

**American International University-Bangladesh (AIUB)**

# Title : Enhancement of fuzzing testing method on web apps.

**Submitted By**

|  |  |
| --- | --- |
| **19-40279-1** | **RAHMAN, MD AKIB UR** |
| **19-39735-1** | **SUPRIYA OSMAN** |
| **19-39364-1** | **NABANITA SAHA DOLA** |
| **19-39547-1** | **TAMANNA MAHEJABIN MOULY** |

**Department of Computer Science**

**Faculty of Science & Technology (FST)**

**American International University-Bangladesh**

|  |
| --- |
| Declaration |

We certify that the work presented in this thesis entitled **“Enhancement of fuzzing testing method on web apps”** is original and our own. This work is totally done by ourself. We have also cited all sources used in the preparation of this thesis and have given due acknowledgment to all contributions made by others.

We take full responsibility for any errors and want to ensure that the paper represents our own work and that it has been completed to the best of our knowledge and ability.

Signed,

|  |
| --- |
| Approval |

This thesis titled **“Enhancement of fuzzing testing method on web apps”** has made a significant contribution to the field of web application security and finally it has been submitted to our respected faculty members.

|  |
| --- |
| Acknowledgment |

We would like to express our sincerest respect to all those who have supported us throughout our academic journey, especially in the completion of this thesis report on **“Enhancement of fuzzing testing method on web apps”**. We would like to extend my heartfelt thanks to our advisor **Sazzad Hossain** sir, for his guidance, encouragement, and invaluable insights. We are also grateful to the Faculty of Science & Technology (FST), for their support, constructive feedback, and helpful suggestions. This thesis would not have been possible without their support and belief in us. Finally, we dedicate this work to all those who have inspired us to pursue our dreams and never give up.

Table of Contents

[Title : Enhancement of fuzzing testing method on web apps. 1](#_Toc130086304)

[Declaration 2](#_Toc130086305)

[Approval 3](#_Toc130086306)

[Acknowledgment 4](#_Toc130086307)

[Abstract 6](#_Toc130086308)

[Chapter 1: Introduction 6](#_Toc130086309)

[1.1 Web application 6](#_Toc130086310)

[1.2 Overview of common vulnerabilities in web application 7](#_Toc130086311)

[1.3 Fuzzing testing on web applications 7](#_Toc130086312)

[1.4 Motivation of the work 8](#_Toc130086313)

[1.5 Research Question: 8](#_Toc130086314)

[1.6 Research Objective: 8](#_Toc130086315)

[Chapter 2: Literature Review 9](#_Toc130086316)

[2.1 Introduction 9](#_Toc130086317)

[2.2 Related Works 9](#_Toc130086318)

[2.3 Data Collection 11](#_Toc130086319)

[2.3.1. Types of software testing: 11](#_Toc130086320)

[2.3.2. Testing technique: 13](#_Toc130086321)

[2.3.3. Fuzz testing 14](#_Toc130086322)

[2.3.3.1 Fuzz Process: 16](#_Toc130086323)

[2.3.3.2 Fuzzing features 17](#_Toc130086324)

[2.3.3.3 Types of fuzzing 17](#_Toc130086325)

[2.3.3.4 Limitations of fuzzing 18](#_Toc130086326)

[2.3.3.5 Survey : 18](#_Toc130086327)

[Chapter 3: Methodology 21](#_Toc130086328)

[3.1 Our Approach 21](#_Toc130086329)

[3.2 Steps of enhanced fuzzing: 23](#_Toc130086330)

[Chapter 4: Results and Implementations 27](#_Toc130086331)

[4.1 Test cases for manual testing: 27](#_Toc130086332)

[4.2 Test cases for automated testing: 29](#_Toc130086333)

[4.3 Test Cases for traditional fuzzing system: 40](#_Toc130086334)

[4.4 By using automated tools: 40](#_Toc130086335)

[4.5 Case summary: 48](#_Toc130086336)

[4.6 Findings: 49](#_Toc130086337)

[Chapter 5: Discussion 50](#_Toc130086338)

[5.1 Fuzzing testing to update analysis 50](#_Toc130086339)

[Chapter 6: Conclusion 51](#_Toc130086340)

[References 51](#_Toc130086341)

|  |
| --- |
| Abstract |

This thesis's objective is to suggest and evaluate an enhanced fuzzing testing approach for web applications. Fuzzing is a popular testing technique for discovering software vulnerabilities by injecting unexpected and invalid inputs into the application. However, conventional fuzzing testing have certain limits. To address these challenges, this thesis presents a novel approach to fuzzing web applications that describes a term static analysis, by this we can work with both input fields and non-input fields together. To find the effectiveness of the enhanced fuzzing, we conducted experiments on a web application, and compare the results with traditional fuzzing methods. Results show that the enhanced method performs better than traditional fuzzing methods in terms of discovering new vulnerabilities and achieving higher code coverage.

|  |
| --- |
| Chapter 1: Introduction |

## Web application

A web application is a piece of software that runs on web servers and is accessed by web browsers on the internet. Web apps are intended to provide users with a rich, interactive experience that is accessible from any location that has an internet connection. E-commerce, banking, healthcare, education, and entertainment are just a few of the areas that use web applications. They have various advantages over typical desktop apps, including as quicker deployment, lower maintenance costs, and better accessibility.[1]

Online apps can be basic, like a single-page form that gathers user data, or complicated, like a multi-page platform that includes features like social networking, e-commerce, and real-time messaging. They are also highly accessible because they can be accessed from a variety of devices, including desktop computers, laptops, tablets, and smartphones.[2]

**Online applications are utilized in a wide range of industries, including:**

1.E-commerce makes extensive use of web applications for online shopping, payment processing, and inventory management.

2.Education uses web apps for online learning, virtual classrooms, and educational resources.

3.Web apps are used in the entertainment business to stream movies, TV shows, and music online.

4.Healthcare uses web apps for telemedicine, online medical records, and patient portals.

## Overview of common vulnerabilities in web application

Due to the complexity of modern web applications and the nature of the Internet, web applications are highly vulnerable to security flaws. Web applications frequently contain the following vulnerabilities:

XSS, or cross-site scripting: A vulnerability known as XSS enables an attacker to insert malicious code into a web page, which users who are not expecting it can use to their advantage. Theft of user data, such as login credentials, or other malicious actions can be carried out using this vulnerability.

Management of Sessions and Authentication: This vulnerability arises when an application fails to properly manage user authentication and session management. This can result in attacks like session hijacking, in which an attacker hijacks a user's session and attempts to guess the user's password or brute-force attacks.

Security Misconfiguration: When an application is not properly configured, it opens itself up to attack, resulting in this vulnerability. Leaving default passwords in place, making sensitive files available to unauthorized users, or using out-of-date software with known vulnerabilities are all examples of this.

Insufficient Validation of the Input: This weakness happens when an application doesn't as expected approve client input, permitting aggressors to infuse vindictive code or orders into the application. XSS and SQLi are two examples of this kind of attack.

## Fuzzing testing on web applications

From online banking to social media platforms, web applications have become an essential part of our day-to-day lives. In any case, with the rising reliance on web applications, security dangers have expanded. Subsequently, it is significant to test web applications completely to guarantee that they are secure and liberated from weaknesses. Fuzzing is a common testing method for identifying software application vulnerabilities, including web application vulnerabilities. Fuzzing is a useful testing technique for web applications because it can help find vulnerabilities that aren't always easy to find. Buffer overflows, injection flaws vulnerabilities can all be found with fuzzing.

## Motivation of the work

Web applications have become much important in a variety of industries, including e-commerce, healthcare, and finance, in recent years. However, the prevalence of web applications raises the risk of security flaws like SQL injection, which can have devastating effects on organizations as well as users. Fuzzing testing, which entails generating and feeding a large number of random or mutated inputs into a web application to identify any unexpected behaviors, is one effective method for identifying and preventing vulnerabilities of this kind.

Traditional fuzzing techniques are not enough to find all kinds of vulnerabilities in web apps. For instance, they frequently rely on inputs that are unable to fully investigate the behavior of the application. As a result, we try to investigate and improve the fuzzing testing method for web applications by investigating, contrasting, and assessing the efficiency of various advanced fuzzing techniques against various web application benchmarks. The proposed research has the potential to improve the security of web applications and produce testing tools for web developers and security analysts that are more dependable and efficient.

## Research Question:

* What are the challenges and limitations of the fuzzing testing method and how can they be addressed?
* How can the performance of the fuzzing testing method be improved?
* How can the fuzzing testing method be integrated into system testing?

## Research Objective:

**Q1. Challenges and limitations:** Fuzzing depends on random inputs, which can make it difficult to find certain types of bugs that are triggered by specific, non-random inputs. This is one of the challenges in fuzzing testing. Another limitation of fuzzing is it can’t replace the black box or white box testing approach because it works with only input fields.

**Q2. Fuzzing testing improvement steps:** Instead of relying on random inputs we have implemented a term which is static analysis. Static analysis activity:

System Under Test (SUT) analysis 🡪 Identify each module 🡪 Identify input & non-input fields.

**Q3. Integration of fuzz testing in a black box or white box testing:** Fuzzing works only on input fields so added a non-input field in the fuzzing process so that the overall software testing process can be integrated and we can work both with fuzzing and black box or gray box testing together.

|  |
| --- |
| Chapter 2: Literature Review |

## Introduction

Here we have explored and analyzed the existing body of knowledge on our research topic. Our goal is to identify the key themes and trends in the literature, as well as any gaps in our understanding of the topic. To do this, we will gather and critically evaluate relevant sources, including academic journal articles, books, conference proceedings, and other sources. By synthesizing this information, we will gain a comprehensive understanding of what is known about the topic, and identify areas where further research is needed. This literature review will serve as the foundation for our research, providing a solid understanding of the state of knowledge and helping us to formulate a clear and well-informed research question.

## Related Works

Compared to previous studies varied explanations have been given:

V. J. M. Manes et al. [4] noticed a disturbing trend in the development of fuzzing techniques -fuzzing software and their design decisions are often poorly documented. To correct this, the authors felt it was relevant, to sum up, the state of the research as a whole and create a generalized framework. Finally, the authors proposed a generic algorithm for fuzzing which was generic enough that it can be used in both grey, white or black box implementations. Finally, they provided a simple framework to understand the processes in which a fuzzer must pass in order to successfully fuzz a program.

Richard Amankwah et al. [5] defined that this paper evaluated software vulnerability detection methods and technologies. On the other hand, they provided some methods for detecting software vulnerabilities were discussed, including fuzzing, scanning, static analysis, CRED, and BRICK. Furthermore, they compared the strengths and weaknesses of different strategies. Finally, they discussed static analysis detection technologies, as well as their detection rates, which were statistically compared.

Hossain Shahriar et al. [6] defined that, The three following techniques which are test case creation, oracle generation, and test case execution were used in this paper to provide an in-depth assessment of the automation support for these works. They also emphasize that few studies. figure the production of test cases based on software design languages such as UML, which can highlight security flaws. Finally, they are given a few tools for complete automation.

Willy Jimenez et al. [7] defined that an overview of software vulnerabilities as well as ways for preventing and detecting them. To construct a concept known as Vulnerability Detection Condition with the objective of automatically testing the source code to find vulnerabilities, the vulnerability model approach and the Vulnerability Cause Graph are taken into consideration as an input.

