

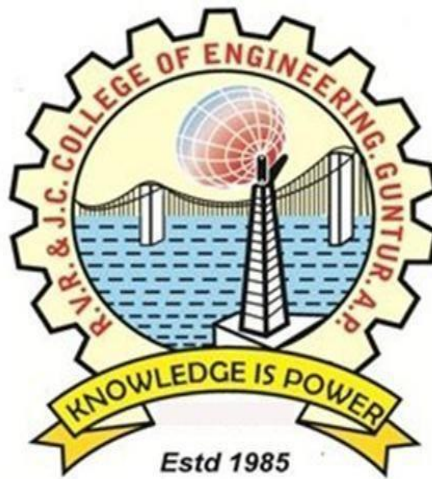
Software Testing Internship

Submitted in partial fulfillment of requirements to CSE (Data Science)

Summer Internship (CD-451) IV/IV B. TECH CSE(DS) (VII Semester)

Submitted by

Sivalasetty Venkata Hari Ratna Lahari(Y22CD163)



August 2025

R. V. R & J.C. College of Engineering (Autonomous)

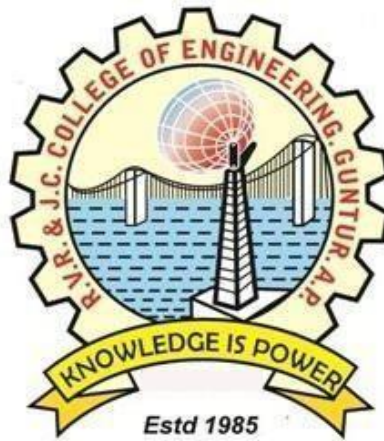
(NAAC A+ Grade) (Approved by A.I.C.T.E.)

(Affiliated to Acharya Nagarjuna University)

Chandramoulipuram : : Chowdavaram

Guntur – 522019

R. V. R & J.C. COLLEGE OF ENGINEERING
DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)



CERTIFICATE

This is to certify that this internship report “**Software Testing Internship**” is the Bonafide work of “**Sivalasetty Venkata Hari Ratna Lahari(Y22CD163)** ” who has carried out the work under my supervision and submitted in partial fulfillment for the award of **Summer Internship (CD-451)** during the year 2025-2026.

Dr.G. Ramanjaiah
Internship Incharge

Dr.M.V.P. Chandra Sekhara Rao
Prof. & HOD, CSE(DS)

ACKNOWLEDGEMENT

I would like to express our sincere gratitude to these dignitaries, who are with us in the journey of my Summer Internship **“Software Testing Internship.”**

First and foremost, we extend our heartfelt thanks to **Dr. Kolla Srinivas**, principal of **R. V. R. & J.C. College of Engineering**, Guntur, for providing with such overwhelming environment to undertake this internship.

I am utmost grateful to **Dr.M.V.P. Chandra Sekhara Rao**, Head of the Department of Computer Science & Engineering (Data Science) for paving a path for me and assisting me to meet the requirements that are needed to complete this internship.

I extend my gratitude to the Incharge **Dr.G. Ramajaiah** for his ecstatic guidance & feedback throughout the internship. His constant support and constructive criticism have helped me in completing the internship.

I would also like to extend my gratitude to **Prof. Sangharatna Godbole** and **Prof. Pisipati Radha Krishna (NIT Warangal)** for their guidance and feedback throughout the whole internship journey which helped us in completing internship. I would also like to express our sincere thanks to my friends and family and family for their moral support throughout our journey.

Sivalasetty Venkata Hari Ratna Lahari

(Y22CD163)

SUMMER INTERNSHIP CERTIFICATE



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
(An Institute of National Importance)

Address: NIT Campus, Warangal, Telangana State, India, Pin Code – 506004. website:
www.nitw.ac.in/departments/cse/, e-mail: prkrishna@nitw.ac.in, Phone: +91 870 246 2703

Dr. P. Radha Krishna
Professor

June 4, 2025

Internship Completion Letter

This is to certify that Ms. Venkata Hari Ratna Lahari Sivalsetty, a student of B.Tech in Computer Science and Engineering with a specialization in Data Science at R.V.R. & J.C. College of Engineering, Guntur, has successfully completed a sixweek internship at the Department of Computer Science and Engineering, NIT Warangal for 6 weeks under the guidance of Prof. Sangharatna Godbole and Prof. P. Radha Krishna.

During the internship, he/she demonstrated a strong understanding of software testing concepts and showcased excellent problem-solving skills. His/her ability to identify and address bugs effectively, even in a fast-paced environment, was commendable. We are confident in his/her potential and wish him/her all the very best in his/her future endeavours.

(P. Radha Krishna)

Abstract

This report documents the comprehensive summer internship on **Software Testing** undertaken at the **National Institute of Technology (NIT) Warangal**. The primary objective was to gain hands-on experience and conduct a comparative analysis of two distinct yet significant testing paradigms: the industry-standard **Selenium** automation framework and the research-oriented **TrustInn** verification toolkit.

The internship involved an in-depth exploration of the **Selenium** suite, including **Selenium IDE** for record-and-playback prototyping, **Selenium WebDriver** for programmatic browser automation, and **Selenium Grid** for parallel, cross-browser testing. Practical projects were executed to automate interactions with major web platforms, demonstrating Selenium's capabilities in enhancing test coverage, execution speed, and regression testing.

In parallel, the internship provided a unique opportunity to work with **TrustInn**, a tool developed at NIT Warangal for formal verification and advanced program analysis. This involved configuring the tool within Oracle VirtualBox and conducting tests using its integrated components like **CBMC**, **SC-MCC-CBMC**, **MCDC-TX**, and **Verisol** for analyzing C programs and Solidity smart contracts.

The comparative study concludes that while **Selenium** excels in functional and UI automation for web applications, **TrustInn** serves as a powerful complement for static analysis and formal verification. This internship not only provided practical proficiency in industry-standard automation tools but also exposed the critical role of innovative research tools.

Table of Contents

Title	Pg. No.
Chapter 1: Introduction	1
1.1 Why Automated Testing over Manual Testing?	1
1.2 Automated Testing Process	1-2
1.3 Popular Automation tools	2
Chapter 2: Introduction to Selenium	3
2.1 Basics of Selenium	3
2.2 Selenium types	4
Chapter 3: Selenium IDE	5
3.1 Introduction to Selenium IDE	5-7
3.2 Selenium Commands	7-9
3.3 Limitations	9-1
3.4 Sample Project	11-13
Chapter 4: Selenium WebDriver	14
4.1 Introduction	14
4.2 Installation	15-16
4.3 Commands	17-19
4.4 Limitations in Selenium WebDriver	19-21
4.5 Sample projects	21-23
Chapter 5: Selenium Grid	24
5.1 Introduction to Selenium Grid	24-25
5.2 Components	25-26

5.3 Installation of Selenium Grid	26-30
Chapter 6: TrustInn	31
6.1 Introduction	31
6.2 Issues faced while opening TrustInn	31-34
6.3 Testing tools	35
6.3.1 Testing the CBMC tool (using C files)	35-37
6.3.2 Testing the SC-MCC-CBMC using C files	37
6.3.3 Testing of MCDC-TX tool using C files	38-39
6.3.4 Testing of Verisol tool using Solidity	39-40
6.3.5 Testing KLEEMA tool	40-41
6.3.6 Testing with Static-Analysis tool	41-42
7. Conclusion & References	43-44

List of Figures

Fig. No.	Figure Name	Pg. No.
2.1	Features of Selenium	3
2.2	Types of Selenium	4
3.1	Components of Selenium IDE	6
3.3.1	Selenium IDE Test Execution log with Window Location Failure	9
3.3.2	Selenium IDE – Empty Command list after Recording	10
3.3.3	Selenium IDE Command Execution with Mouse Over Failure	11
3.4.1	Selenium IDE Test Script for Automating Amazon Search	11
3.4.2	Exporting Test Script to Different Programming Languages	12
3.4.3	ExTest.java	13
4.1	Working of Selenium WebDriver	14
4.4	StateElementReferenceException	21
4.5.1	Automating ChatGPT chat and Getting Student Results	22
4.5.2	Automating Google Search & Site Opening	23
5.1	Working of Selenium Grid	24
5.3.1	Verification of Java Version	26
5.3.2	Selenium-server-usage-instructions-4.27.0	27
5.3.3	Selenium Hub Startup – CLI Output	27
5.3.4	Verification of Port Address	28
5.3.5	Selenium Node Connecting to Hub – CLI Output	28
5.3.6	Verification of Navigation Selenium Grid	29
5.3.7	Python Script for Parallel Selenium Remote WebDriver	29
5.3.8	Working with Selenium Grid	30
6.2.1	Aborted Error	31
6.2.2	Not Listed	32

6.2.3	Bug Clearance	32
6.2.4	Resolution Approach	33
6.2.5	VirtualBox VM Settings – General and System Configuration	34
6.3.1.1	CBMC Tool (file: age.c)	35
6.3.1.2	CBMC Verification Output	35
6.3.1.3	CBMC Tool (filename: m34CTLAI-B2.c)	36
6.3.1.4	CBMC Output Verification	36
6.3.1.5	CBMC (file: m34CTLAI-B3.c)	37
6.3.1.6	Output Verification	37
6.3.2	SC-MCC-CBMC Tool	38
6.3.3	MCDC-TX Tool	39
6.3.4	Verisol Tool	39
6.3.5.1	KLEEMA Tool	40
6.3.5.2	Output Verification	41
6.3.6.1	Static Analysis Tool	41
6.3.6.2	Output Validation	42

List of Tables

Table. No	Table Name	Pg. No.
3.2	Commands of Selenium IDE	9
4.3	Available Functions and Commands in Selenium WebDriver	17-19

Chapter 1: Introduction

Manual testing is a technique to test the software that is carried out using the functions and features of an application. In manual software testing, a tester tests the software by following predefined test cases. In this testing, testers make test cases for the codes, test the software, and give the final report about that software. Manual testing is time-consuming because it is done by humans, and there is a chance of human errors.

1.1 Why Automated Testing over Manual Testing?

Automated Testing was mostly preferred because of its efficiency, accuracy and alignment with modern development practices. Automated tests run significantly faster and reduce human error making them more reliable for repeated executions. They are also easily integrated with CI/CD pipelines, supporting faster development cycles and continuous feedback.

Although initial setup may require more effort, automated tests are reusable and scalable across different versions and environments, resulting in long-term time and cost savings. Tools such as [insert tool name] were used to implement these tests, providing practical experience with industry-standard testing frameworks.

Overall, automation improved both the quality of the software and the efficiency of the development process, aligning with current best practices in software engineering.

1.2 Automated Testing Process

1. **Test Tool Selection:** There will be some criteria for the Selection of the tool. The majority of the criteria include: Do we have skilled resources to allocate for automation tasks, Budget constraints, and do the tool satisfies our needs?

2. **Define Scope of Automation:** This includes a few basic points such as the Framework should support Automation Scripts, Less Maintenance must be there, High Return on Investment, not many complex Test Cases
3. **Planning, Design, and Development:** For this, we need to Install particular frameworks or libraries, and start designing and developing the test cases such as [NUnit](#), [JUnit](#), [QUnit](#), or required Software Automation Tools.
4. **Test Execution:** Final Execution of test cases will take place in this phase and it depends on Language to Language for .NET, we'll be using NUnit, for Java, we'll be using JUnit, for JavaScript, we'll be using QUnit or Jasmine, etc.
5. **Maintenance:** Creation of Reports generated after Tests and that should be documented to refer to that in the future for the next iterations.

1.3 Popular Automation tools

- **Selenium:** Selenium is an automated testing tool that is used for Regression testing and provides a playback and recording facility. It can be used with frameworks like [JUnit](#) and [Test-NG](#). It provides a single interface and lets users write test cases in languages like Ruby, Java, Python, etc.
- **QTP:** Quick Test Professional (QTP) is an automated functional testing tool to test both web and desktop applications. It is based on the VB scripting language and it provides functional and regression test automation for software applications.
- **Sikuli:** It is a GUI-based test automation tool that is used for interacting with elements of web pages. It is used to search and automate graphical user interfaces using screenshots.
- **Appium:** Appium is an open-source test automation framework that allows QAs to conduct automated app testing on different platforms like iOS, Android, and Windows SDK.

Chapter 2: Introduction to Selenium

2.1 Basics of Selenium

The Selenium automation tool is the most popular and open-source tool for testing the web applications. Selenium is a powerful tool for controlling web browsers through programs and performing browser automation. It is functional for all browsers, works on all major OS and its scripts are written in various languages.

Selenium was created by **Jason Huggins** in **2004** at **Thoughtworks**. He was working on an internal/web application at ThoughtWorks after some time he noticed that instead of testing his application manually, he could automate his testing. He developed a JavaScript program to test his web application, allowing him to automatically rerun tests. He called his program "**JavaScriptTestRunner**". After some time, this tool was open-sourced and renamed as **Selenium Core**.

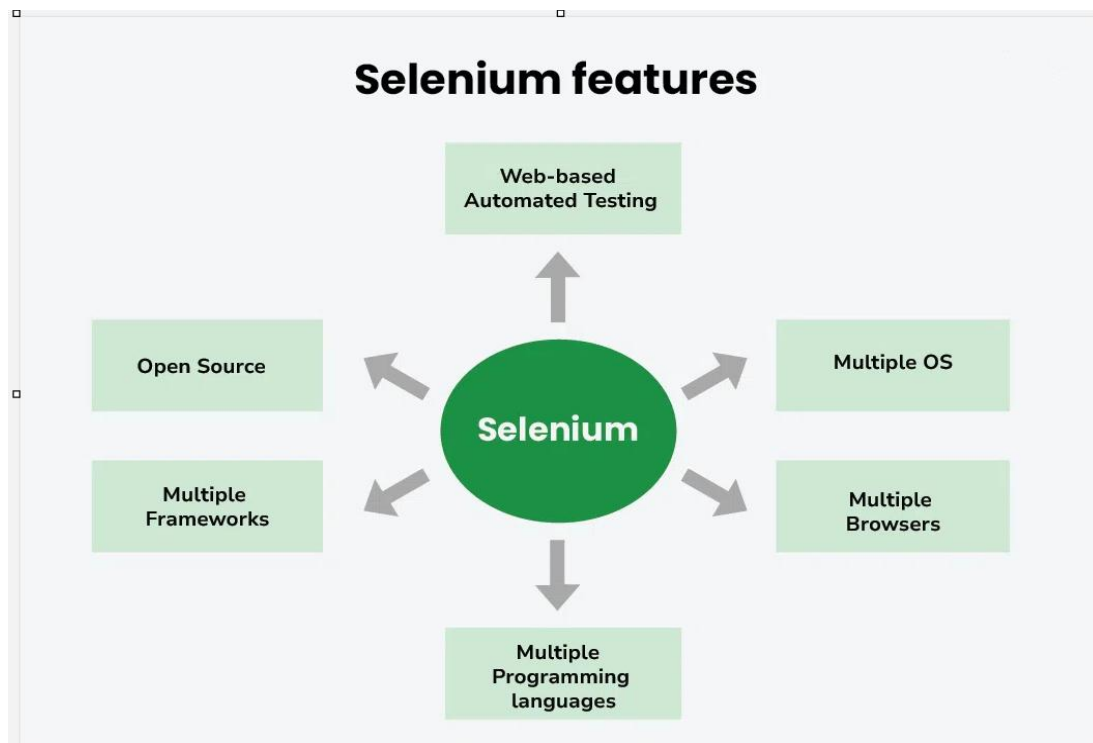


Fig 2.1 Features of Selenium

2.2 Selenium Types

Selenium IDE: A browser-based tool for recording and playing back user interactions with a web application.

Selenium RC: Selenium RC is used to automate a web browser, which was replaced by Selenium WebDriver. It let the user write test scripts in his preferred programming language and execute those on a web browser.

