selenium with Different Frameworks**

**1. Data-Driven Framework**

* **Definition:** Uses external data sources (Excel, CSV, Database) to drive test cases.
* **Example:**
  + Login test runs with multiple username/password combinations stored in Excel.

**2. Keyword-Driven Framework**

* **Definition:** Uses predefined keywords (e.g., Click, EnterText) to describe actions in test scripts.
* **Example Table:**

| **Action** | **Locator** | **Value** |
| --- | --- | --- |
| OpenBrowser | Chrome |  |
| Navigate | URL | google.com |
| Type | name=q | Selenium |
| Click | name=btnK |  |

**3. Hybrid Framework**

* **Definition:** Combination of Data-Driven and Keyword-Driven approaches.
* **Example:** Test steps written with keywords, but input values fetched from Excel.

**4. BDD (Behavior Driven Development)**

* **Definition:** Uses natural language (Gherkin syntax) for writing test cases.
* **Example:**

gherkin

CopyEdit

Feature: Login Functionality

Scenario: Successful login

Given user is on login page

When user enters valid username and password

Then user should be redirected to dashboard

**5. Modular Testing Framework**

* **Definition:** Divides application into small independent modules; tests are created for each module.
* **Example:** Separate test scripts for Login, Search, Checkout.

**6. Page Object Model (POM)**

* **Definition:** Creates a separate class for each web page containing locators and methods.
* **Example (Java):**

java

CopyEdit

public class LoginPage {

WebDriver driver;

By username = By.id("user");

By password = By.id("pass");

By loginBtn = By.id("login");

public LoginPage(WebDriver driver) {

this.driver = driver;

}

public void login(String user, String pass) {

driver.findElement(username).sendKeys(user);

driver.findElement(password).sendKeys(pass);

driver.findElement(loginBtn).click();

}

}

**Hybrid + BDD + POM (Advanced Framework Example)**

* **Hybrid:** Data-driven + keyword-driven combination for flexible test execution.
* **BDD:** Gherkin scenarios describe tests in plain English.
* **POM:** Manages locators & methods separately for maintainability.

**Example Workflow:**

1. **Feature File (BDD):**

gherkin

CopyEdit

Scenario: Login with valid credentials

Given user is on login page

When user enters "admin" and "password123"

Then dashboard is displayed

1. **Step Definitions (Java):**

java

CopyEdit

@When("user enters {string} and {string}")

public void user\_enters\_credentials(String user, String pass) {

loginPage.login(user, pass);

}

1. **Page Object Class (POM):**  
   *(same as above LoginPage example)*
2. **Data Source (Hybrid):** Excel file provides multiple sets of credentials.