M. Bajer et al. [8] provided a brief description of software testing in research environment. It is divided into several parts and addresses the problem of testing devices not only from the source code perspective, but it also takes into consideration testing the complete system behavior. The paper describes mainly testing industrial automation devices and some methodologies dedicated for testing this type of devices. A significant part of the paper is related to unit testing and certification tests of device’s safety critical functions.

H.-M. Qian et al.[9] introduced software testing technology in host and target testing. They also provide a comprehensive software testing process model is put forward with the shortcomings of software testing V-model. The dynamic memory allocation and use are monitored to prevent the memory leak, and the quality of the software is further enhanced, the model has a practical value. In this research, some software testing tools is introduced, software testing automation is implemented, the coverage of system testing is collected, the dynamic allocation and free of memory is monitored, the memory leak is prevented.

Lj. Lazic et al.[10] provide some original solutions with regard to the deployment of the U.S. Department of Defense Simulation, Test and Evaluation Process (DoD STEP), using an automated target tracking radar system as a case study. Besides the integration of modelling and simulation, to form a model-based approach to the software testing process, the number of experiments, i.e. test cases, have been dramatically reduced by applying an optimized design-of-experiment plan and an orthogonal array-based robust testing methodology. This paper has aimed to raise awareness of simulation-based software testing combined with an optimized DOE and OART methodology that dramatically reduces the number of experiments, i.e. test cases.

W. Liu et al. [11] addressed a solution on security problems on IoT devices, and proposes a security architecture for IoT devices based on the trusted computing technology. They implements a security management system for IoT devices, which can perform integrity measurement, real-time monitoring and security management for applications, providing a safe and reliable execution environment and whitelist-based security protection for IoT devices. Finally they propose a prototype system which is implemented in a development board of embedded system (Xilinx ZC7010). They have provided the experimental results and scheme that has strong security and good  
performance.

J. Seo et al. [12] propose the software interface as two essential parts: interface function that represents the statement of communication between heterogeneous layers, and interface variable that represents software and/or hardware variable which are defined in different layer from integrated software and used to expected output for decision of fault. They mainly work on the test coverage, fault efficiency and accuracy in order to show the effectiveness of their proposed interface-based test method.

Although there have been numerous studies on this subject in the past, the main approaches that have received the most attention are the fuzzing test.

## Data Collection

### Types of software testing:

**Manual Testing:**

Manual testing is the practice of identifying flaws and problems in a software product. A tester assumes the role of an end-user and checks that all features are operational. The test cases are manually executed by the tester. Manual testing is the process of utilizing an application's functionality as an end-user. Manual testing involves a tester manually running tests on the software. This procedure is used to identify flaws or faults. Manual testing is the most fundamental sort of testing in the application under test.[13]

**Testing Process:**

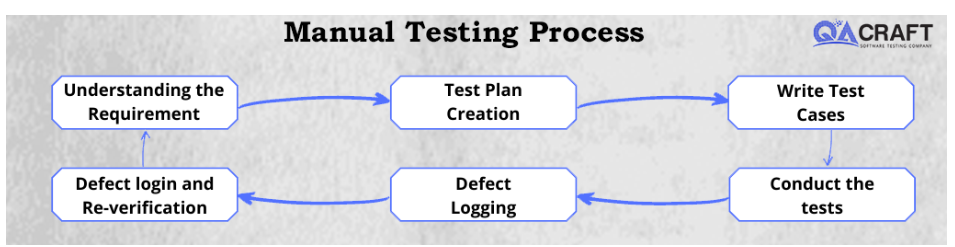


Fig : Manual testing process

**Automated Testing:**

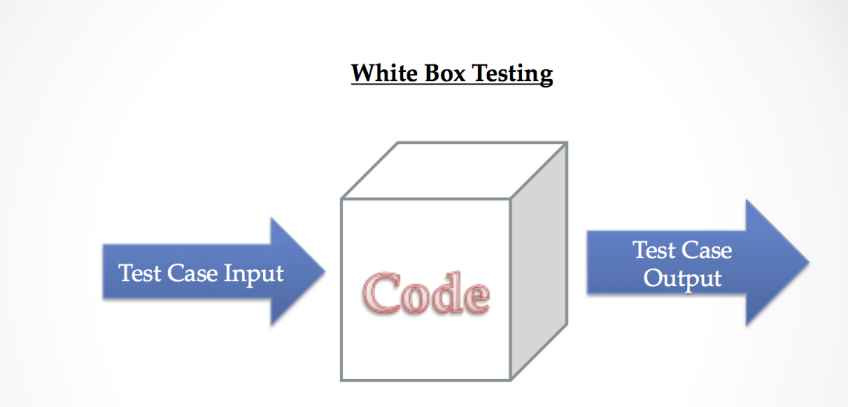
Using software tools and scripts, automated testing is the process of running pre-defined tests on a software application or system. By reducing the amount of manual effort required, expanding test coverage, and enabling faster feedback on the application's quality, automated testing aims to enhance testing's efficiency and effectiveness.

Unit testing, functional testing, regression testing, performance testing, and security testing are just a few of the many types of testing that can benefit from automated testing. In most cases, automated testing entails creating test cases that reenact user interactions with the application, automatically running these tests, and comparing the outcomes to what was expected. In comparison to manual testing, automated testing can offer a number of advantages, including increased speed, repeatability, accuracy, and scalability. By identifying defects early in the development cycle, it can also aid in lowering the overall cost of testing and improving software quality. However, automated testing also necessitates significant initial investments in test automation frameworks, infrastructure, and tools, as well as ongoing upkeep and updates.[14]

### Testing technique:

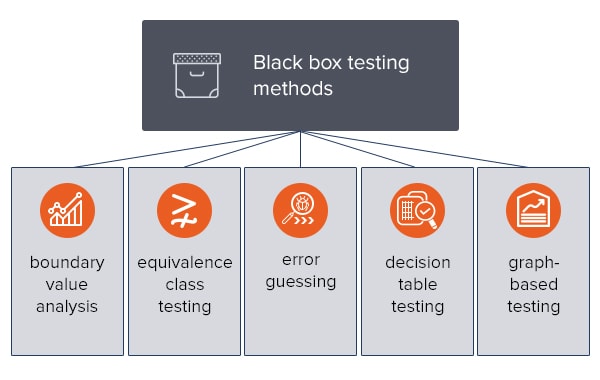
There are two traditional testing techniques for software testing. These are,

* White Box Testing
* Black Box Testing

**I. White-Box Testing** is a method of testing the internal structure, architecture, and coding of software to ensure input-output flow and to improve design, usability, and security. Because the code is visible to testers, white box testing is also known as clear box testing, open box testing, transparent box testing, code-based testing, and glass box testing. The term "Whitebox" was coined as a result of the see-through box concept. The term "clear box" or "Whitebox" refers to the capability of inspecting the software's outside shell.[15]

**Fig: white box testing**

**II. Black Box Testing** is a software testing method that tests the functionality of software applications without knowing the internal code structure, implementation details, or internal routes. Black Box Testing is totally based on software requirements and specifications and focuses on the input and output of software applications. It is also known as Behavioral Testing. The test engineers execute the black box testing.[16]

 Fig: Black box testing

### Fuzz testing

Testing technique

Fuzz Testing

Fuzz testing is a software testing techniques which is basically used to find unexpected behavior of a software.

##### Introduction:

Fuzz testing involves automatically feeding a program with a large amount of randomly generated inputs, called "fuzz", in order to find bugs, vulnerabilities, or other unexpected behavior. Fuzz testing is commonly used to test complex software systems, such as web applications or operating systems, and can help identify issues that might not be caught by traditional testing methods. Fuzz testing improve the overall quality and reliability of software by finding and fixing potential problems before they can cause harm or lead to security breaches.[1]

Fuzzing



System under test

Crush event monitor

Fig: Fuzz testing

System Under Test

Random data acting to input

in SUT

Fig: fuzz testing

#### Fuzz Process:

Fig: fuzzing steps

1. Identify target system: Determine the specific system or component that will be subjected to fuzz testing.
2. Identify inputs: Determine the specific inputs that will be fuzzed, such as file formats.
3. Generate fuzzed data: Use a fuzzing tool or generate random data to be used as inputs for the test.
4. Execute the test using fuzz data: Run the test with the fuzzed inputs and monitor the system's behavior.
5. Monitor system behavior: Observe the behavior of the target system and look for any crashes, exceptions, or other unexpected results.
6. Log defects: Document any defects or issues discovered during the test and classify them based on severity.

#### Fuzzing features

1. Fuzzing can find both known and unknown bugs: Fuzzing can discover both known bugs that have been documented in the software's code or documentation, as well as unknown bugs that have not yet been discovered. It can also help to find bugs that are specific to certain configurations or input, even if the bugs do not appear when using other inputs.
2. Fuzzing can be automated: Fuzzing can be automated using different fuzzing tools that are available. These tools can generate input, feed it to the system, and analyze the results.

Overall, fuzzing is a powerful technique that can help to improve the security and reliability of software systems by identifying and mitigating bugs and vulnerabilities.

#### Types of fuzzing

**Random fuzzing**: This is the most basic form of fuzzing, where random data is generated and fed into the system. This can help to find bugs that are caused by unexpected input, but it is not very efficient and may not discover all possible bugs.

**Mutation-based fuzzing**: In this approach, valid input is taken as a seed and then small mutations are made to it, such as changing specific bytes or adding new data. This can help to find bugs that are caused by slight variations in input.

**Coverage-based fuzzing**: In this approach, fuzzer is aware of the code coverage of the input it has generated, and it uses this information to guide the generation of new inputs. This can help to find bugs that are located in code that has not been executed during previous fuzzing runs.