Selenium WebDriver: This is a tool for automating browser interactions and for controlling web browsers programmatically.

Selenium Grid: used for parallel testing across multiple machines so that it increases the speed of execution of test and utilization of resources.

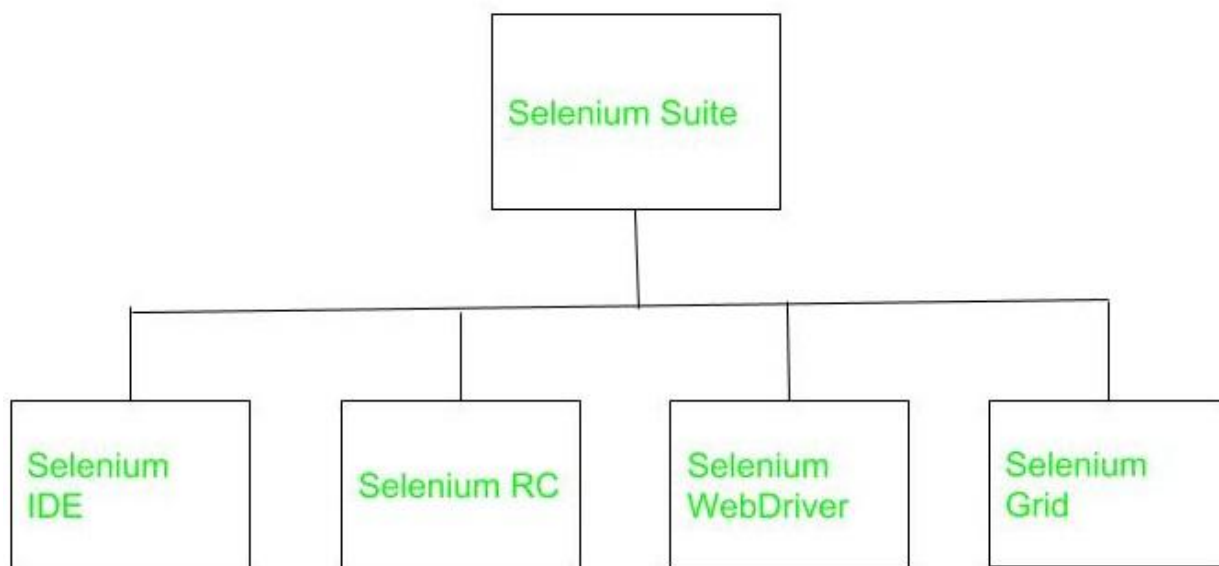


Fig 2.2 Types of Selenium

Chapter 3: Selenium IDE

3.1 Introduction to Selenium IDE

Selenium IDE is an open-source tool, mainly used for test case creation and running for web applications. It offers a very intuitive interface that helps users to automate testing through recording, editing, and debugging of tests easily.

Key Features:

- Record-and-playback
- Cross-browser testing; Chrome, Firefox
- Exporting tests into programming languages: Python, Java, JavaScript

How to install Selenium IDE?

To download and install the Selenium IDE extension, follow these steps based on your browser:

- **For Google Chrome:**

Open the Chrome Web Store: [Selenium IDE Chrome Extension](#).

Click "Add to Chrome".

Confirm by clicking "Add Extension".

The Selenium IDE icon will appear in your browser's toolbar once installed.

- **For Mozilla Firefox:**

Open the Firefox Add-ons page: [Selenium IDE Firefox Add-on](#).

Click "Add to Firefox".

Confirm the installation by clicking "Add" when prompted.

The Selenium IDE icon will be now added to the browser toolbar.

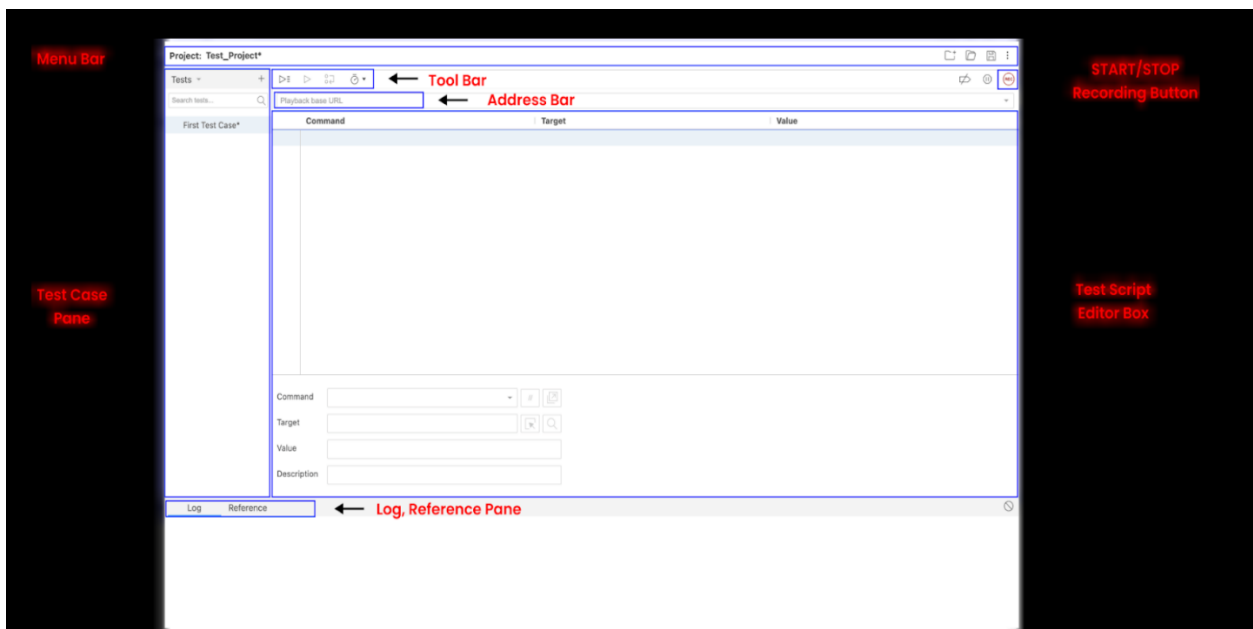


Fig 3.1 Components of Selenium IDE

The Selenium IDE (Integrated Development Environment) provides a simple graphical interface to record, edit, and execute automated test cases. The major components of the Selenium IDE window are:

1. Menu Bar

- Located at the top-left corner.
- Provides options to create new test cases, save projects, open existing scripts, and configure settings.

2. Tool Bar

- Contains essential playback controls (Run, Pause, Step, Stop).
- Allows testers to execute single steps or full test cases.

3. Address Bar

- Used to set the *Base URL* of the web application under test.
- All relative paths in test cases are executed with respect to this base URL.

4. Start/Stop Recording Button

- Helps in recording user interactions with the web application.
- Clicking this button starts capturing user actions (clicks, inputs, navigation) as test commands.

5. Test Case Pane

- Displays a list of all test cases in the current project.
- Allows easy navigation between different test cases.

6. Test Script Editor Box

- The main workspace for writing and editing Selenium commands.
- Contains three columns:
 - **Command** – the Selenium action (e.g., click, type).
 - **Target** – the web element locator (ID, XPath, CSS, etc.).
 - **Value** – input data or parameters for the command.

7. Log/Reference Pane

- Provides detailed logs of test execution, showing success, failure, or errors.
- The *Reference* tab explains the syntax and usage of the currently selected Selenium command.

3.2 Selenium Commands

Selenese is the language used within Selenium IDE to create test scripts. Selenese uses commands in Selenium IDE to mimic actions on webpages like clicking buttons.

- It's like giving instructions to a computer on how to interact with a website.
- Selenese helps testers talk to Selenium IDE without needing lots of coding knowledge.
- This makes creating test scripts easier for everyone.

Selenium commands come in three categories:

1. Actions: Commands that broadly manipulate the state of the application. They do things like **"click this link"** and **"select that option"**. Many Actions can be used with the **"AndWait"** suffix, for example **"clickAndWait"**. The suffix tells Selenium that the action will cause the browser to call the server, and that Selenium should wait for a new page to load.

2. Accessors: Access the state of the application and store the results in variables, e.g. **"storeTitle"**. They are also used to automatically generate Assertions.

3. Assertions: Assertions like Accessors, but they indicate the state of the application conforms to what is expected. Examples include **"make sure the page title is X"** and **"verify that this checkbox is checked"**.

All Selenium Assertions can be used in three modes:

- assert**
- verify**
- waitFor**

For example: **"assertText"**, **"verifyText"** and **"waitForText"**. When an **"assert"** fails, the test is aborted. When a **"verify"** fails, the test will continue execution, logging the failure. This allows a single **"assert"** to ensure that the application is on the correct page, followed by a bunch of **"verify"** assertions to test form field values, labels, etc. **"waitFor"** commands wait for some condition to become true (which can be useful for testing Ajax applications). They will succeed immediately if the condition is already true. However, they will return and stop the test if the condition doesn't become true within the current timeout setting (see the **setTimeout** action below).

Some of the commonly used Selenium Commands are listed in below table:

Command	Explanation
Open	Opens a page using a URL
click/clickAndWait	Performs a click operation, and optionally waits for a new page to load
verifyTitle/assertTitle	Verifies an expected page title
verifyElementPresent	Verifies an expected UI element, as defined by its HTML tag, is present on the page
verifyText	Verifies expected text and its corresponding HTML tag are present on the page
waitForPageToLoad	Pauses execution until an expected new page loads. Called automatically when clickAndWait is used
waitForElementPresent	Pauses execution until an expected UI element, as defined by its HTML tag, is present on the page
Store	The plain store command is the most basic of the many store commands and can be used to simply store a constant value.
echo	Echo statements can be used to print the contents of Selenium variables
storeLocation	Store current selected window's URL in selenium IDE software testing tool

Table 3.2 Commands of Selenium IDE

3.3 Limitations

- It is not able to perform tab switching.

Running 'Untitled'

1. open on <https://www.google.com/> OK
2. executeScript on window.open('https://chat.openai.com/chat', '_blank'); OK
3. pause on 3000 OK
4. selectWindow on win_ser_1 Failed:
No such window locator

Fig 3.3.1 Selenium IDE Test Execution Log with Window Locator Failure

- Even after recording we are not able to get the commands in test script editor box.
(in some cases)

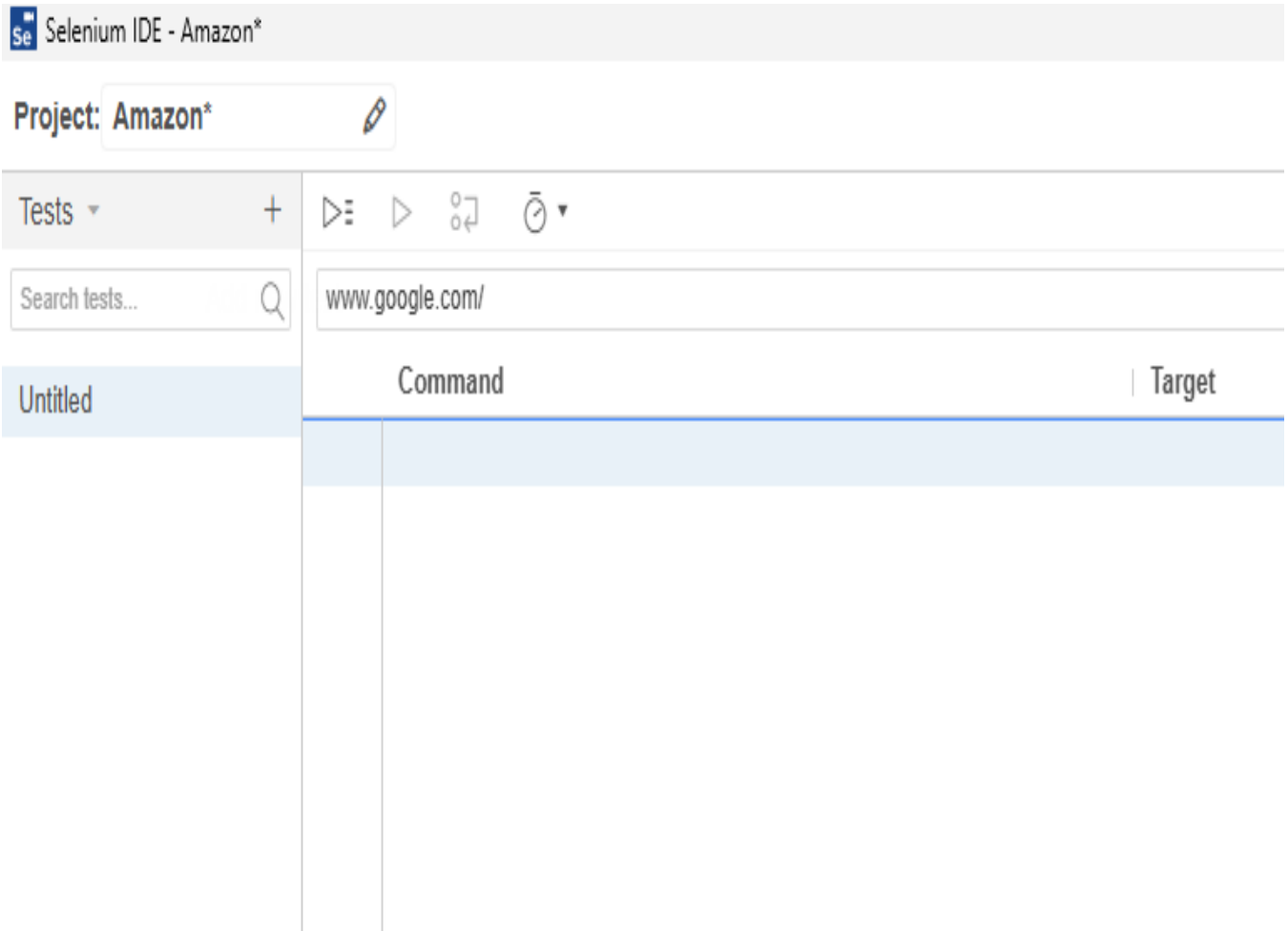


Fig 3.3.2 Selenium IDE – Empty Command List after Recording

- **Dynamic Content:** The content on the page is constantly changing and makes locators fail as the dynamic IDs or elements change every time the page loads.