Fuzzing can be **done manually**, or using **tools.**

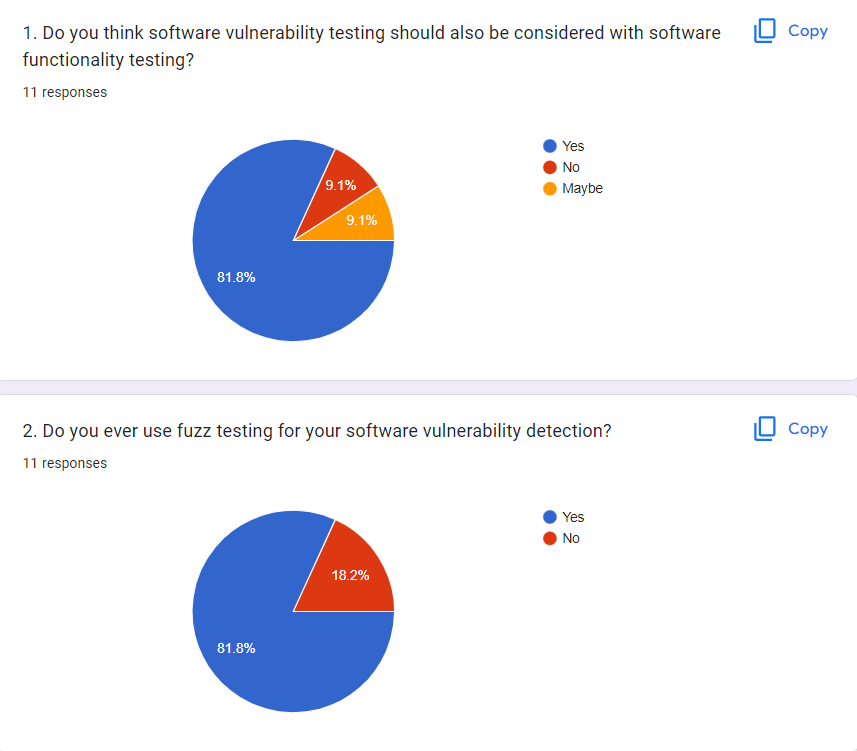
#### Limitations of fuzzing

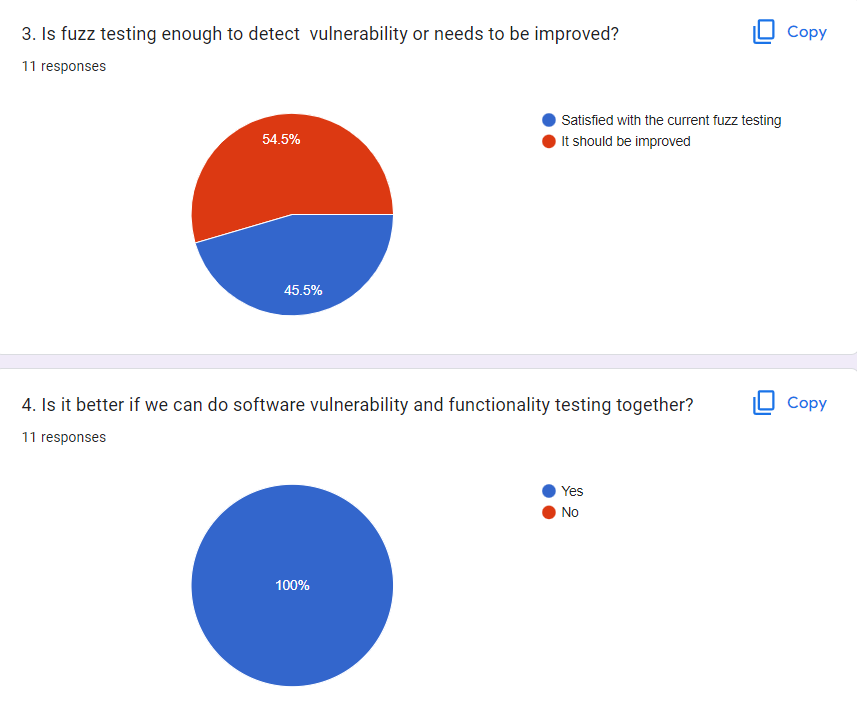
Fuzzing has its limitations: Fuzzing is not a silver bullet and it has its limitations. Fuzzing can only find bugs that are caused by unexpected input.

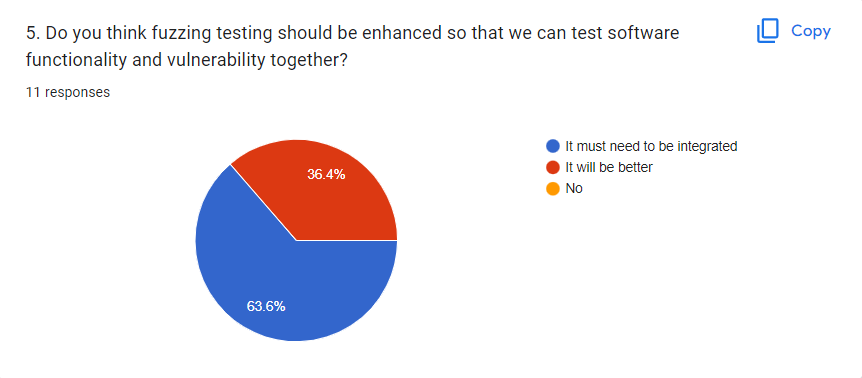
1. Limited coverage of bugs: Fuzzing **depends on generating random inputs**, which can **make it difficult to find certain types of bugs** that are triggered by specific, non-random inputs
2. Limited scope of security vulnerabilities: Fuzzing is typically used to **find input-based vulnerabilities**, such as **buffer overflows or SQL injection**. However, there are many other types of vulnerabilities, such as memory corruption or race conditions, that fuzzing may not be able to detect.

#### Survey :

To conduct our methodology firstly we have made a survey based on the fuzzing system to get feedback from QA testers to get their feedback about fuzzing testing. Here we provided the survey question with responses.







|  |
| --- |
| Chapter 3: Methodology |

## Our Approach

Fuzzing



System under test

Static analysis

Crush event monitor

**Fig: enhanced fuzz testing**

This diagram is almost like normal fuzzing. But we have bring a extra term which is static analysis. In static analysis there are sub steps, software analysis 🡪 Identify each modules 🡪 Identify input fields and non input fields.

Fig: Static analysis steps

We can see in static analysis, first term is software analysis. Second term is identify each modules and finally there is identify input and non-input fields.

By this term static analysis, we are able to find more bugs than normal fuzzing. We don’t need to depend only on input fields , we can work with non-input fields also. The most important advancement is we can integrated fuzz testing with testing techniques like black box testing.

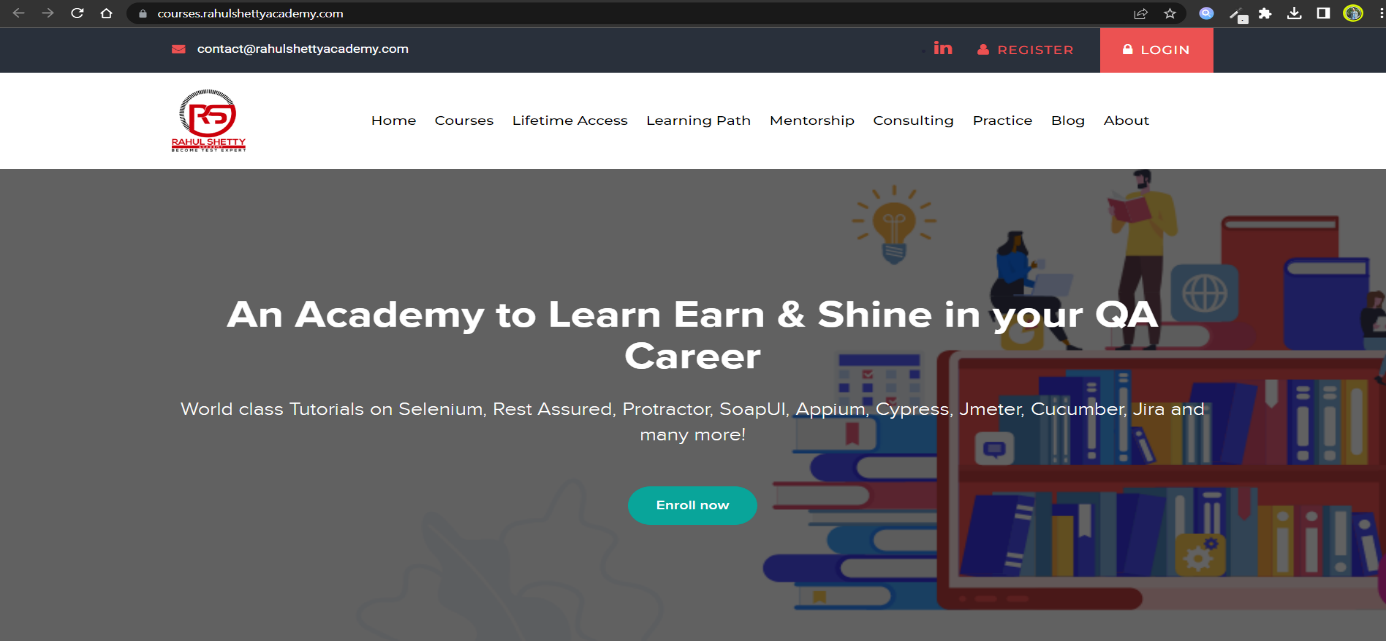
## Steps of enhanced fuzzing:

****

Fig: Enhanced fuzz testing process

Step 1. Software analysis:

In this step, we will gain detailed knowledge about how the software works, what functionality it has, what modules it has. Suppose I want to test the below web based software, which is https://courses.rahulshettyacademy.com/



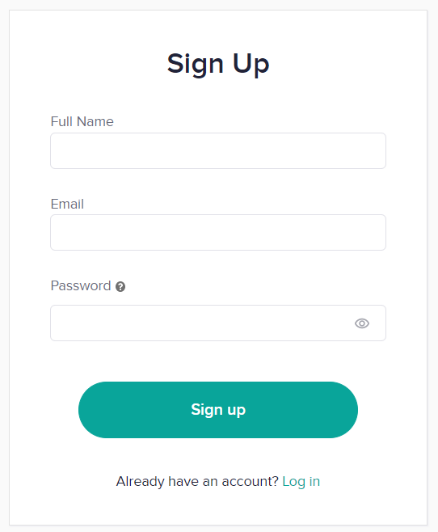
So, first step is to gain knowledge about this website. It has one navigation bar in the top section, inside the navigation there has login and registration. Then there has a menu bar. In the body section, there has a enroll now option. By this way we have to explore each part of the website.

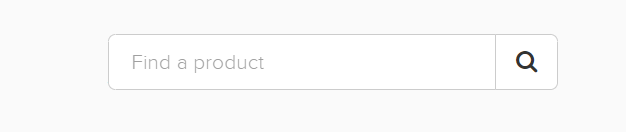
Step2. Identify each modules:

Instead of relying on random inputs, first of all we will identify different modules of a software so that we can customize the inputs based on the type of different module of the software and potential bugs and design specific test cases to cover the combinations of inputs that are likely to cause problems.

Step3. Identify input fields:

In step 2, we have already identified all the modules of the software. So, in this step we will identify the input fields from that modules. Because one of the key steps in a fuzzing process is identifying the input fields of the software that will receive the random or unexpected inputs. Input fields are areas within the software application where user input is expected or required, such as text boxes, drop-down menus, or checkboxes. By identifying these input fields, a tester can determine which areas of the software are most likely to be vulnerable to unexpected input or malformed data, and can focus their fuzzing efforts on those areas.

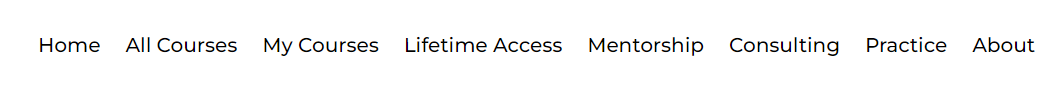




Step 4 : Identify non-input-fields:

In step 3, we have already identified the modules where user can take input. So, in this step, our task is to identify the rest of modules that doesn’t exist input fields.

Such as, menu bar.



Step 4 : Generate fuzz data

In this step, our task is to generate test cases for each of the modules that we identified in step 2 and step 3. That means, we will generate test cases for both input fields and non-input fields.

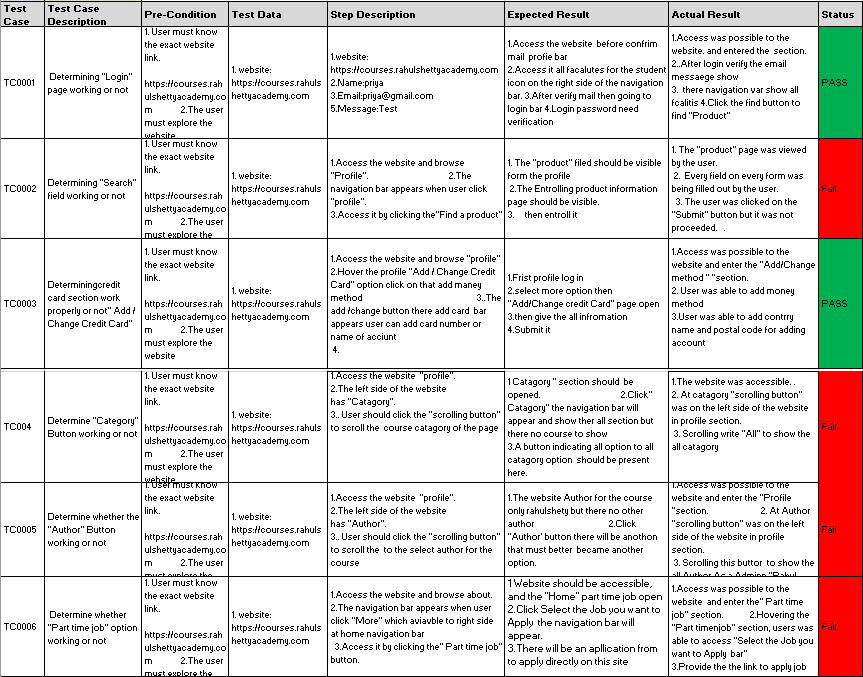
Step 5 : Execute the test

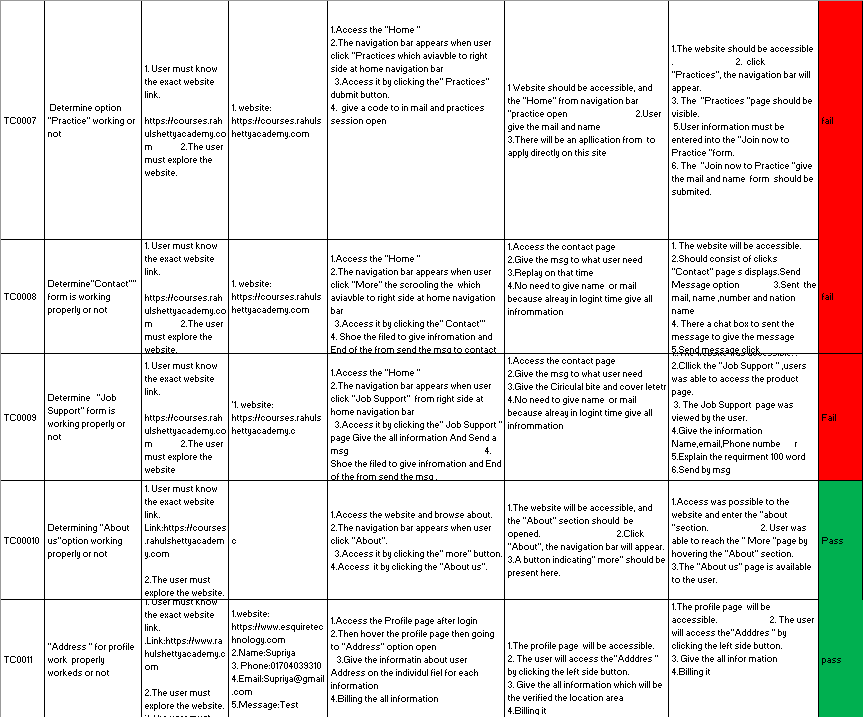
This step, involves execute the predefined step 4 in order to verify the functionality, performance, or other characteristics of a software application or system. The test case specifies the expected result for each step, and the actual result is compared to the expected result to determine if the test has passed or failed.

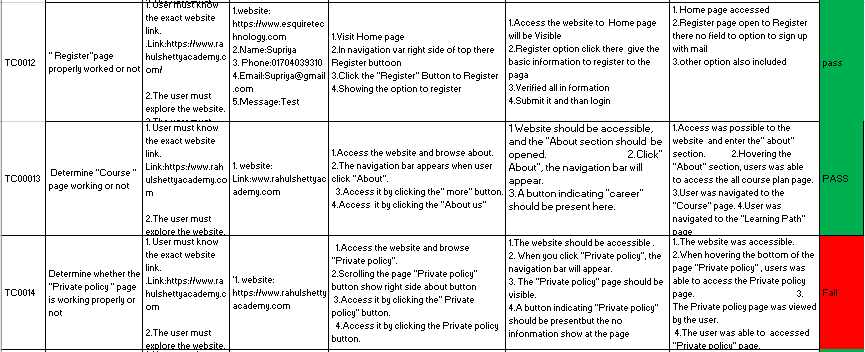
Before executing a test case, it is important to ensure that all necessary test data and prerequisites are in place. This may involve setting up test environments, configuring test tools, or creating test data sets. Once the test environment is ready, the tester can execute the test case by following the defined steps and recording the results.

|  |
| --- |
| Chapter 4: Results and Implementations |

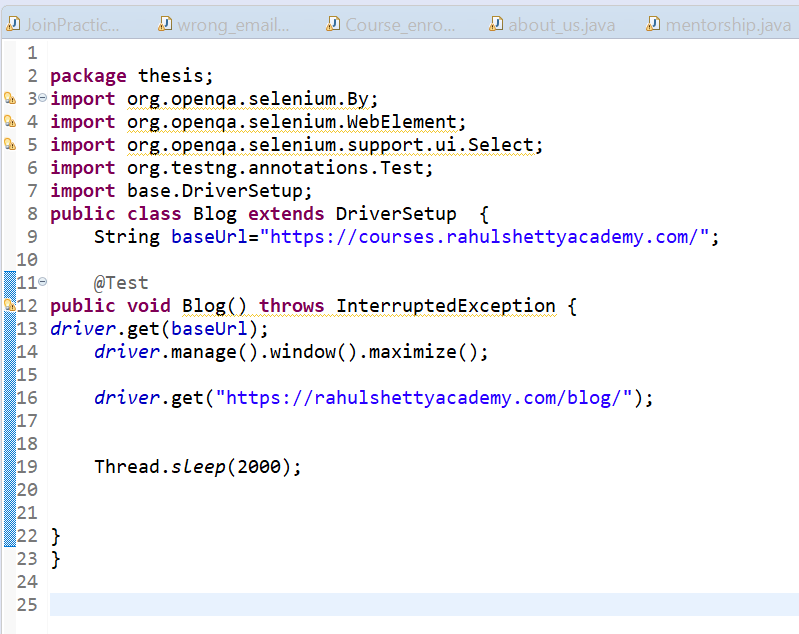
## Test cases for manual testing:

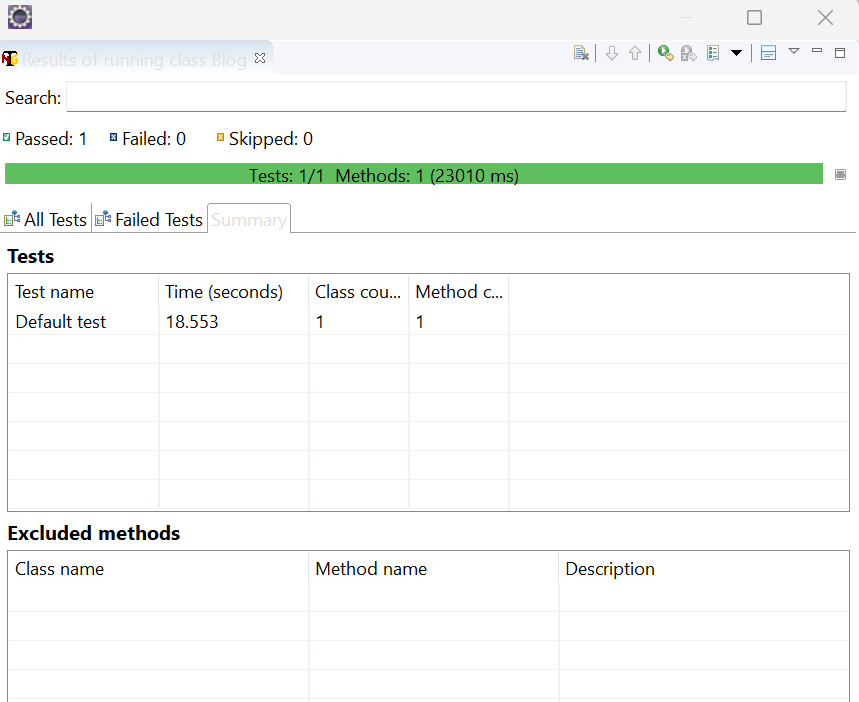


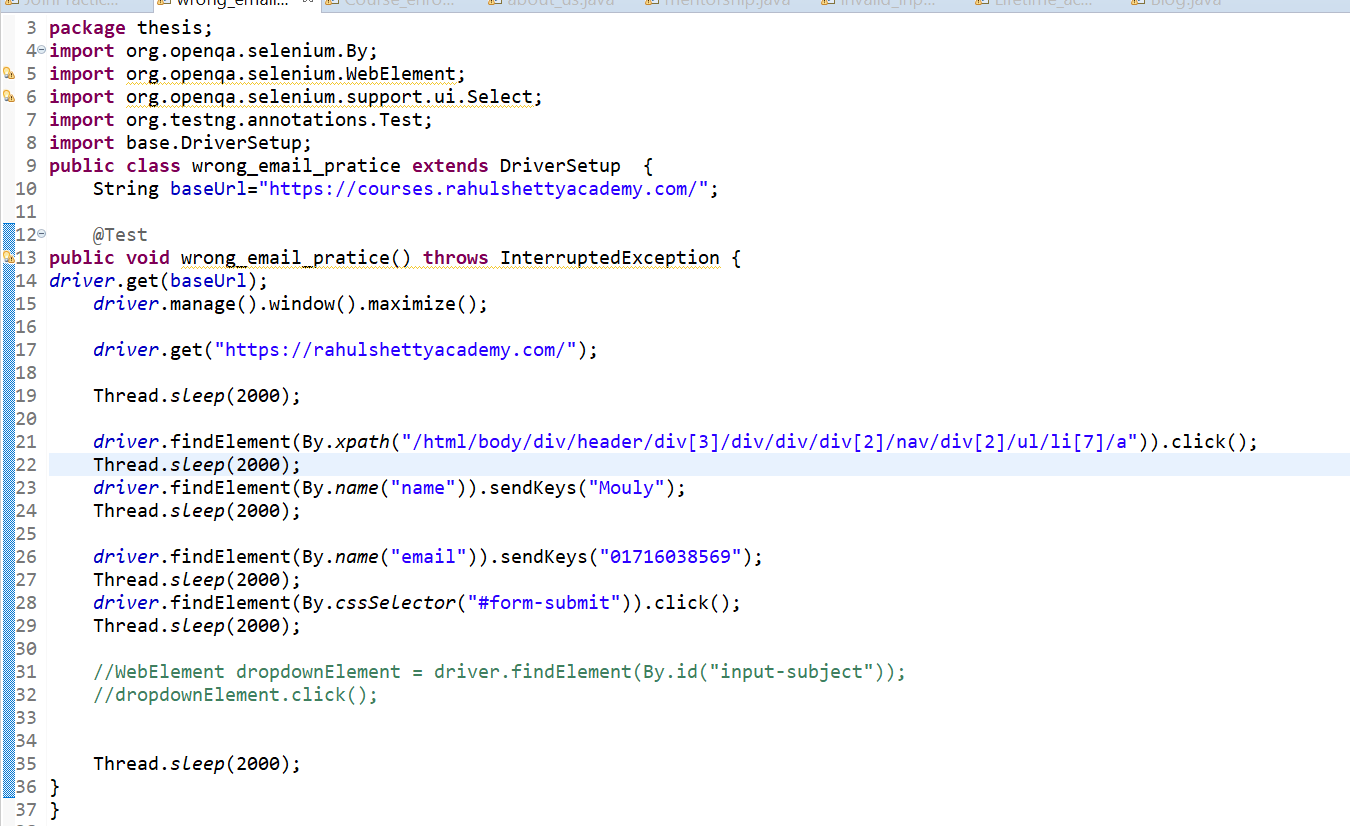


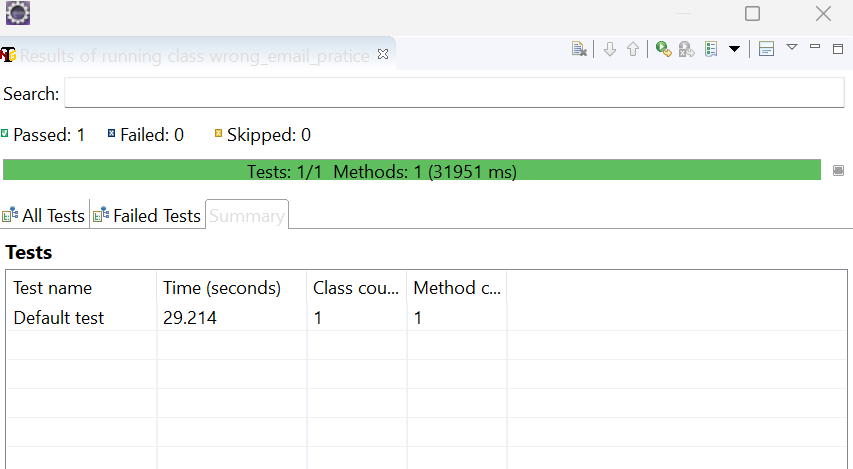


## Test cases for automated testing:

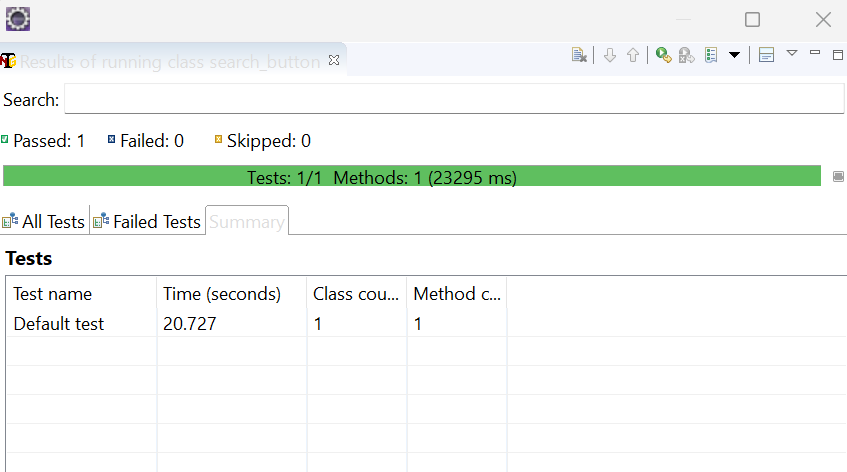


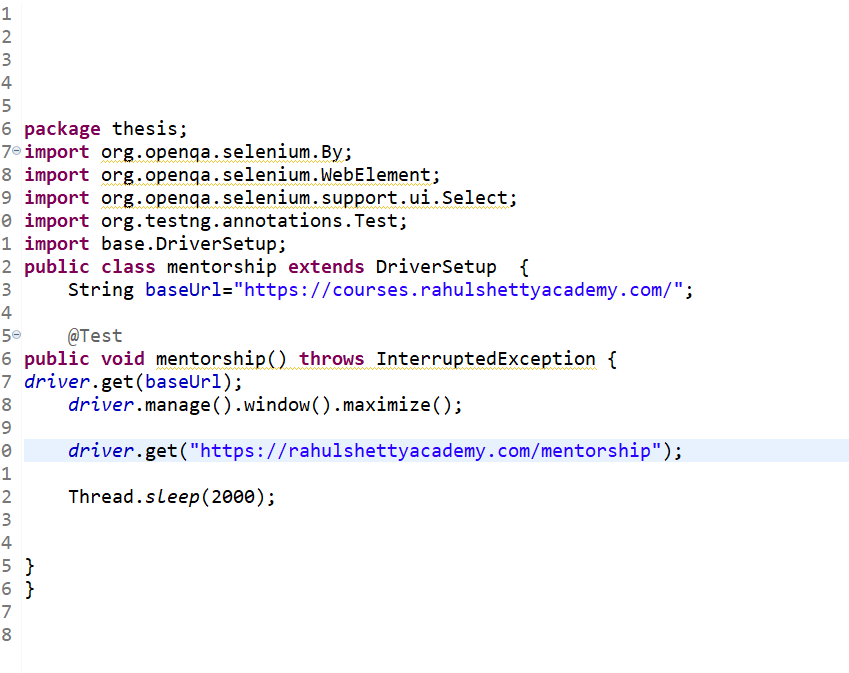


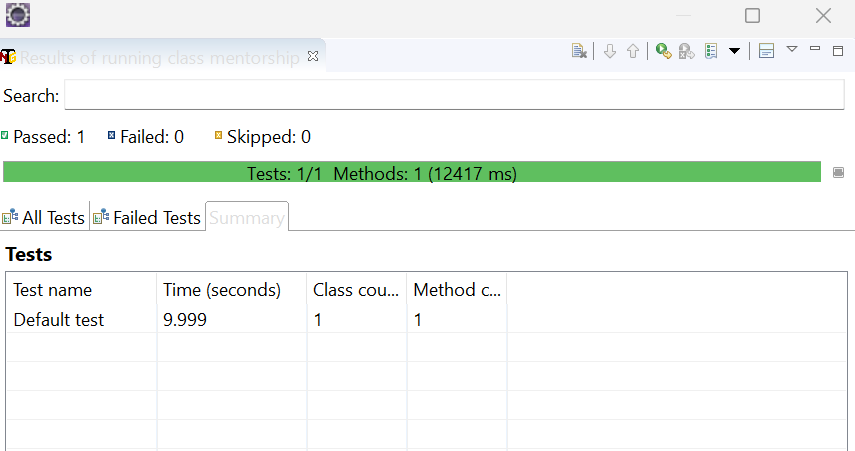