	Command	Target	Value
8	✓ type	name=Passwd	Thesecret_
9	✓ click	css=.ViPpkd-mJHVfI-bMcfAe	
10	✓ click	name=Passwd	
11	✓ type	name=Passwd	Thesecret_01
12	✓ click	css=.ViPpkd-LqbsSe-OWXExe-k8QpJ > .ViPpkd-vQzf8d	
13	✓ click	css=.ytSearchboxComponentInputBox	
14	✓ type	name=search_query	fakira song
15	✗ mouse over	css=.style-scope:nth-child(1) > #dismissible .yt-core-image	
16	click	css=.style-scope:nth-child(1) > #dismissible .yt-core-image	
17	mouse out	css=.style-scope:nth-child(1) > #dismissible .yt-core-image	

Fig 3.3.3 Selenium IDE Command Execution with Mouse over Failure

3.4 Sample Project

Selenium IDE - Amazon*

Project: Amazon*

Executing ▾

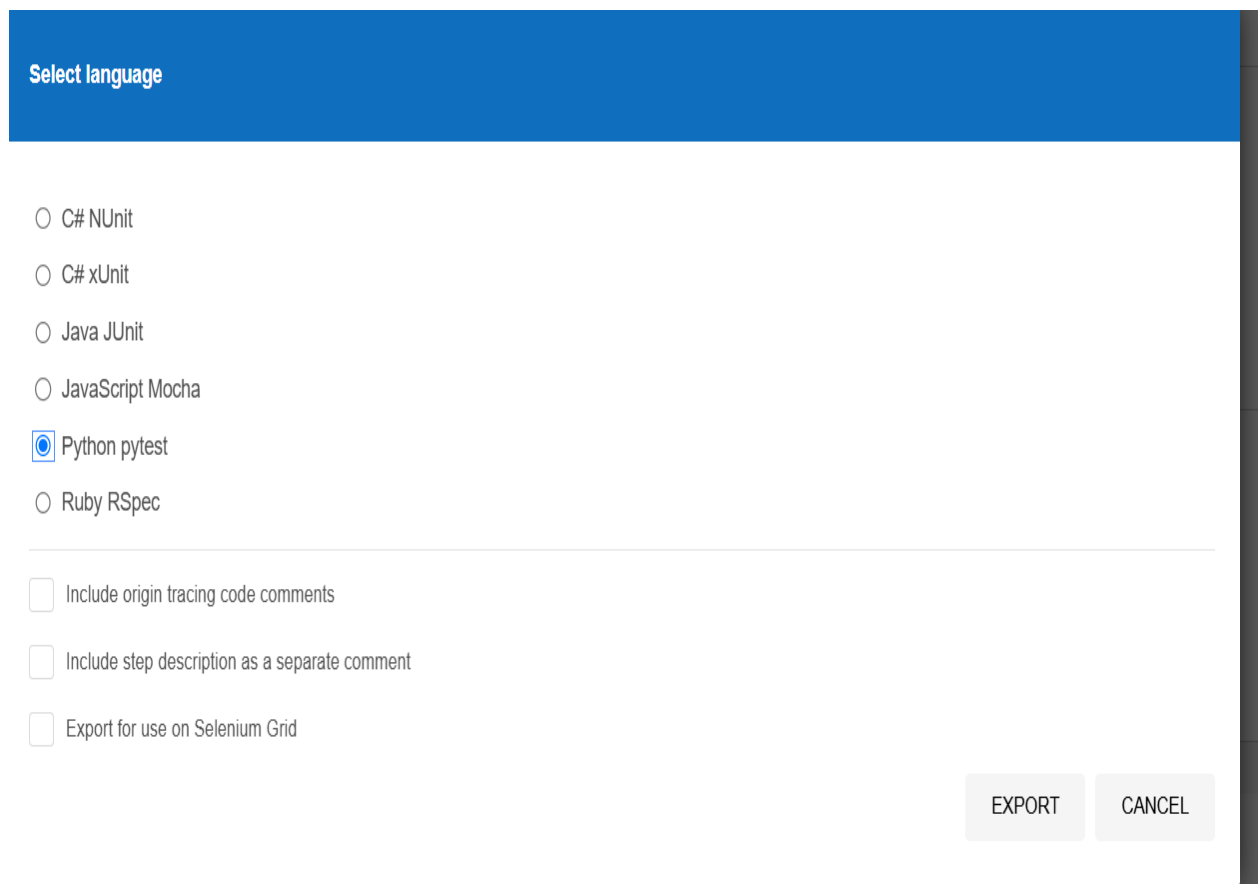
Untitled* www.google.com/

	Command	Target
1	open	https://www.amazon.in/
2	set window size	1200x800
3	type	id=twotabsearchtextbox
4	click	id=nav-search-submit-button
5	pause	
6	click	css=.s-main-slot .s-result-item h2 a
7	pause	
8	captureEntirePageScreenshot	

Fig 3.4.1 Selenium IDE Test Script for Automating Amazon Search

In this sample project we are opening the amazon website and then trying to sign in using the information(mail and password we provided) and then after signing into our amazon account click on search bar and then type a particular brand phone we want (for example Samsung Galaxy M31) and then we get list of products. Then click on first product to review the phone details.

For suppose if I want to export script in any coding language(like python) after saving project in selenium IDE we can export so that now the script that is recorded in Selenium IDE can be converted into any language format if we want.



Select language

- ☐ C# NUnit
- ☐ C# xUnit
- ☐ Java JUnit
- ☐ JavaScript Mocha
- ☒ Python pytest
- ☐ Ruby RSpec

☐ Include origin tracing code comments

☐ Include step description as a separate comment

☐ Export for use on Selenium Grid

EXPORT CANCEL

Fig 3.4.2 Exporting Test Script to Different Programming Languages

```

C:\Users\MAADHAVI\Documents\ExTest.java
1  // Generated by Selenium IDE
2  import org.junit.Test;
3  import org.junit.Before;
4  import org.junit.After;
5  import static org.junit.Assert.*;
6  import static org.hamcrest.CoreMatchers.is;
7  import static org.hamcrest.core.IsNot.not;
8  import org.openqa.selenium.By;
9  import org.openqa.selenium.WebDriver;
10 import org.openqa.selenium.firefox.FirefoxDriver;
11 import org.openqa.selenium.chrome.ChromeDriver;
12 import org.openqa.selenium.remote.RemoteWebDriver;
13 import org.openqa.selenium.remote.DesiredCapabilities;
14 import org.openqa.selenium.Dimension;
15 import org.openqa.selenium.WebElement;
16 import org.openqa.selenium.interactions.Actions;
17 import org.openqa.selenium.support.ui.ExpectedConditions;
18 import org.openqa.selenium.support.ui.WebDriverWait;
19 import org.openqa.selenium.JavascriptExecutor;
20 import org.openqa.selenium.Alert;
21 import org.openqa.selenium.Keys;
22 import java.util.*;
23 import java.net.MalformedURLException;
24 import java.net.URL;

```

Fig 3.4.3 ExTest.java

Selenium IDE serves as an excellent entry point into the world of web automation testing. As a user-friendly, record-and-playback browser extension, it allows individuals, particularly manual testers and those with limited programming experience, to quickly create and execute automated tests without writing a single line of code.

However, its limitations become apparent in more complex scenarios. Selenium IDE is not suited for handling dynamic data, creating detailed reports, or testing intricate application logic that requires advanced programming constructs. For these reasons, it is best utilized as a prototyping tool for creating simple, repeatable tests, such as smoke tests or quick sanity checks.

Chapter 4: Selenium WebDriver

4.1 Introduction

Selenium WebDriver is the most popular framework in web automation that lets developers and testers interact with web applications automatically. It gives a programming interface for executing test scripts in different languages, including Java, Python, C#, and JavaScript. It does not need a server, unlike Selenium RC, where WebDriver directly communicates with web browsers. This supports browsers like Chrome, Firefox, Edge, and Safari, through dedicated browser drivers like ChromeDriver and GeckoDriver.

It allows the web element to be located and manipulated with the help of different locator strategies like ID, Name, Class Name, CSS Selector, and XPath. Synchronization techniques in the form of implicit and explicit waits ensure stable test execution. It supports headless browser testing, allowing tests to run without opening a visible browser window. This reduces automation time and increases resource efficiency.

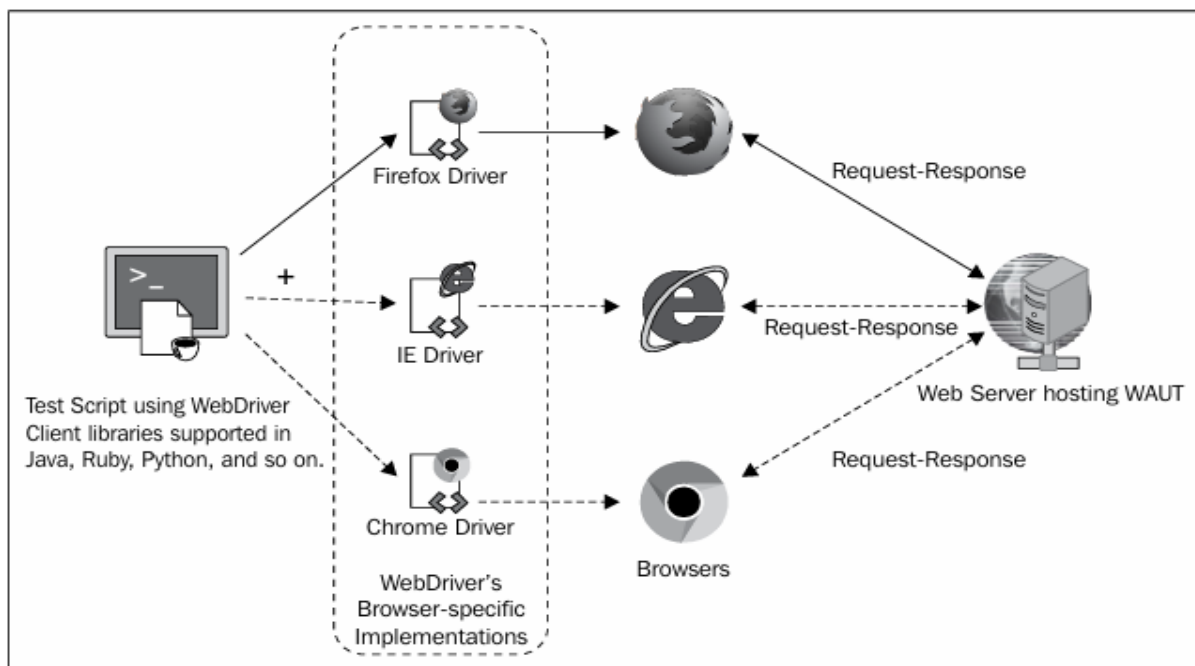


Figure 4.1 Working of Selenium WebDriver

4.2 Installation

To start using Selenium WebDriver in Java, you need to set up your development environment by installing the Selenium Java bindings. Follow these steps:

1. **Install Java:** Ensure that you have Java installed on your system. You can download and install the latest version of Java JDK from the official [Oracle website](#) if you haven't already.
2. **Set Up IDE:** Use an Integrated Development Environment (IDE) like **Eclipse**, **IntelliJ IDEA**, or **NetBeans** to write and run your Java code. Download and install your preferred IDE.
3. **Download Selenium WebDriver:**
 - Visit the official Selenium website and download the Selenium Java client driver (JAR file).
 - You can also use a build tool like **Maven** or **Gradle** to manage Selenium dependencies.
4. **Download Browser Driver:**
 - Download the appropriate WebDriver for the browser you want to automate, such as **ChromeDriver** for Google Chrome or **GeckoDriver** for Firefox. You can get these drivers from their respective official websites:
 - ChromeDriver
 - GeckoDriver(FireFox Driver)
5. **Configure WebDriver in Your Java Project:**
 - After adding the Selenium WebDriver JAR files to your project's build path, configure the WebDriver in your code. For example, to set up ChromeDriver:

```
System.setProperty("webdriver.chrome.driver",  
"path/to/chromedriver");  
WebDriver driver = new ChromeDriver();  
driver.get("https://www.example.com");
```


6. Run Your First Script:

After installation, you can write and run your first Selenium script. It will open a browser, navigate to a webpage.

We have used multiple drivers, with ChromeDriver being the most common. However, some of us have used different drivers, as shown below:

1. ChromeDriver

ChromeDriver automates Google Chrome, enabling Selenium WebDriver to run test scripts efficiently. It supports all standard WebDriver commands, including interacting with elements, navigating web pages, and handling pop-ups. Users must ensure that ChromeDriver matches the installed Chrome version for proper functionality. ChromeOptions allows advanced configurations, such as running in headless mode, disabling extensions, and managing cookies.

2. GeckoDriver (FirefoxDriver)

GeckoDriver automates Mozilla Firefox and lets Selenium talk to the browser with the WebDriver protocol. It's necessary for version 47 of Firefox and higher. GeckoDriver supports multi-threaded execution, alert handling, and JavaScript execution. Users may customize it by using FirefoxOptions to set the browser profiles and enable headless execution.

3. EdgeDriver

EdgeDriver automates Microsoft Edge. It supports the legacy EdgeHTML and the new Chromium-based one. It's similar to ChromeDriver because the new Edge is based on Chromium. Users have to ensure compatibility of versions to get stable automation. EdgeDriver supports browser configurations, extensions, and debugging tools, which can be used smoothly for testing purposes.

4. SafariDriver

SafariDriver automates Apple's Safari browser and comes pre-installed on macOS. Users must enable "Remote Automation" in Safari's Developer menu before using it. SafariDriver optimizes automation for Apple devices but offers fewer customization features than ChromeDriver and GeckoDriver.

4.3 Commands

Category	Command/Function	Description
1. Browser Command	driver.get(url)	Opens the given URL in the browser.
	driver.quit()	Closes all browser windows and ends the WebDriver session.
	driver.close()	Closes the current browser window.
	driver.maximize_window()	Maximizes the browser window.
	driver.minimize_window()	Minimizes the browser window.
	driver.refresh()	Refreshes the current page.
2. Navigation Command	driver.back()	Navigates back in browser history.
	driver.forward()	Navigates forward in browser history.
3. Locating Elements	driver.find_element(By.ID, "id")	Finds an element using ID.
	driver.find_element(By.NAME, "name")	Finds an element using name attribute.
	driver.find_element(By.XPATH, "xpath")	Finds an element using XPath.
	driver.find_element(By.CSS_SELECTOR, "selector")	Finds an element using CSS Selector.
	driver.find_elements(By.TAG_NAME, "tag")	Finds multiple elements using tag name.
4. Interacting with Elements	element.click()	Clicks on a button or link.

	<code>element.send_keys("text")</code>	Sends text input to a textbox.
	<code>element.clear()</code>	Clears text from an input field.
	<code>element.get_attribute("attribute")</code>	Retrieves an attribute value of an element.
	<code>element.text</code>	Gets the visible text of an element.
5. Handling Dropdown	<code>Select(element).select_by_visible_text("Option")</code>	Selects an option by visible text.
	<code>Select(element).select_by_index(index)</code>	Selects an option by index.
	<code>Select(element).select_by_value("value")</code>	Selects an option by value.
6. Handling Alerts	<code>Alert(driver).accept()</code>	Accepts an alert (OK).
	<code>Alert(driver).dismiss()</code>	Dismisses an alert (Cancel).
	<code>Alert(driver).text</code>	Retrieves the text of an alert.
7. Handling Frames & Windows	<code>driver.switch_to.frame("frameName")</code>	Switches to a specific iframe.
	<code>driver.switch_to.window("windowName")</code>	Switches between browser windows.
8. Handling Mouse Actions	<code>ActionChains(driver).move_to_element(element).perform()</code>	Hovers over an element.
	<code>ActionChains(driver).double_click(element).perform()</code>	Performs a double-click action.

	<code>ActionChains(driver).context_click(element).perform()</code>	Performs a right-click action.
9. Handling Cookies	<code>driver.get_cookies()</code>	Retrieves all cookies.
	<code>driver.add_cookie({"name": "test", "value": "123"})</code>	Adds a new cookie.
10. Taking Screenshots	<code>driver.save_screenshot("screenshot.png")</code>	Captures a screenshot of the current window.
11. Scrolling the Page	<code>driver.execute_script("window.scrollTo(0,500);")</code>	Scrolls down by 500 pixels.
12. Implicit & Explicit Waits	<code>driver.implicitly_wait(10)</code>	Sets an implicit wait time of 10 seconds.
	<code>WebDriverWait(driver, 10).until(EC.presence_of_element_located(By.ID, "element"))</code>	Explicit wait until an element appears.

Table 4.3. Available Functions and Commands in Selenium WebDriver

4.4 Limitations in Selenium WebDriver

1. Dynamic Web Elements

- Many web elements update dynamically with changing IDs, classes, or attributes.
- Elements can be not available immediately in the DOM.

2. Synchronization

- The script could try to act on elements when they are not yet fully loaded.
- Elements appear after AJAX calls or delayed rendering.

3. Browser Compatibility Issues

- Elements behave differently in browsers.
- Some things work in Chrome but fail in Firefox or Edge.

4. Pop-ups & Alerts Handling

- Pop-ups or authentication alerts caused by JavaScript can halt the execution of a script.
- Handling pop-ups involves context switching.