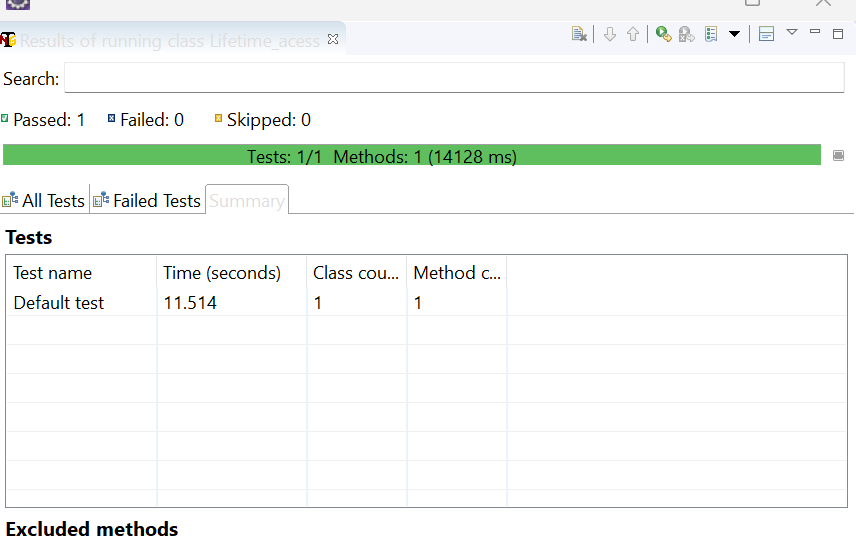


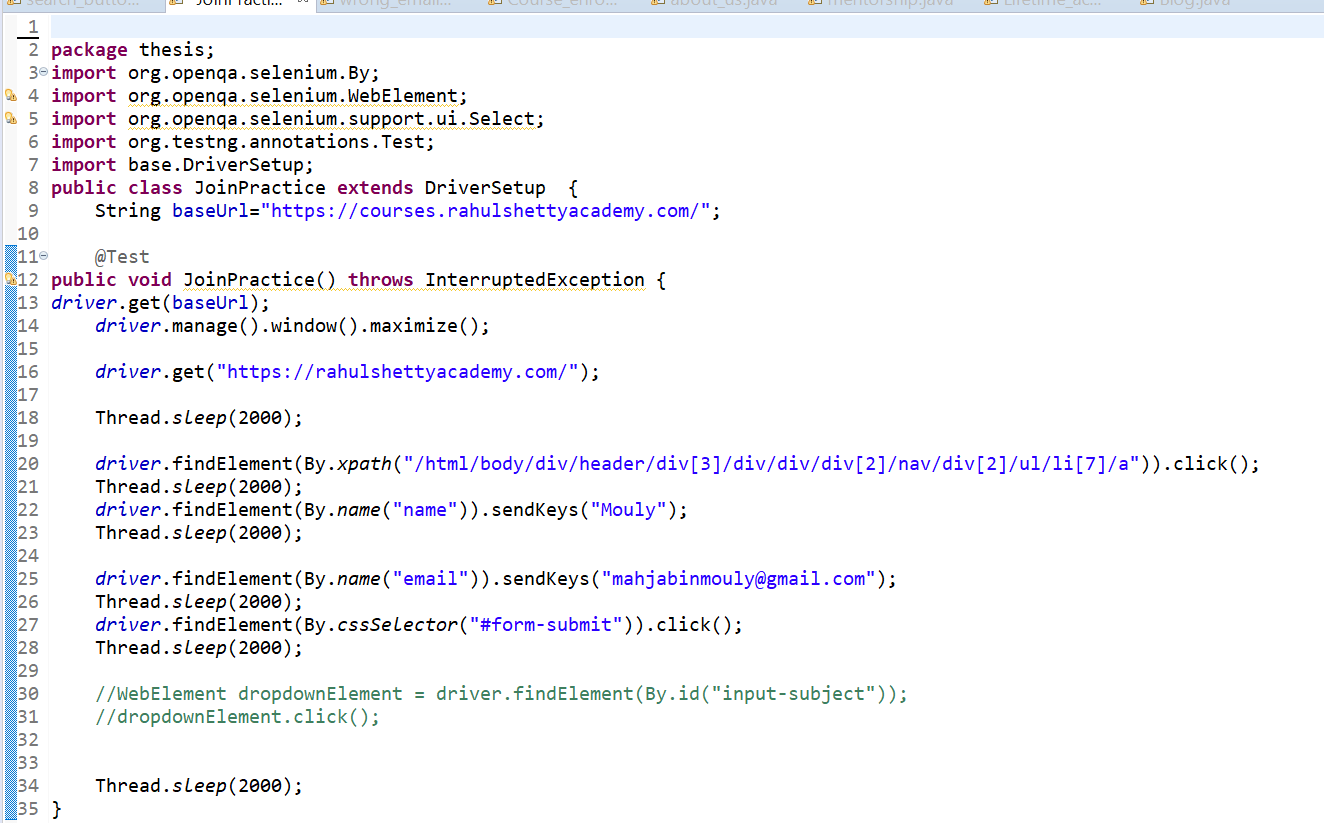


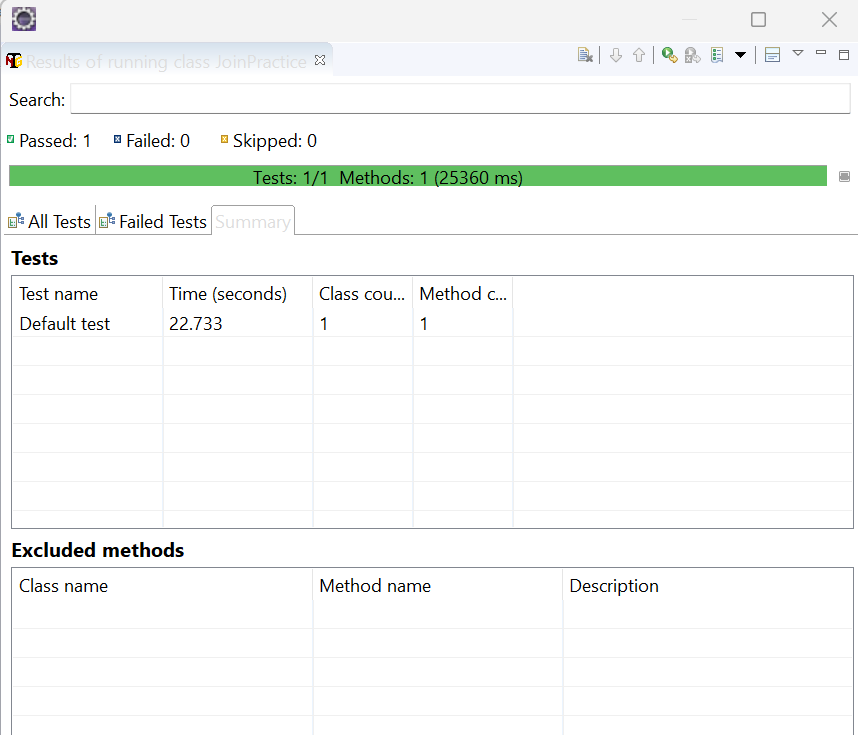




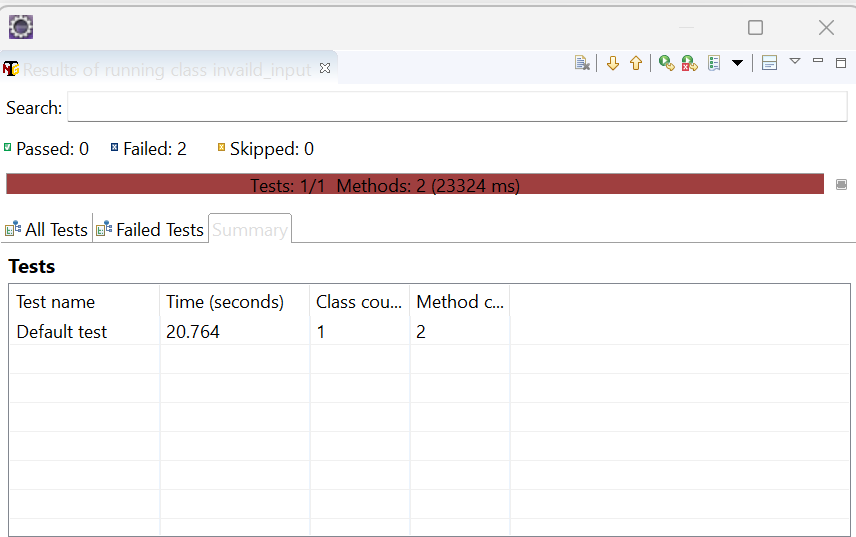


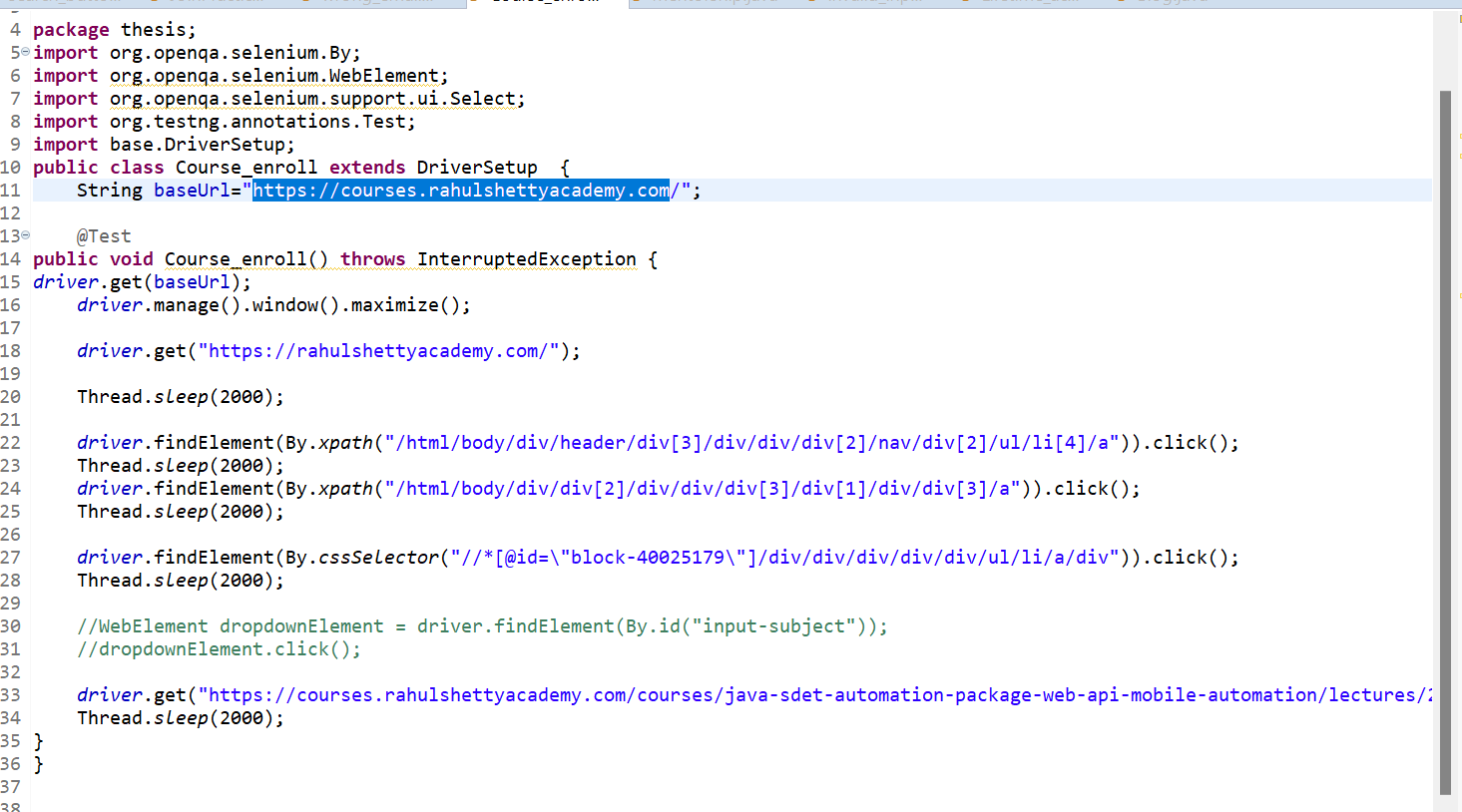


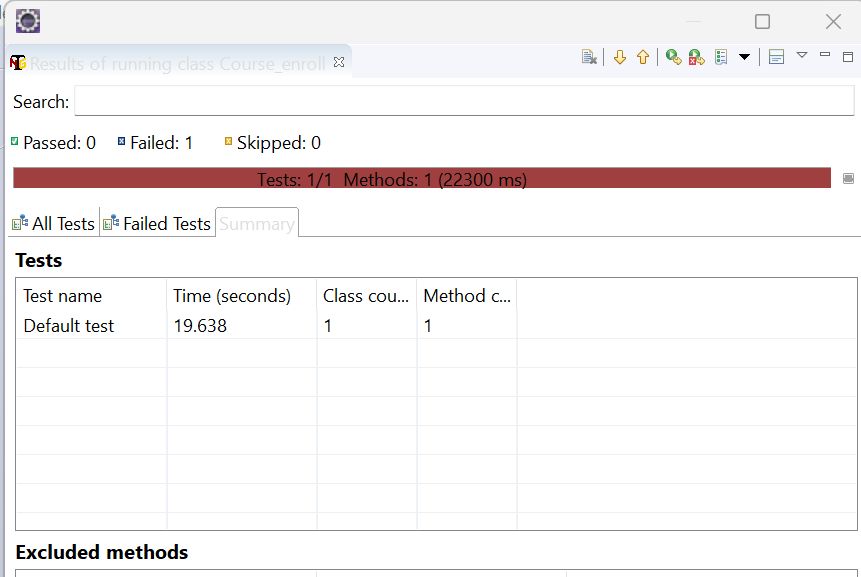