5. CAPTCHA and ReCAPTCHA Handling

- Selenium cannot solve CAPTCHA automatically.
- CAPTCHA prevents automated interactions and needs manual intervention.

6. File Upload and Download Issues

- Selenium cannot directly interact with OS file upload/download dialogs.
- File handling is different for different browsers.

7. Stale Element Exception

- An element that was in the DOM is stale after page refresh or dynamic content update.
- Interacting with a stale element throws an exception.

8. Shadow DOM Handling

- Few elements are encapsulated within a Shadow DOM; hence, are not accessible via locators
- Traditional XPath as well as CSS selectors do not work on elements of Shadow DOM

9. Handling iFrames

- Selenium does not communicate with elements residing within an iframe
- Switches between iframes to access contents.

10. Slow Test Execution

- Web automation tests take relatively longer time when compared to unit or API test.
- Browser rendering, loading, and UI interaction increase execution time.

11. Multiplication of Browser Windows & Tabs

- Selenium doesn't automatically switch between multiple windows created using browsers.
- Handling and context switching of multiple tabs is complex.

12. Locating Elements Reliably

- UI frequent changes can break locators.
- Using fix attributes like ID or Name isn't reliable for the dynamic elements.

```
StaleElementReferenceException      Traceback (most recent call last)
Cell 28[1], line 30
    28 emails = driver.find_elements(By.CSS_SELECTOR, ".list")
    29 for email in emails:
--> 30     sender = email.find_element(By.CLASS_NAME, 'ba4').text
    31     subject = email.find_element(By.CLASS_NAME, 'ba4').text
    32     preview = email.find_element(By.CLASS_NAME, 'ba4').text

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\selenium\webdriver\remote\webelement.py:400, in WebElement.find_element(self, by, value)
    398 """Find an element given a by strategy and locator.
    399 """
    400 (by, value)
    401 (...)
    405 (by, value)
    406 """
    407 by, value = self.parent_locator_converter.convert(by, value)
--> 408 return self.execute(Command.FIND_CHILD_ELEMENT, {"using": by, "value": value})["value"]

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\selenium\webdriver\remote\webelement.py:398, in WebElement._execute(self, command, params)
    393 params = {}
    394 params["id"] = self.id
--> 395 return self.parent.execute(command, params)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\selenium\webdriver\remote\webdriver.py:383, in WebDriver.execute(self, driver_command, params)
    382 response = self.command_executor.execute(driver_command, params)
    383 if response:
--> 384     self.error_handler.check_response(response)
    385     response["value"] = self._unwrap_value(response.get("value", None))
    386     return response
```

Figure 4.4 StateElementReferenceException

4.5 Sample Projects

Here are the Sample Projects that we have done in testing and Automation of Selenium WebDriver.

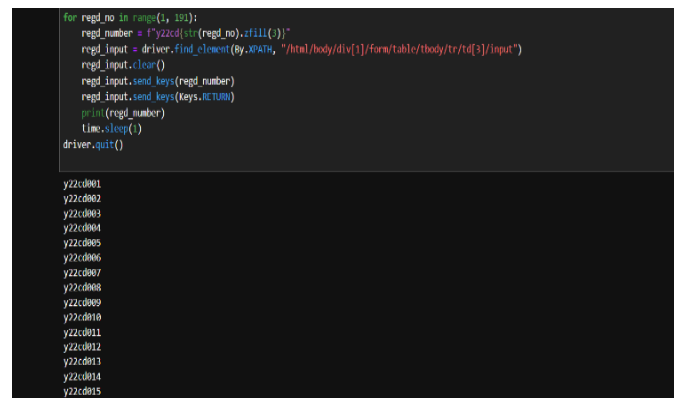
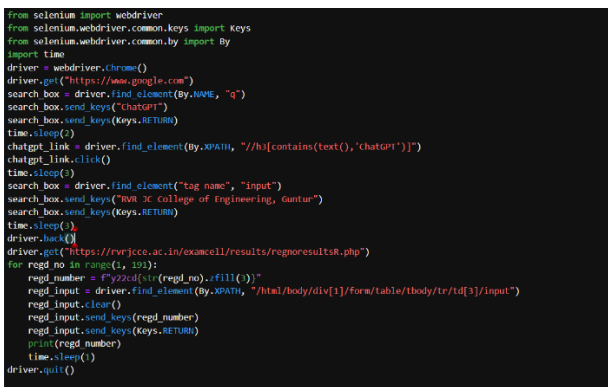
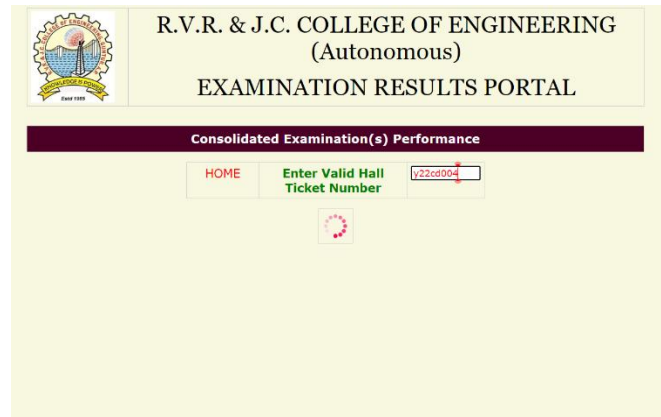
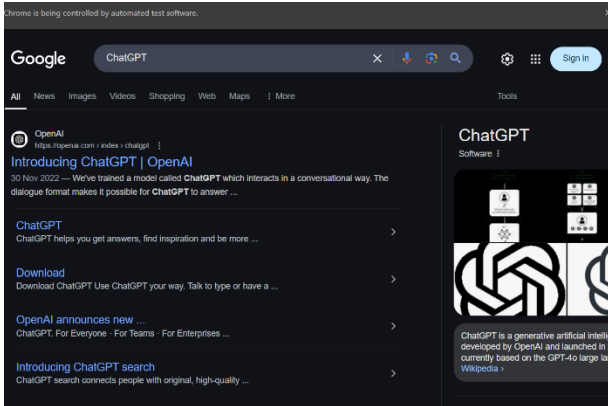
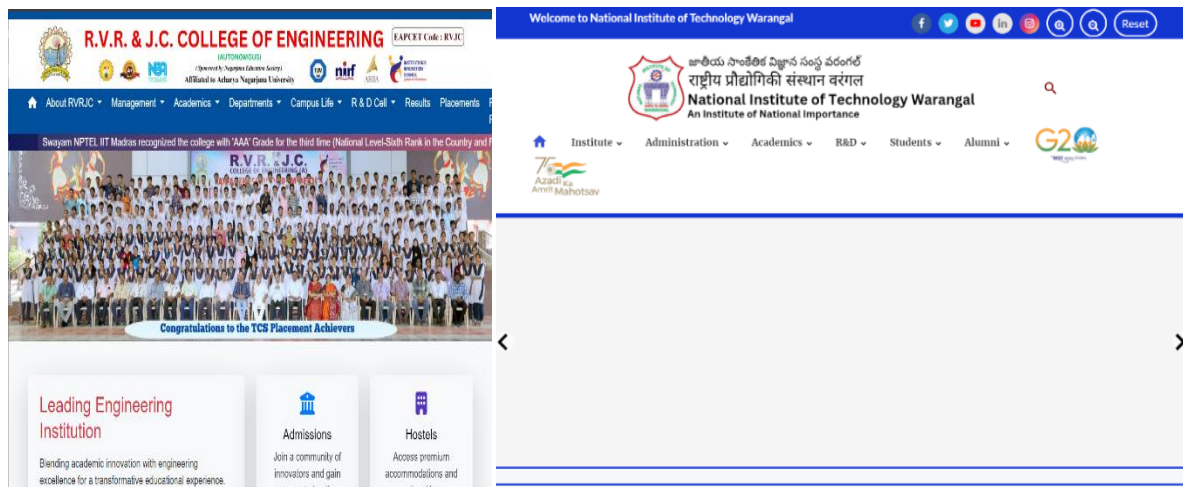


Fig 4.5.1 Automating ChatGPT chat and Getting Student Results

Note: Sometimes We Faced error with captcha, in that case we must complete captcha verification



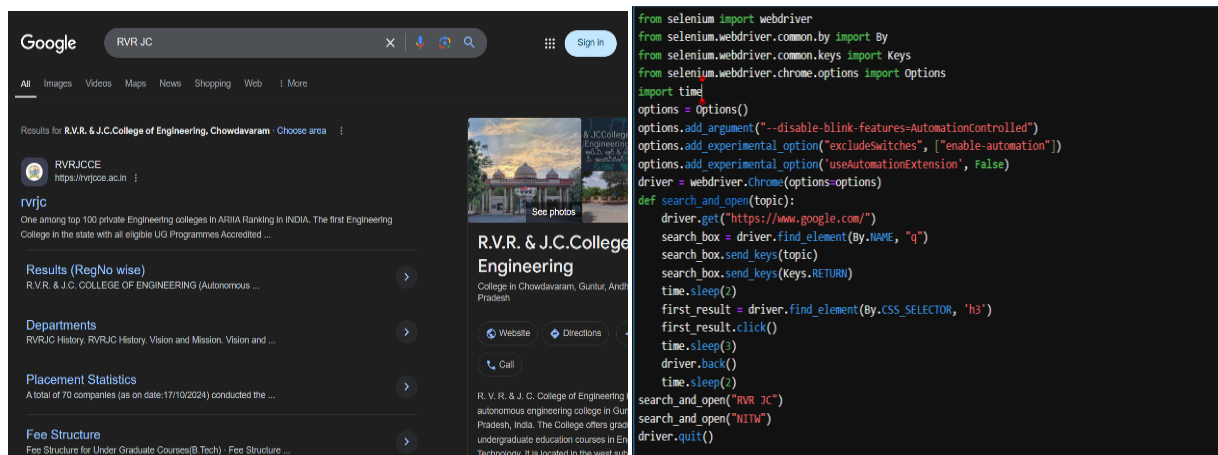


Figure 4.5.2 Automating Google Search & site opening

We have found some of the features that could be added to Selenium WebDriver to make it perfect

1. Version Control

It is important for version control as it helps in tracking changes to test scripts and handling different versions of code in Selenium automation. The integration of version control systems like Git makes it easy for testers to handle test scripts efficiently, collaborate with others, and keep track of changes made. It makes rollbacks to previous versions very easy and helps maintain stability in the testing process. Version control helps prevent conflicts when multiple testers are working on the same codebase and facilitates a structured approach to manage test case modifications, ensuring continuous improvements in automated tests over time.

2. Cloud Features

This capability allows Selenium to integrate with cloud services such as BrowserStack or Sauce Labs to execute cross-platform, scalable testing. Through these services, one can test on thousands of browsers and devices, without any need to build physical infrastructure to run tests, hence executing tests on real environments. Cloud-based execution of Selenium automation can automatically test on different platforms at the same time, therefore offering full coverage on testing. Cloud services also offer parallel test execution that reduces time taken.

Chapter 5: Selenium Grid

5.1 Introduction to Selenium Grid

Selenium Grid is a powerful component of the Selenium suite that enables developers and testers to perform distributed and parallel test execution across multiple browsers, operating systems, and machines. It provides a centralized hub-and-node architecture, allowing the hub to manage and route test commands to various nodes. This flexibility ensures efficient and scalable test execution.

The framework supports popular browsers like Chrome, Firefox, Edge, and Safari by using dedicated browser drivers such as ChromeDriver and GeckoDriver, ensuring compatibility and reliability. It allows running tests in parallel, significantly reducing the total execution time and enhancing productivity in large-scale testing scenarios.

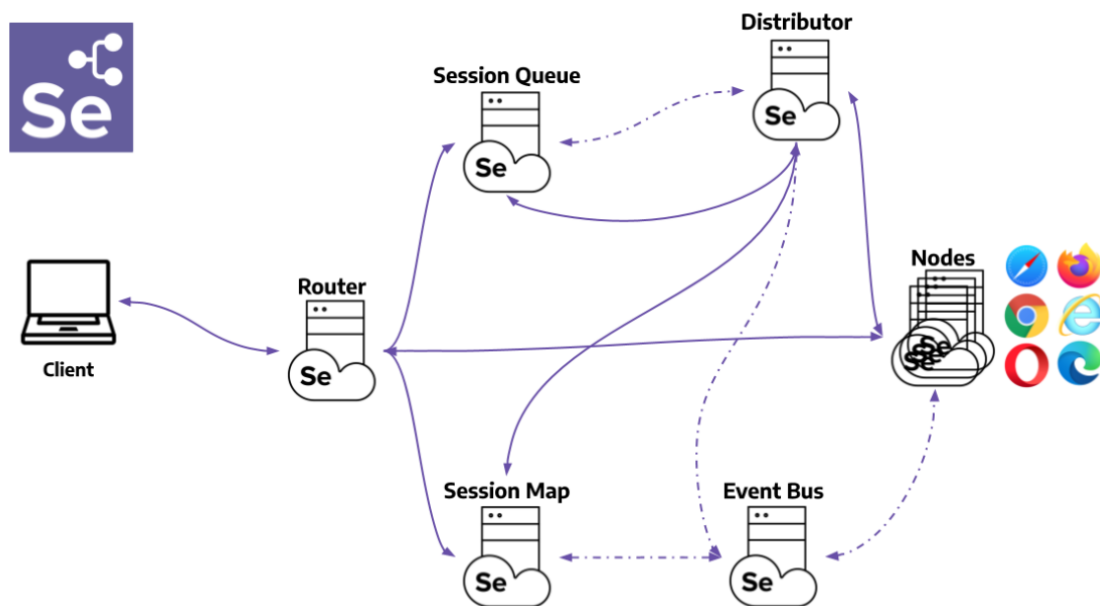


Fig 5.1 Working of Selenium Grid

This Selenium Grid comprises of three main roles:

Standalone:

- **Role:** A simpler configuration where both the Hub and Node run on the same machine.
- **Responsibilities:** Suitable for smaller or simpler test setups, where tests are executed on a single machine with a single browser.

Hub and Node (Centralized):

- **Role:** The **Hub** acts as the central server, and **Nodes** are individual machines that execute tests.
- **Responsibilities:** The Hub manages test requests and sends them to available Nodes, which run the tests on different browsers and operating systems.

Distributed:

- **Role:** A more scalable and complex setup where multiple Hubs and Nodes are spread across different machines.
- **Responsibilities:** The Hub(s) distribute test requests to multiple Nodes based on available resources, enabling parallel test execution across various environments (browsers, OS).

5.2 Components

Router:

- Directs incoming test requests to the appropriate hub or node based on load balancing or routing rules.
- Ensures efficient distribution of traffic in large-scale grid setups.

Distributor:

- Manages the distribution of test sessions to different available nodes.
- Ensures that tests are assigned to the most suitable node based on capabilities (e.g., browser, OS).

Node:

- Machines or environments where tests are executed.

Event Bus:

- A messaging system that facilitates communication between grid components (Hub, Node, Router, etc.).
- Ensures smooth coordination by handling notifications like node availability or test completion.

Selenium Hub:

- Central server that manages the test requests and allocates them to the appropriate nodes based on availability.
- Acts as a coordinator for the entire grid.

Selenium WebDriver:

- Interface used in the test scripts to interact with browsers on the nodes.
- Executes commands like clicks, typing, or navigation within browser.