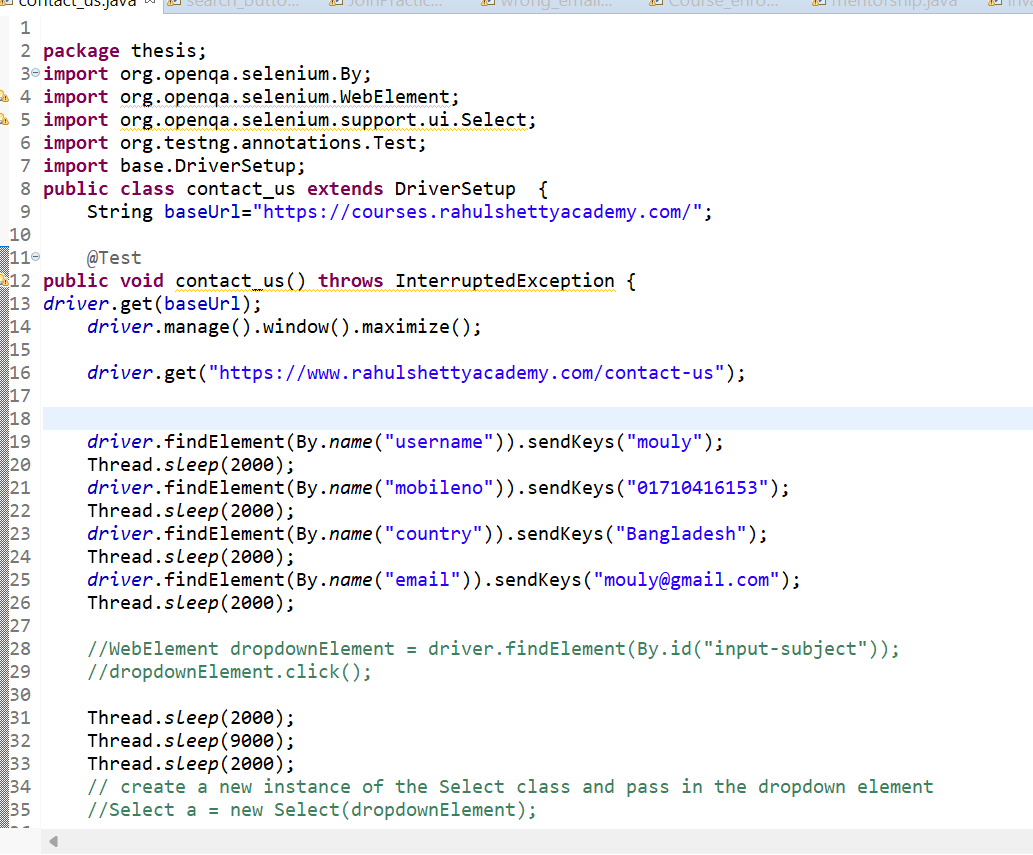


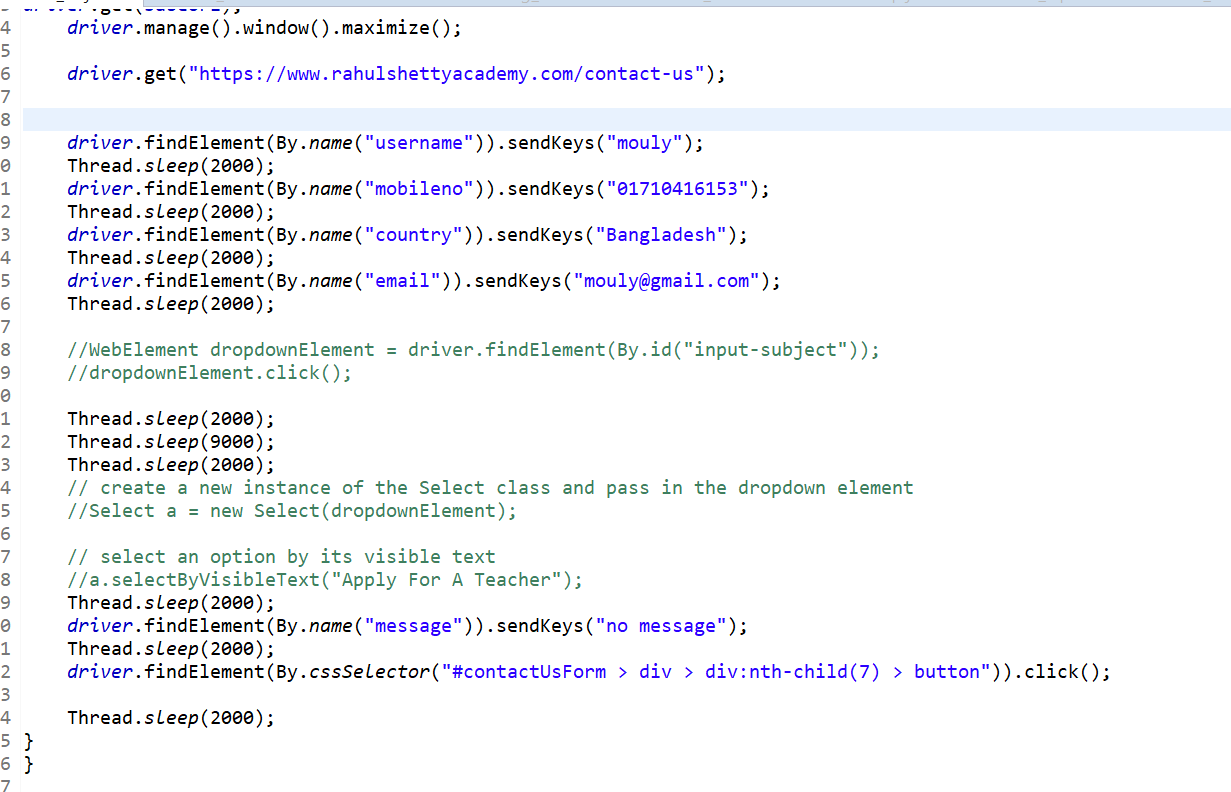


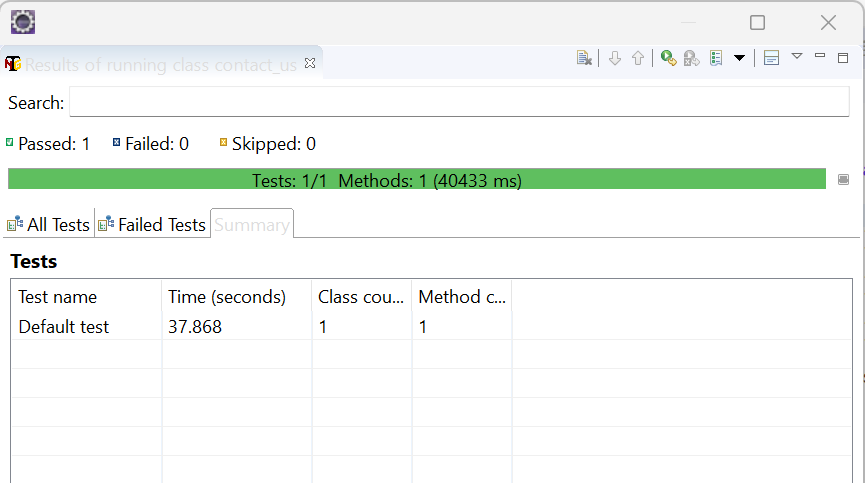


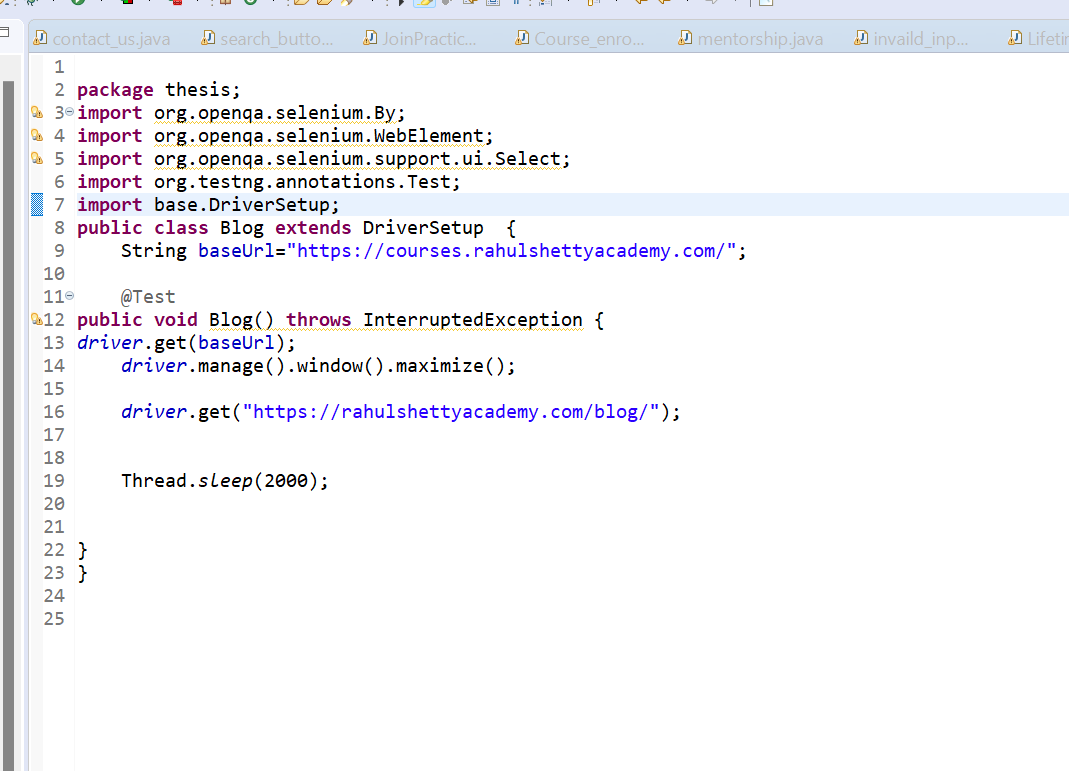


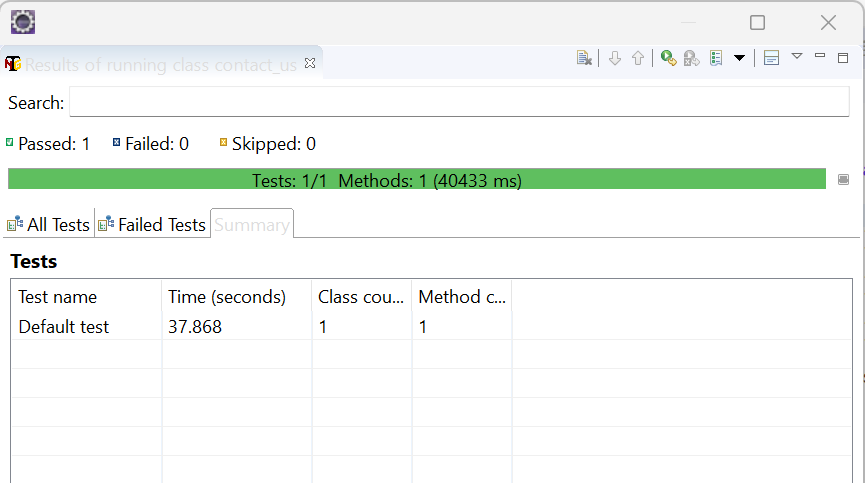




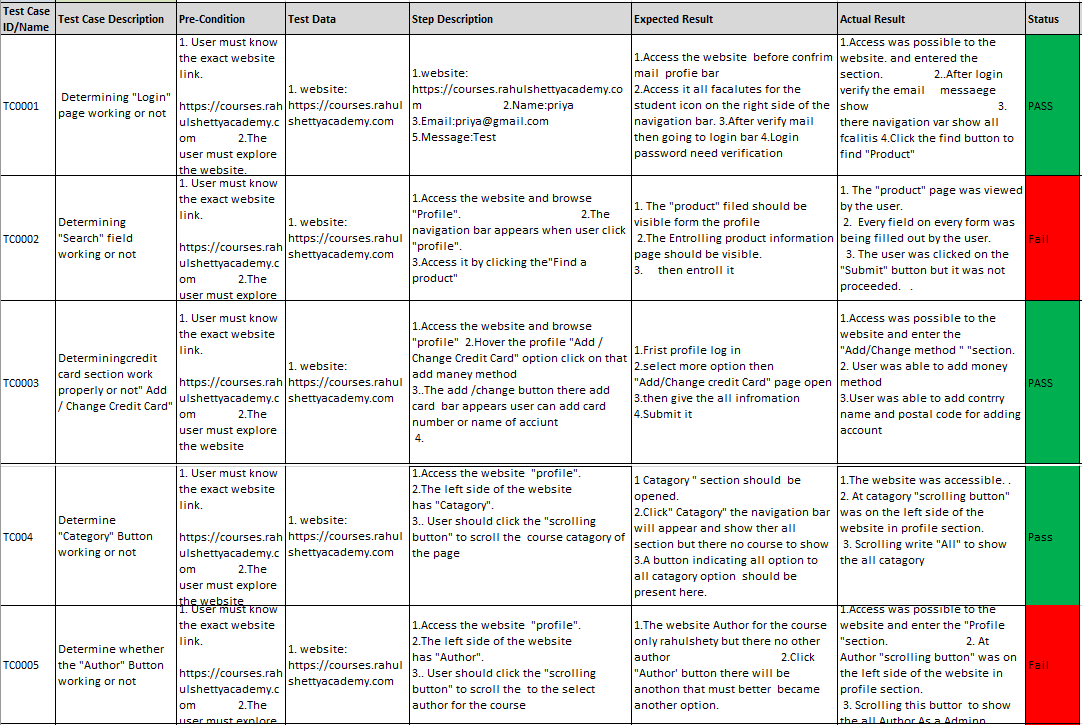