5.3 Installation of Selenium Grid

Selenium Grid requires Java to run. Install the latest Java Development Kit (JDK) from the official Oracle website if it is not already installed. Verify the installation with:

```
C:\Users\Rishi\Downloads>java --version
java 20.0.2 2023-07-18
Java(TM) SE Runtime Environment (build 20.0.2+9-78)
Java HotSpot(TM) 64-Bit Server VM (build 20.0.2+9-78, mixed mode, sharing)
```

Fig 5.3.1 Verification of java version

Visit the official [Selenium website](https://www.selenium.dev/) and download the Selenium standalone server JAR file & verify.

```

C:\Users\Rishi\Downloads>java -jar selenium-server-4.27.0.jar
Selenium Server commands

A list of all the commands available. To use one, run `java -jar selenium.jar
commandName`.

completion    Generate shell autocompletions
distributor    Adds this server as the distributor in a selenium grid.
hub            A grid hub, composed of sessions, distributor, and router.
info           Prints information for commands and topics.
node           Adds this server as a Node in the Selenium Grid.
router         Creates a router to front the selenium grid.
sessionqueue   Adds this server as the new session queue in a selenium grid.
sessions       Adds this server as the session map in a selenium grid.
standalone     The selenium server, running everything in-process.

For each command, run with `--help` for command-specific help

Use the `--ext` flag before the command name to specify an additional classpath
to use with the server (for example, to provide additional commands, or to
provide additional driver implementations). For example:

java -jar selenium.jar --ext example.jar;dir standalone --port 1234

```

Fig 5.3.2 selenium-server-usage-instructions-4.27.0

➤ Now start the Hub

Use the command: `java -jar selenium-server-<version>.jar hub`

```

C:\Users\Sowjanya Narisetty\Downloads>java -jar selenium-server-4.27.0.jar hub
16:42:30.933 INFO [LoggingOptions.configureLogEncoding] - Using the system default encoding
16:42:30.939 INFO [OpenTelemetryTracer.createTracer] - Using OpenTelemetry for tracing
16:42:31.652 INFO [BoundZmqEventBus.<init>] - XPUB binding to [binding to tcp://*:4442, advertising as tcp://192.168.56.1:4442], XSUB binding to [binding to tcp://*:4443, advertising as tcp://192.168.56.1:4443]
16:42:31.724 INFO [UnboundZmqEventBus.<init>] - Connecting to tcp://192.168.56.1:4442 and tcp://192.168.56.1:4443
16:42:31.762 INFO [UnboundZmqEventBus.<init>] - Sockets created
16:42:32.771 INFO [UnboundZmqEventBus.<init>] - Event bus ready
16:42:35.370 INFO [Hub.execute] - Started Selenium Hub 4.27.0 (revision d6e718d): http://192.168.56.1:4444

```

Fig 5.3.3 Selenium Hub Startup - CLI Output

- Verify the connection by navigating to **http://<IP address>:<port number>**

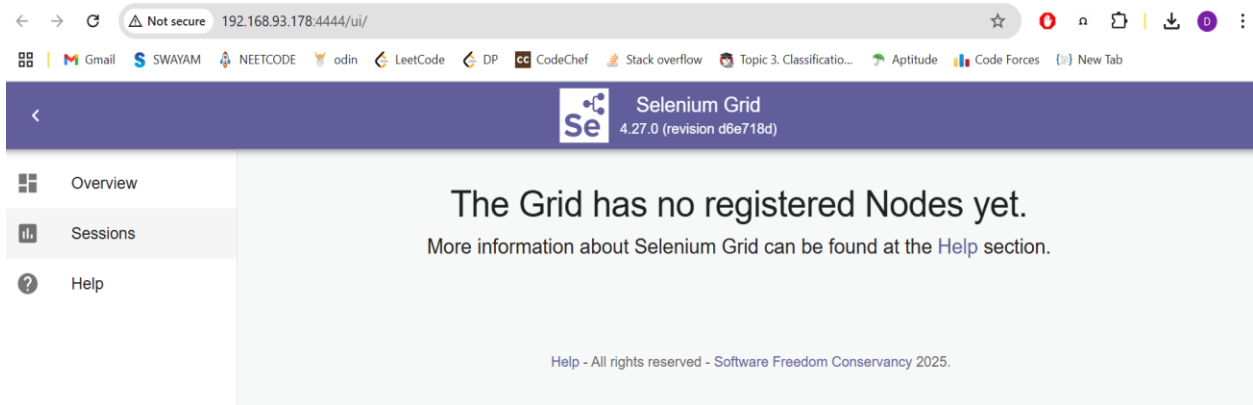


Figure 5.3.4 Verification of port address

- On each machine that will act as a node, execute the following command:

java -jar selenium-server-<version>.jar node --hub http://<hub-ip>:4444

```
PS C:\Users\Rishi\downloads> java -jar selenium-server-4.27.0.jar node --hub https://192.168.56.1:4444
16:45:37.239 INFO [LoggingOptions.configureLogEncoding] - Using the system default encoding
16:45:37.255 INFO [OpenTelemetryTracer.createTracer] - Using OpenTelemetry for tracing
16:45:37.454 INFO [UnboundZmqEventBus.<init>] - Connecting to tcp://192.168.56.1:4442 and tcp://192.168.56.1:4443
16:45:37.531 INFO [UnboundZmqEventBus.<init>] - Sockets created
16:45:38.538 INFO [UnboundZmqEventBus.<init>] - Event bus ready
16:45:39.241 INFO [NodeServer.createHandlers] - Reporting self as: http://192.168.93.178:5555
16:45:39.785 INFO [NodeOptions.getSessionFactories] - Detected 8 available processors
16:45:39.789 INFO [NodeOptions.discoverDrivers] - Looking for existing drivers on the PATH.
16:45:39.789 INFO [NodeOptions.discoverDrivers] - Add '--selenium-manager true' to the startup command to setup drivers automatically.
16:45:42.691 INFO [NodeOptions.report] - Adding Chrome for {"browserName": "chrome", "platformName": "Windows 11"} 8 times
16:45:42.691 INFO [NodeOptions.report] - Adding Edge for {"browserName": "MicrosoftEdge", "platformName": "Windows 11"} 8 times
16:45:42.691 INFO [NodeOptions.report] - Adding Internet Explorer for {"browserName": "internet explorer", "platformName": "Windows 11"} 1 times
16:45:42.699 INFO [NodeOptions.report] - Adding Firefox for {"browserName": "firefox", "platformName": "Windows 11"} 8 times
16:45:42.777 INFO [Node.<init>] - Binding additional locator mechanisms: relative
16:45:43.591 INFO [NodeServer.$2.start] - Starting registration process for Node http://192.168.93.178:5555
16:45:43.600 INFO [NodeServer.execute] - Started Selenium node 4.27.0 (revision d6e718d): http://192.168.93.178:5555
16:45:43.626 INFO [NodeServer.$2.lambda$start$1] - Sending registration event...
16:45:43.934 INFO [NodeServer.lambda$createHandlers$2] - Node has been added
```

Fig 5.3.5 Selenium Node Connecting to Hub - CLI Output

➤ Verify the node by navigating to the browser

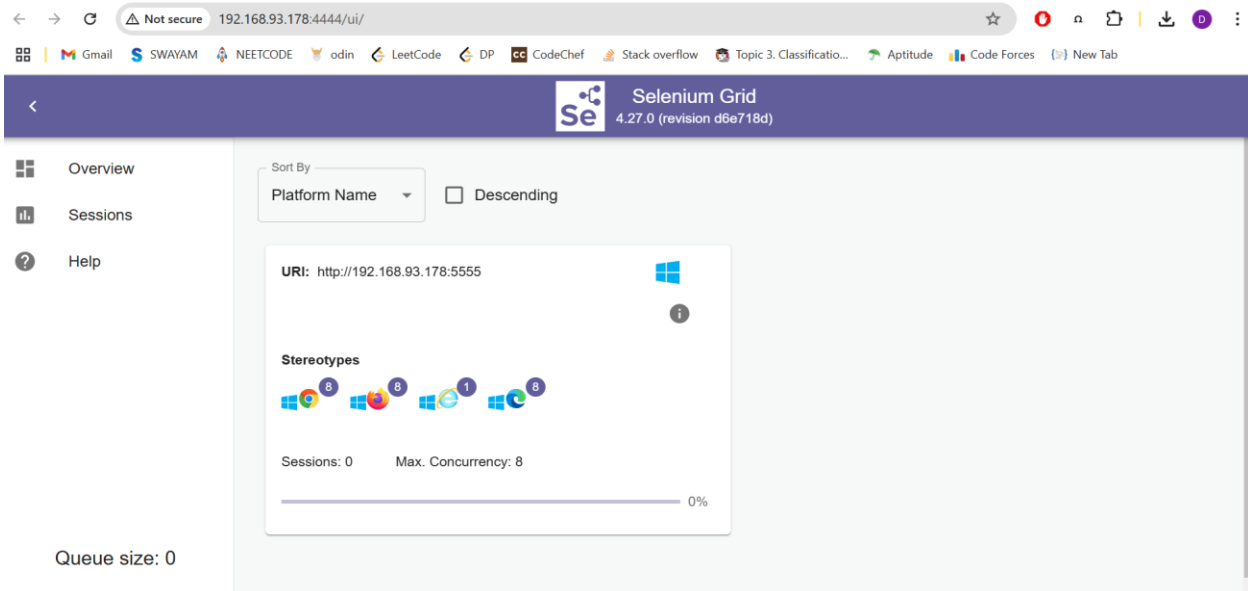


Figure 5.3.6 Verification of Navigation Selenium Grid

➤ Automate starting selenium nodes from hub

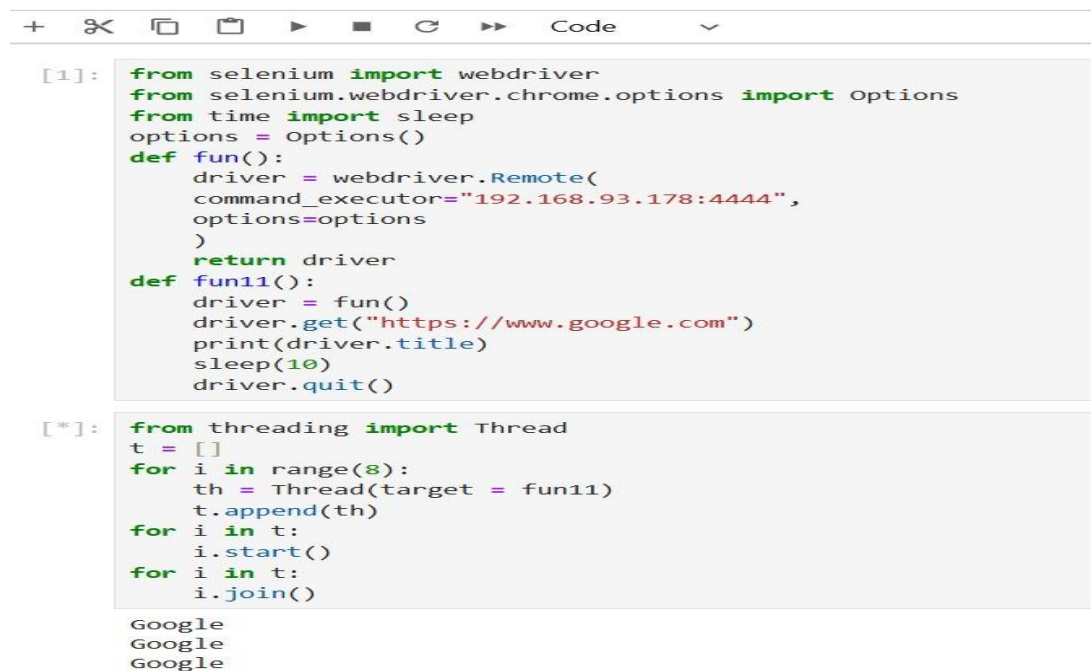


Fig 5.3.7 Python Script for Parallel Selenium Remote WebDriver Sessions

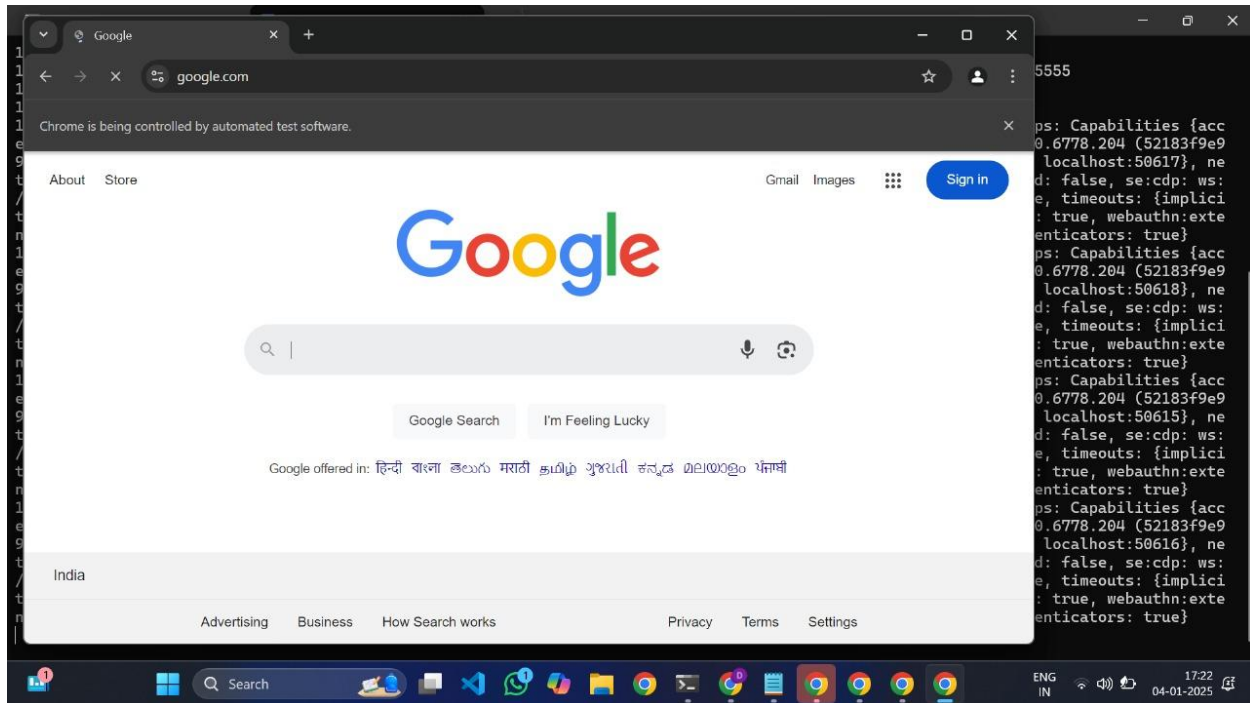


Fig 5.3.8 Working with Selenium Grid

With 2 Nodes, Selenium Grid allows tests to run in parallel, distributing them across the available nodes. Here's an example:

- **Node 1: Firefox (latest version)**
- **Node 2: Chrome (latest version)**

The formula for execution time is:

- $\text{Number of Tests} * \text{Average Test Time} / \text{Number of Nodes} = \text{Total Execution Time}$

Example:

- Without Grid: 15 tests * 45s = 11m 15s (on 1 node)
- With Nodes: $15 \text{ tests} * 45\text{s} / 2 = 6\text{m } 45\text{s}$. This means that with 2 Nodes, the total execution time is reduced to 6 minutes and 45 seconds, demonstrating how parallel execution across nodes can greatly speed up test execution.

Chapter 6: TrustInn

6.1 Introduction

Software testing plays a vital role in ensuring the quality and reliability of applications. Among the most widely used tools in industry is **Selenium**, an open-source framework primarily designed for automating web applications. Selenium enables functional and regression testing across multiple browsers and platforms.

To address these gaps, during our internship we explored **TrustInn**, a research-driven verification tool developed at **NIT Warangal**. Unlike Selenium, which focuses on user interaction testing, TrustInn is designed to run inside **Oracle VirtualBox** and supports **formal verification techniques** through tools like CBMC, SC-MCC-CBMC, MCDL-TX, Verisol, and others. TrustInn thus complements Selenium by offering a **deeper layer of program analysis** that extends beyond functional testing, enabling detection of logical errors, unreachable code, and even smart contract vulnerabilities.

6.2 Issues we faced while opening TrustInn:

- a) **Aborted Error:** It sometimes doesn't open on a few PCs, showing an 'Aborted' message.

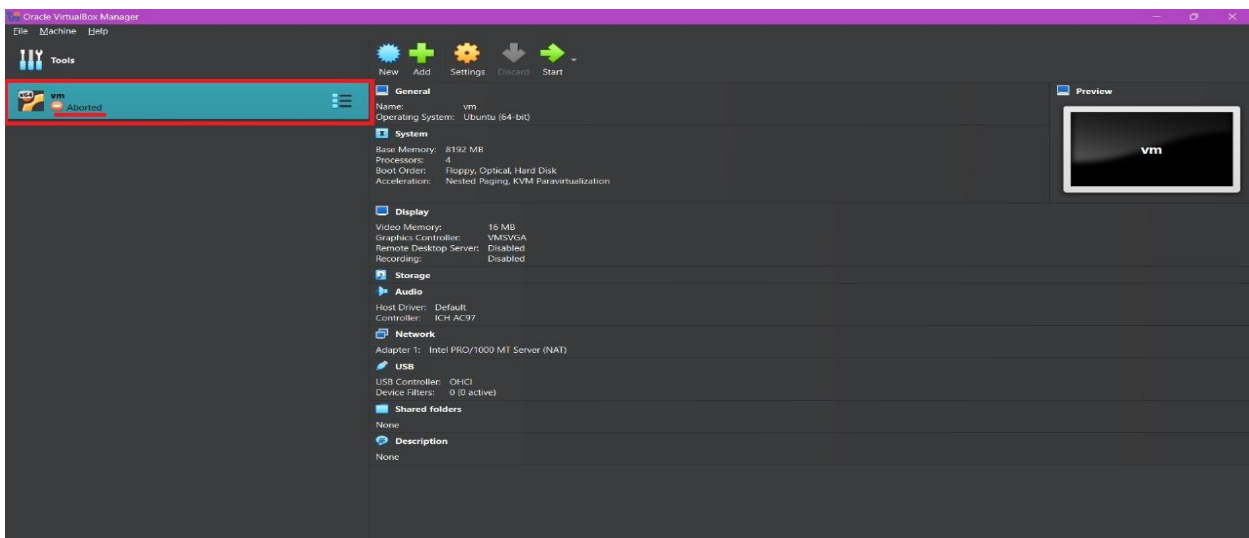


Fig 6.2.1 Aborted error

- b) **Not Listed:** The tool does not appear or get recognized in other systems, hinting at potential environmental setup problems.



Fig 6.2.2 not listed error

To solve problem, we have to first power off the machine and then we have to go to settings of VM and then go to expert mode. Now in display change settings to “VMSVGA”. Now go to **Settings >> expert >> display >> change to “VMSVGA”**.

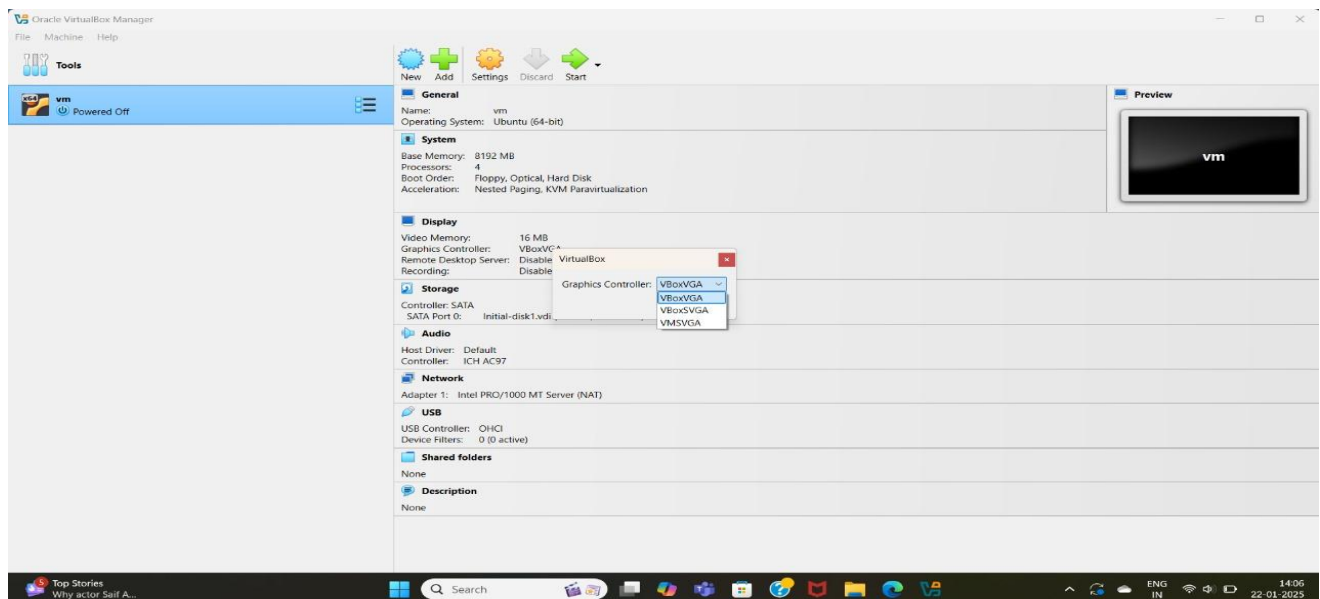


Fig 6.2.3 Bug clearance

c) **Resolution Approach:** Ensure the environment is configured consistently across systems to avoid such discrepancies.

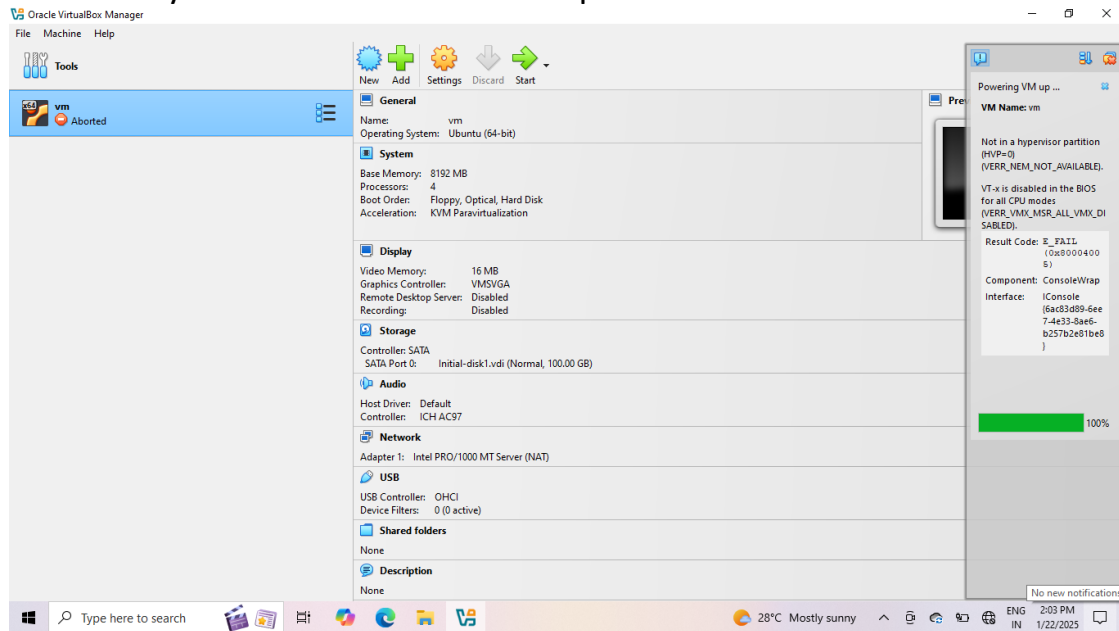


Figure 6.2.4 Resolution Approach

To solve this problem, we have to enable virtualization in system configurations.

a) **Action Steps:** Check for tool installation, ensure the paths for the system are valid, and search for any version conflicts in other installed software.

Proper troubleshooting will ensure consistent and reliable operation of the tool.

b) **Copy/Paste from host to guest OS**

The copy/paste is however not allowed by guest to host. Similarly, we cannot copy/paste from host to guest OS.

Settings (must be in expert mode) > Advanced > Shared clipboard – [Bi directional] > Drag ‘n drop -[Bi directional] > ok

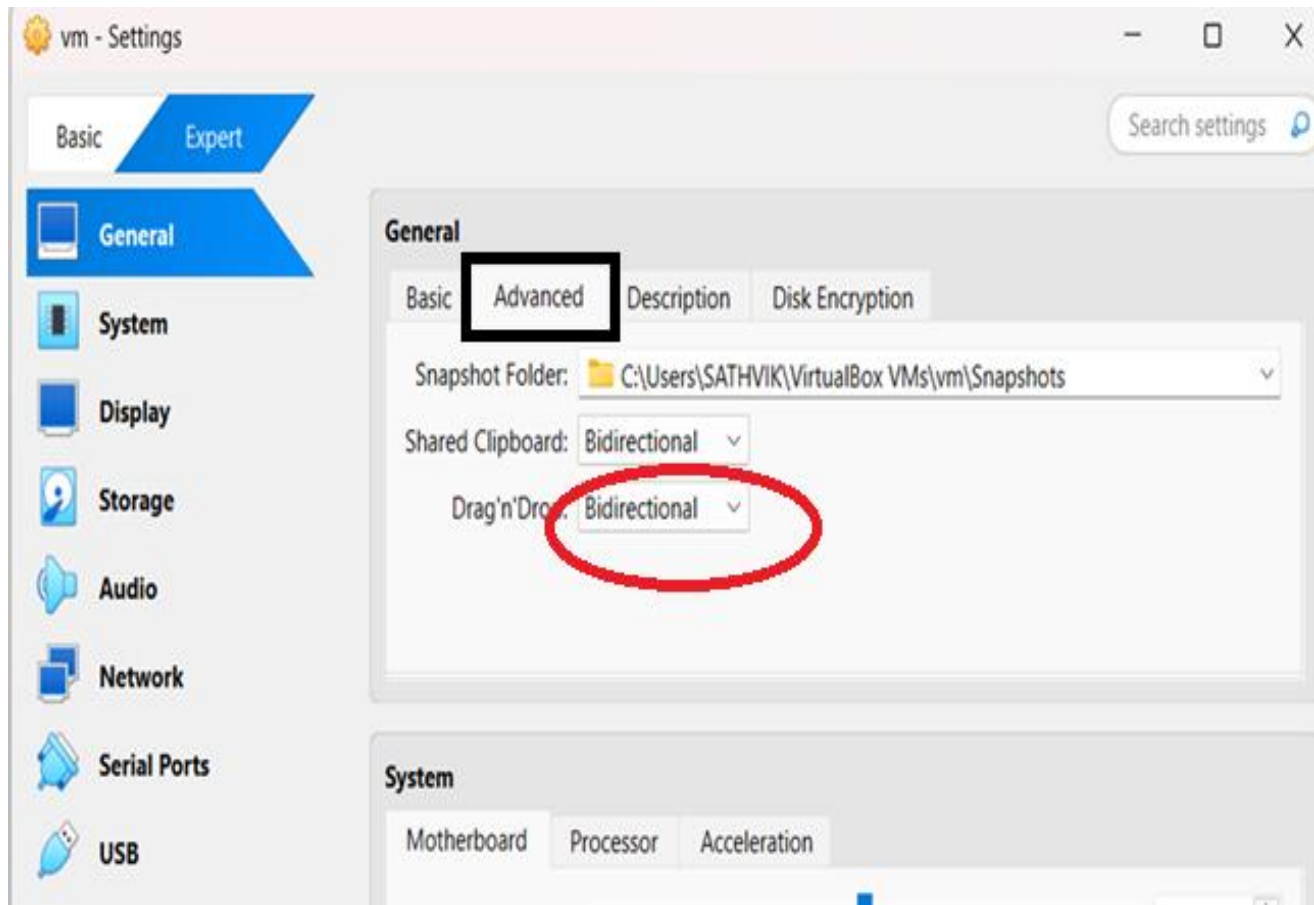


Fig 6.2.5 VirtualBox VM Settings - General and System Configuration

6.3 Testing Tools

6.3.1 Testing the CBMC tool (using C files):

a) Input

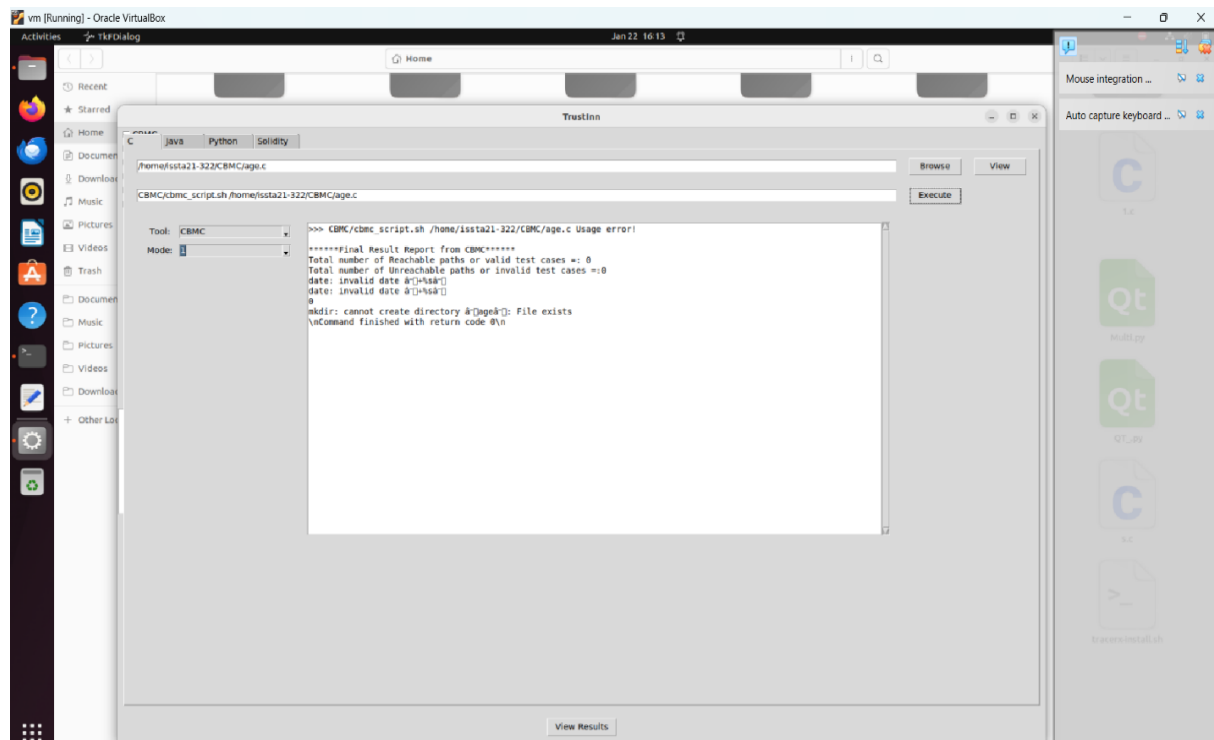


Fig 6.3.1.1 CBMC tool (file: age.c)

Expected output:

```
issta21-322@STT202122:~/CBMC/age$ ls
age.c age-report.txt age-result.txt
issta21-322@STT202122:~/CBMC/age$ cbmc age.c
CBMC version 5.12 (cbmc-5.12) 64-bit x86_64 linux
Parsing age.c
Converting
Type-checking age
Generating GOTO Program
Adding CPROVER library (x86_64)
Removal of function pointers and virtual functions
Generic Property Instrumentation
Running with 8 object bits, 56 offset bits (default)
Starting Bounded Model Checking
**** WARNING: no body for function __isoc99_scanf
size of program expression: 120 steps
simple slicing removed 0 assignments
Generated 0 VCC(s), 0 remaining after simplification
VERIFICATION SUCCESSFUL
issta21-322@STT202122:~/CBMC/age$
```

Fig 6.3.1.2 CBMC Verification Output

b) Input

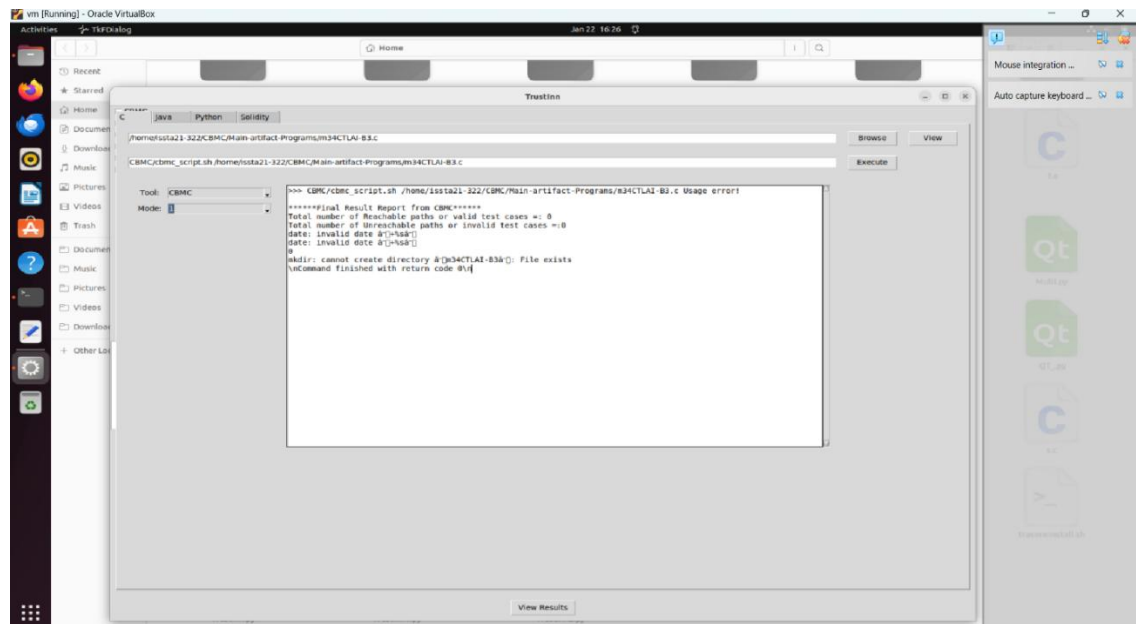


Fig 6.3.1.3 CBMC Tool (filename: m34CTLAI-B2.c)

Expected output:

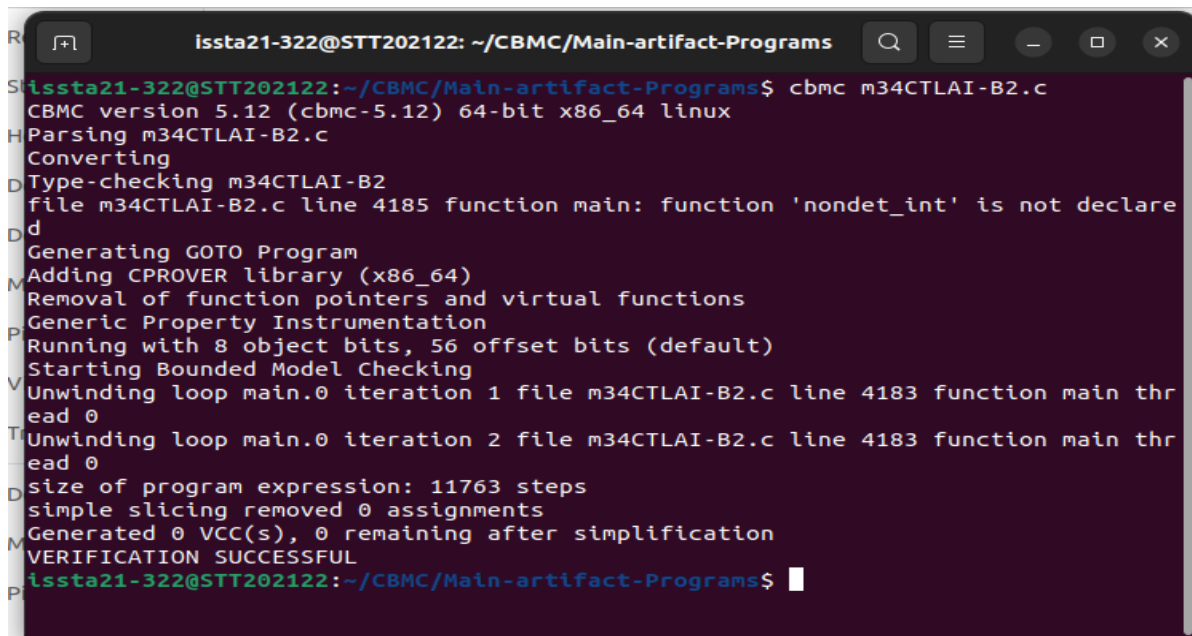


Fig 6.3.1.4 CBMC output verification

c) Input

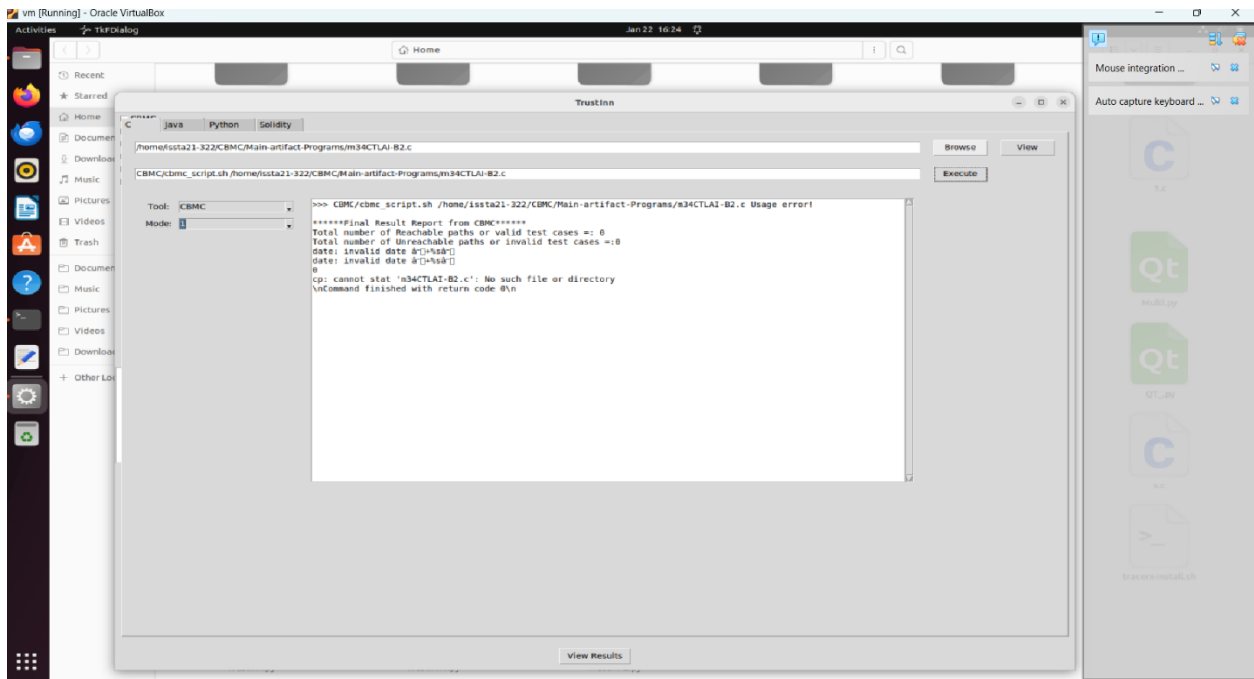


Fig 6.3.1.5 CBMC (file : m34CTLAI-B3.c)

Expected output:

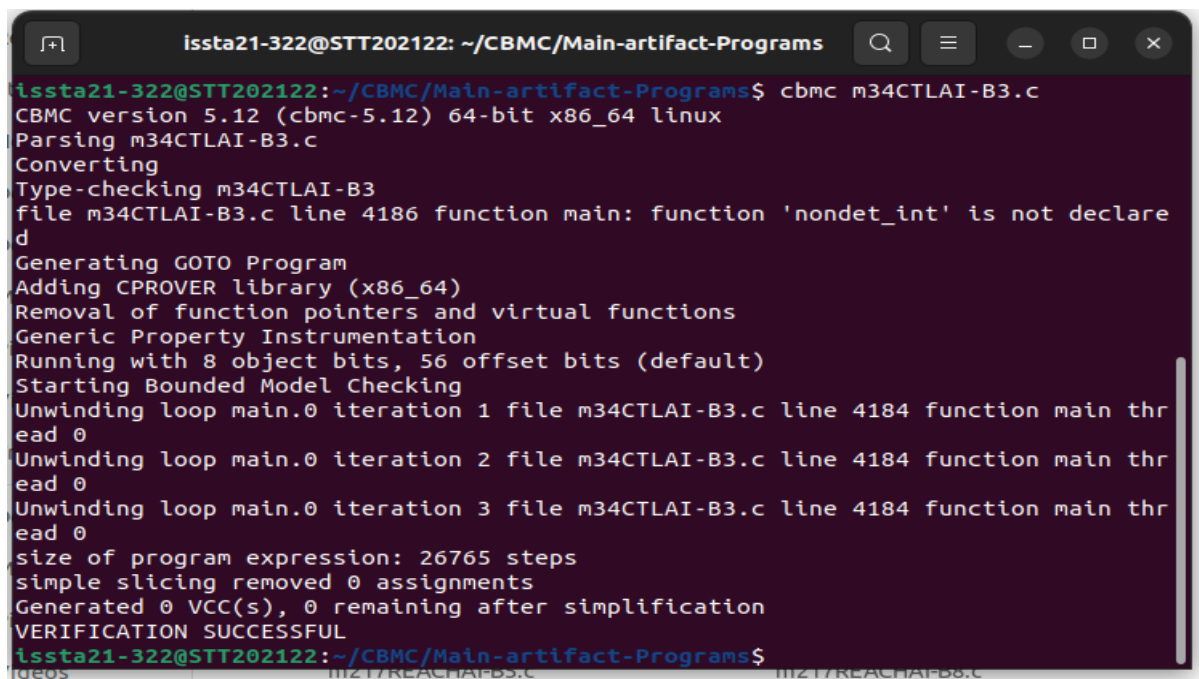


Fig 6.3.1.6 Output verification

When the CBMC tool was used to execute file.c, the execution process had various usage errors according to the error logs. This report shows issues such as date format errors including; date: invalid date +%sa and date: invalid date +%sâ and a mkdir: cannot create directory 'age': File exists error. It has also been presented that the final result report reveals no reachable paths or valid test cases were identified, with a total of zero invalid test cases. These errors suggest potential misconfigurations in the script or input file paths. It is therefore recommended to check the correctness of the script commands, proper file permissions, and check the existence of required directories for rectifying these issues.

Here we tested the CBMC tool with files (.c files) which are existing in CBMC directory like age.c, zodiac.c, m34CTLAI-B3.c etc.

6.3.2 Testing the SC-MCC-CBMC using C Files

We have tested various files using SC-MCC-CBMC tool and the results are as follows:

Initially we have tested with mode 1 and mode 2, Surprisingly, the tool works without a mode too

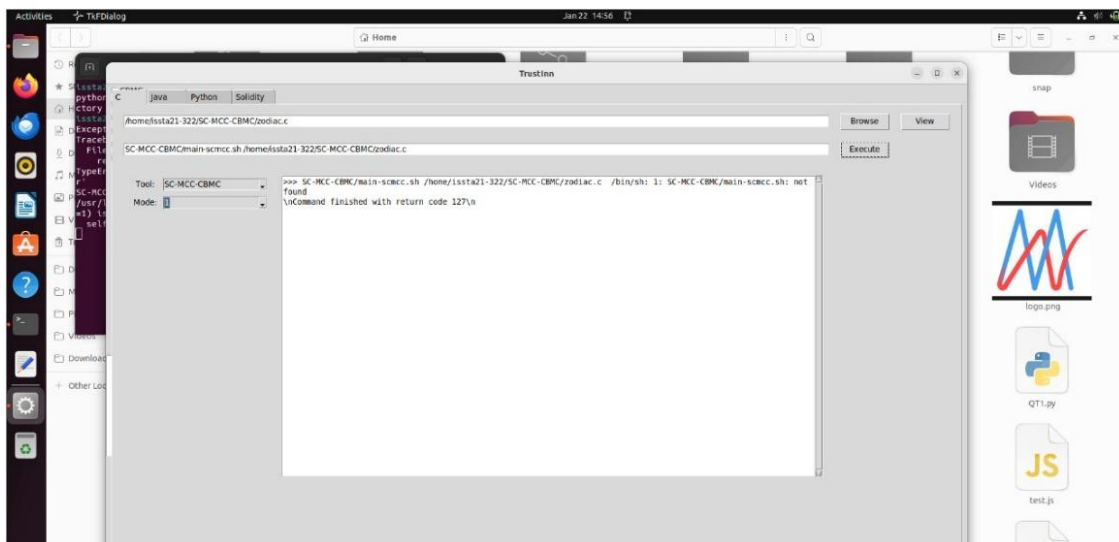


Fig 6.3.2 SC-MCC-CBMC tool

6.3.3 Testing of MCDC – TX Tool using C files

We have tested MCDC – TX tool using C file in mode 1 and mode 2 respectively. All the tests are shown below: Mode 1

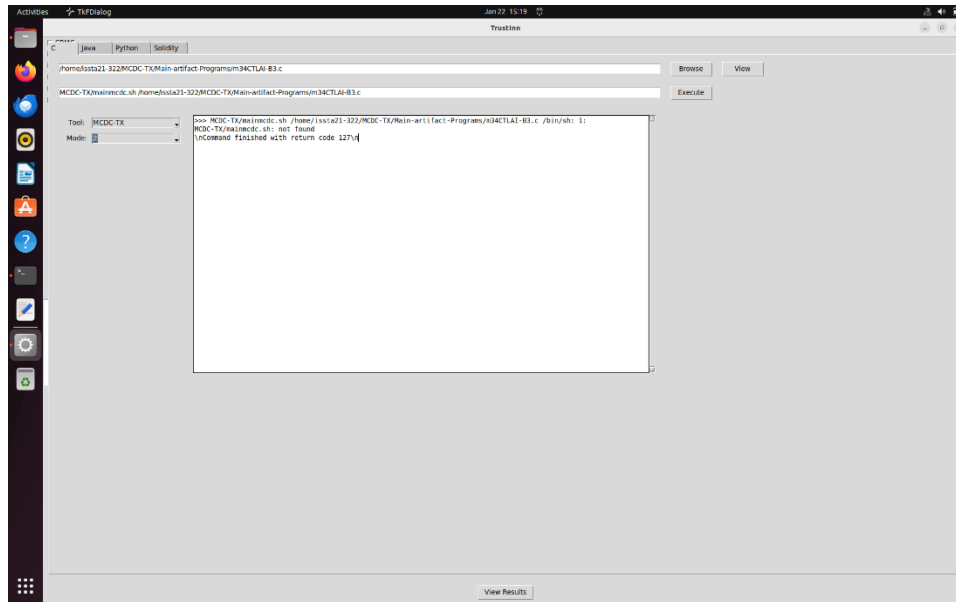


Fig 6.3.3 MCDC-TX tool

6.3.4 Testing of Verisol Tool using Solidity

We have also tested Verisol using Solidity and here is the test report as follows:

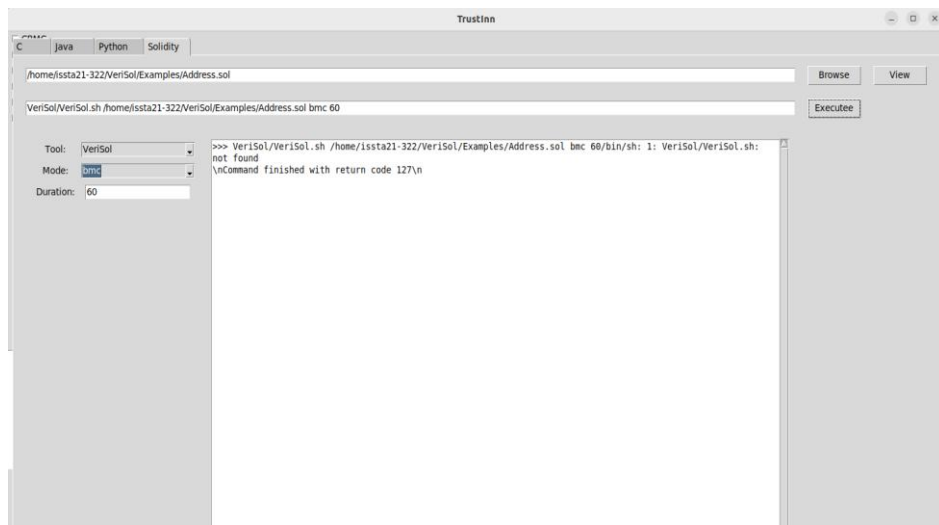


Fig 6.3.4 Verisol tool

File name: Address.sol

Mode: bmc

We conducted a series of tests using the SC-MCC-CBMC tool on various C files. Initially, tests were executed in both mode 1 and mode 2, and surprisingly, the tool also operated without specifying a mode. For example, the file *zodiac.c* was tested in mode 1, producing expected outputs. Similarly, *file2.c* was evaluated under both mode 1 and mode 2 for comparative results, while *PS-P18-R12-B1.c* underwent mode 1 testing with expected outputs noted.

For MCDC-TX, we tested multiple C files in modes 1 and 2. While *m34CTLai_B4.c* in mode 1 provided insights, time constraints limited our ability to document all expected outputs.

Additionally, we tested Solidity files using the Verisol tool. File *Address.sol* was tested in BMC mode, and *Animalia.sol* in CHC mode. Notably, the file *Anamilia.sol* was also tested in CHC mode to analyze outputs for Solidity-based applications.

Overall, these tests offered valuable insights into tool capabilities and potential optimizations, though certain results remain pending due to time constraints.

Finally, we find the tools should be reconstructed again.

6.3.5 Testing KLEEMA tool

Filename: sample.c

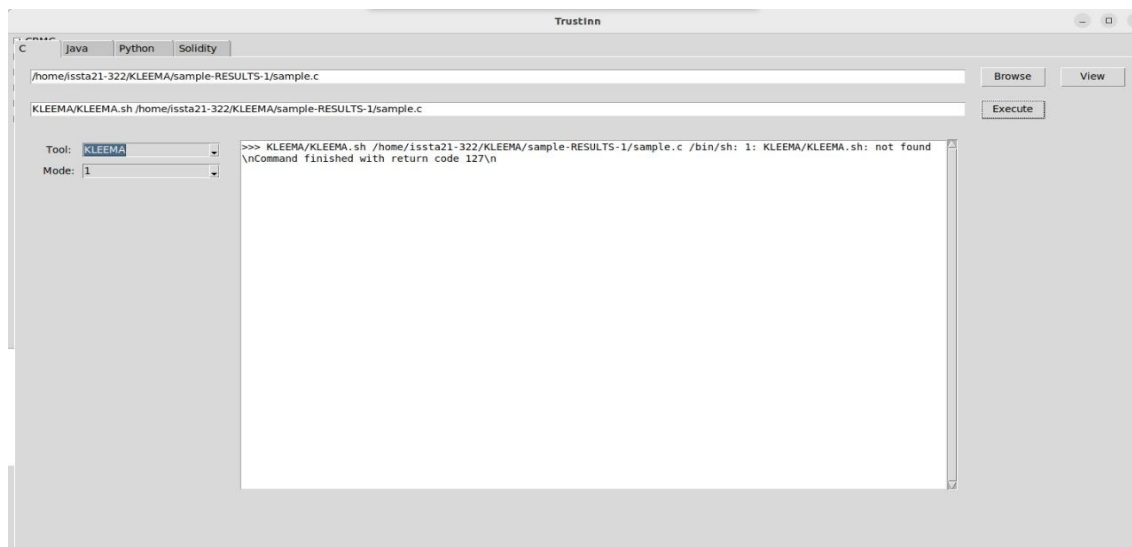
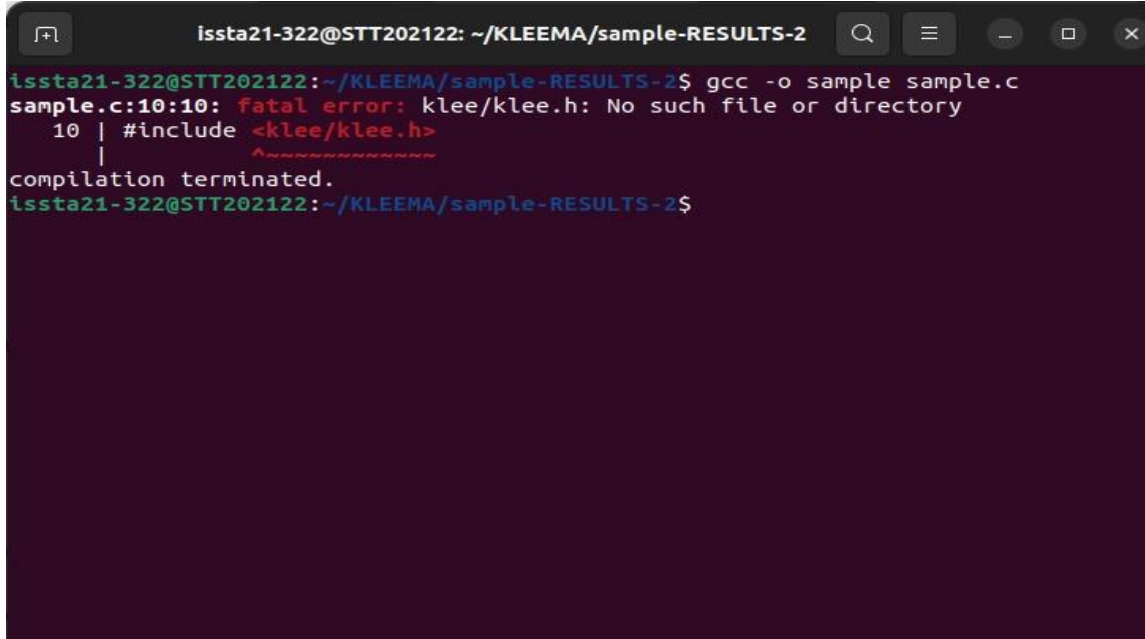


Fig 6.3.5.1 KLEEMA tool

Expected Output:



```
issta21-322@STT202122: ~/KLEEMA/sample-RESULTS-2
issta21-322@STT202122:~/KLEEMA/sample-RESULTS-2$ gcc -o sample sample.c
sample.c:10:10: fatal error: klee/klee.h: No such file or directory
   10 | #include <klee/klee.h>
      |          ^
compilation terminated.
issta21-322@STT202122:~/KLEEMA/sample-RESULTS-2$
```

Fig 6.3.5.2 Output verification

When worked on KLEE we got this error <klee/klee.h> No file or directory when we executed sample.c file this error persisted on both mode 1 and mode 2.

6.3.6 Testing with Static-Analysis tool

Filename: trail-teaching.c

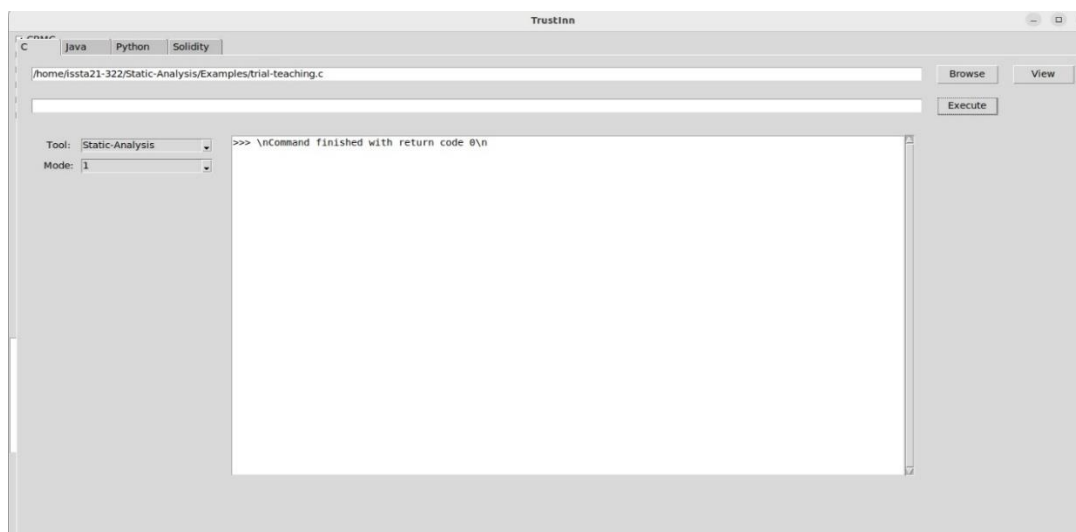
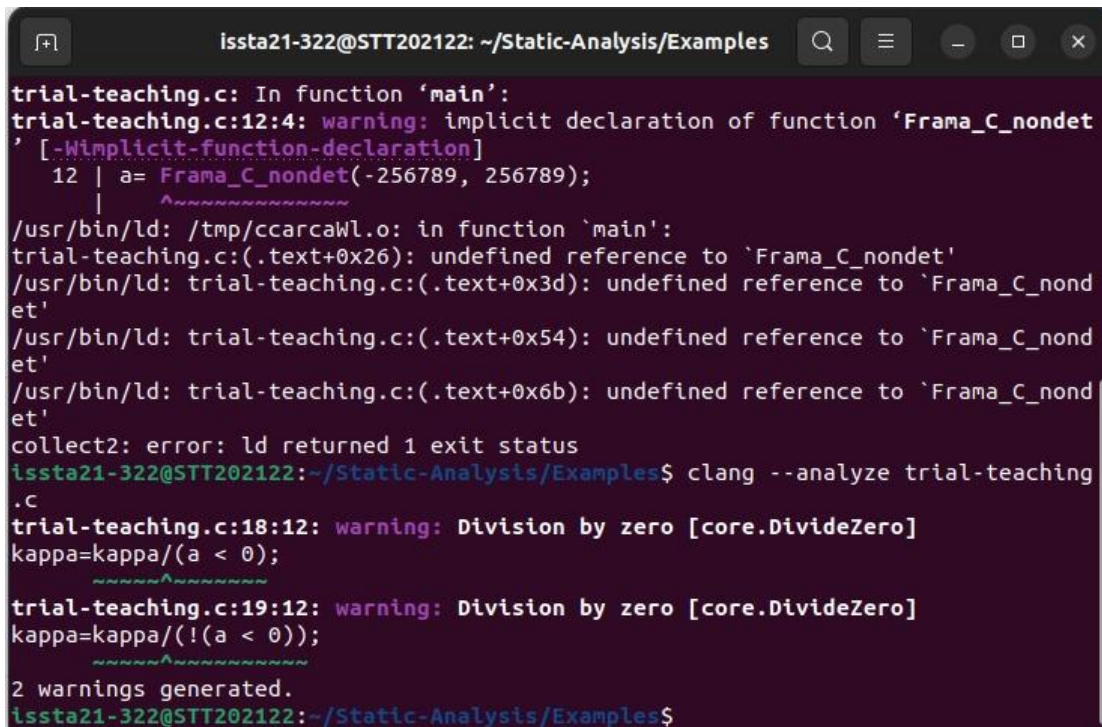


Fig 6.3.6.1 Static Analysis tool

Output:



```
issta21-322@STT202122: ~/Static-Analysis/Examples
trial-teaching.c: In function 'main':
trial-teaching.c:12:4: warning: implicit declaration of function 'Frama_C_nondet'
' [-Wimplicit-function-declaration]
   12 | a= Frama_C_nondet(-256789, 256789);
      |
/usr/bin/ld: /tmp/ccarcaWl.o: in function 'main':
trial-teaching.c:(.text+0x26): undefined reference to `Frama_C_nondet'
/usr/bin/ld: trial-teaching.c:(.text+0x3d): undefined reference to `Frama_C_nondet'
/usr/bin/ld: trial-teaching.c:(.text+0x54): undefined reference to `Frama_C_nondet'
/usr/bin/ld: trial-teaching.c:(.text+0x6b): undefined reference to `Frama_C_nondet'
collect2: error: ld returned 1 exit status
issta21-322@STT202122:~/Static-Analysis/Examples$ clang --analyze trial-teaching.c
trial-teaching.c:18:12: warning: Division by zero [core.DivideZero]
kappa=kappa/(a < 0);
trial-teaching.c:19:12: warning: Division by zero [core.DivideZero]
kappa=kappa/(!(a < 0));
2 warnings generated.
issta21-322@STT202122:~/Static-Analysis/Examples$
```

Fig 6.3.6.2 Output validation

In static-analysis when executing the file “trial-teaching.c” we got Division by Zero warnings. These warnings persisted on both the modes 1 and 2.

Conclusion

The document serves as a thorough overview of two important software testing tools: **Selenium** and **TrustInn**. The findings highlight that **Selenium** is an effective open-source framework for automating web browsers, which enhances test coverage, execution speed, and overall quality in web application development. However, the document also points out its limitations in handling dynamic content, synchronizing, and working with CAPTCHA. Conversely, **TrustInn**, a verification tool from NIT Warangal, is described as a complementary tool that provides a deeper layer of program analysis, which is useful for detecting logical errors and unreachable code, and for validating smart contracts. Overall, the report concludes that the development of innovative tools like these is vital for advancing test automation technologies and improving software reliability.

The document outlines several areas for future development and improvement, suggesting that the tested tools are a starting point for more advanced work.

- **Tool Reconstruction:** The report explicitly states that the tools, particularly those related to TrustInn, should be "**reconstructed again**" to resolve various errors and ensure consistent performance across different systems.
- **Version Control Integration:** The report recommends integrating version control systems like Git into the Selenium automation framework. This would allow for better tracking of changes, easier collaboration among testers, and the ability to roll back to previous versions of test scripts.
- **Cloud Features:** To enhance scalability and efficiency, the document suggests incorporating cloud features into Selenium. This would enable parallel test execution across various browsers and devices without the need for physical infrastructure, significantly reducing overall testing time.
- **Issue Resolution:** Specific technical issues need to be addressed, including date format errors, "**no file or directory**" errors, and the "**Division by Zero**" warnings encountered during testing with various tools.

References

[1] GeeksforGeeks - Software Testing Tutorials

<https://www.geeksforgeeks.org/software-testing/>

[2] Selenium Official Documentation available at

<https://www.selenium.dev/documentation/>

[3] Selenium IDE Documentation available at

<https://www.geeksforgeeks.org/software-testing/selenium-ide/>

[4] BrowserStack - Selenium Grid Guide available at

<https://www.browserstack.com/guide/selenium-grid-tutorial>

[5] CBMC (C Bounded Model Checker) Official Documentation available at

<https://diffblue.github.io/cbmc/>

[6] Solidity Documentation available at

<https://www.tutorialspoint.com/solidity/index.htm>