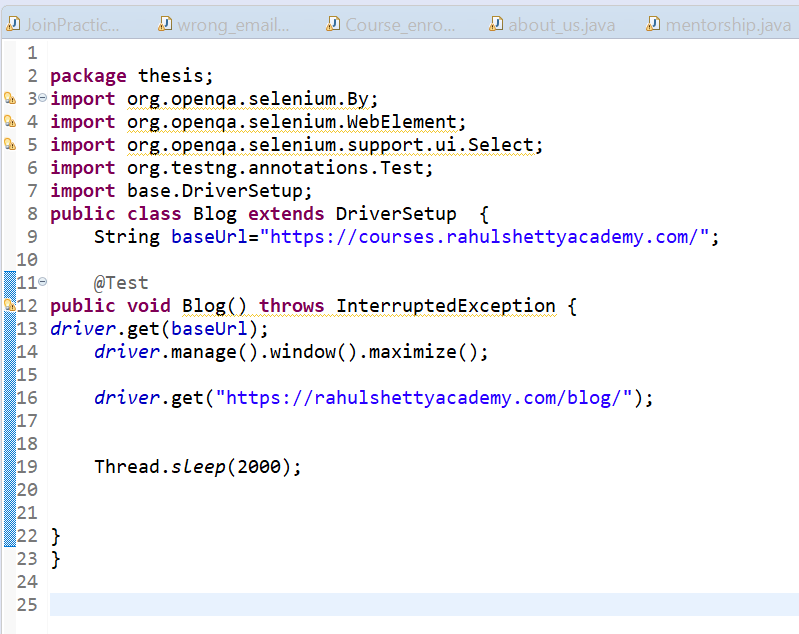


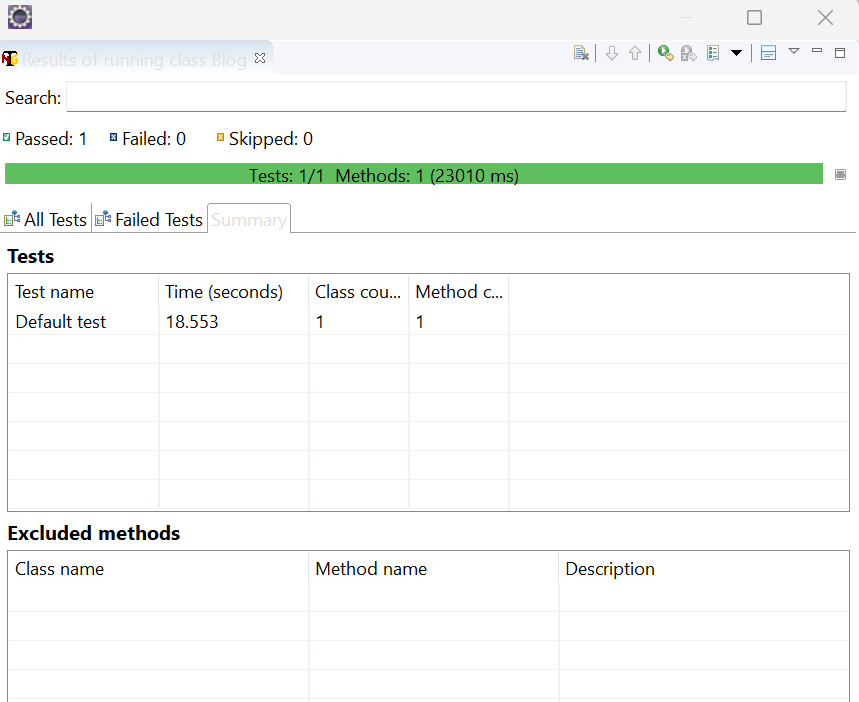


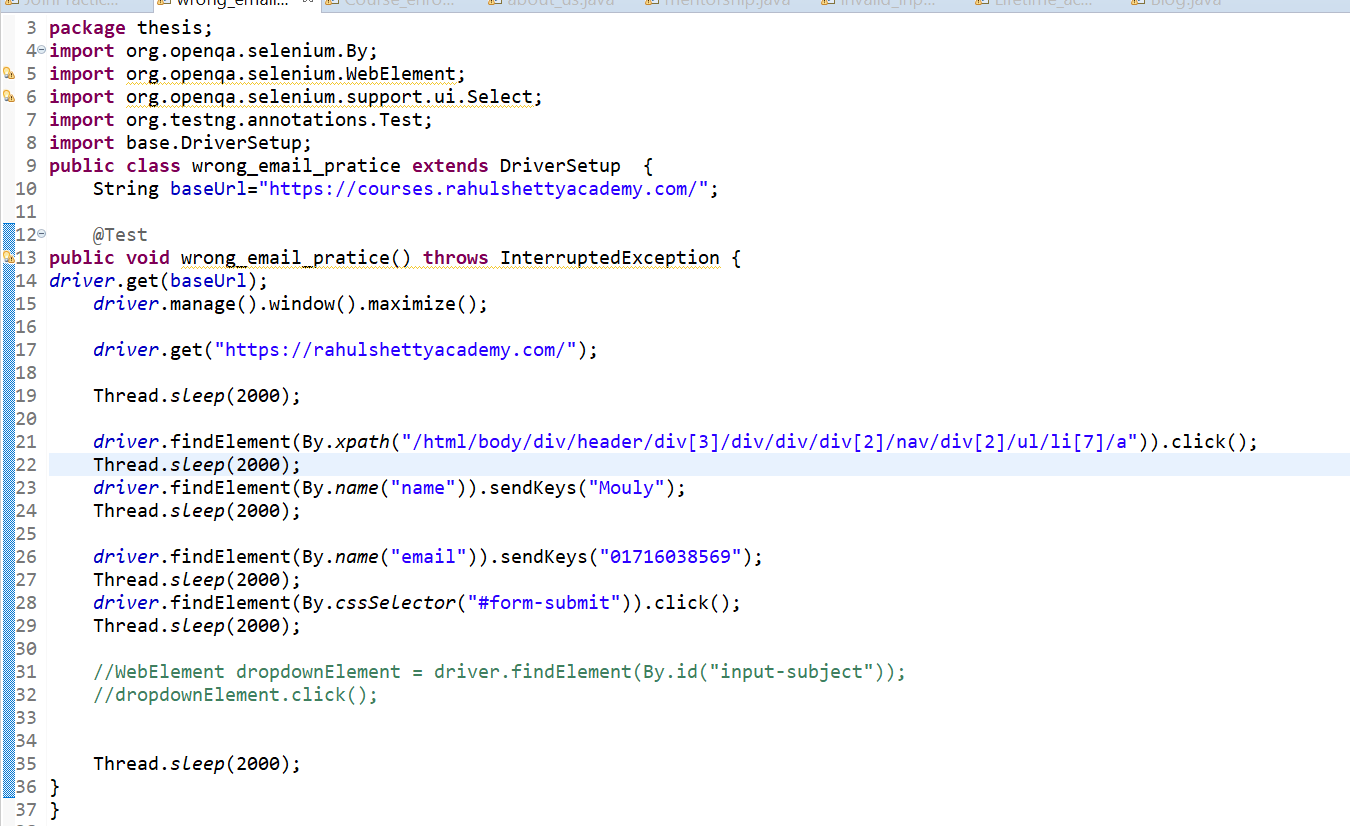
## Test Cases for traditional fuzzing system:

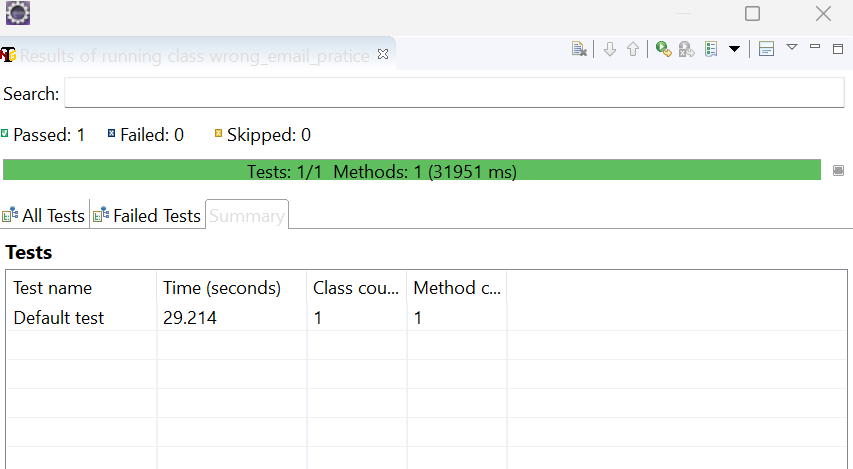


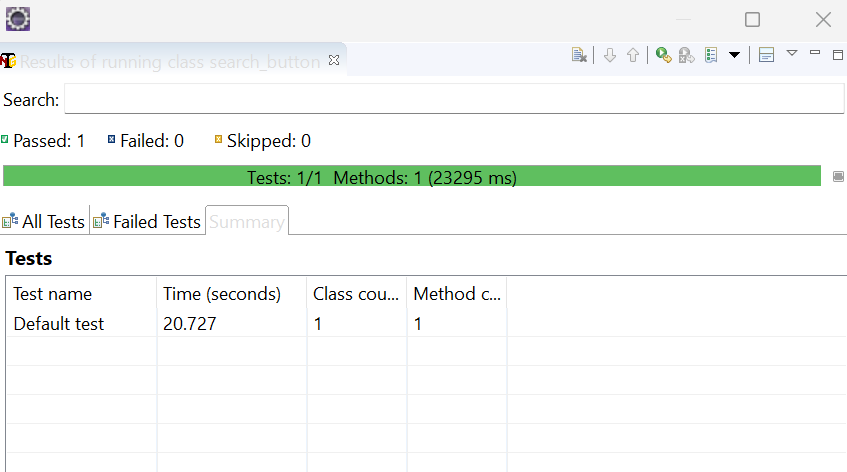
## By using automated tools:

R

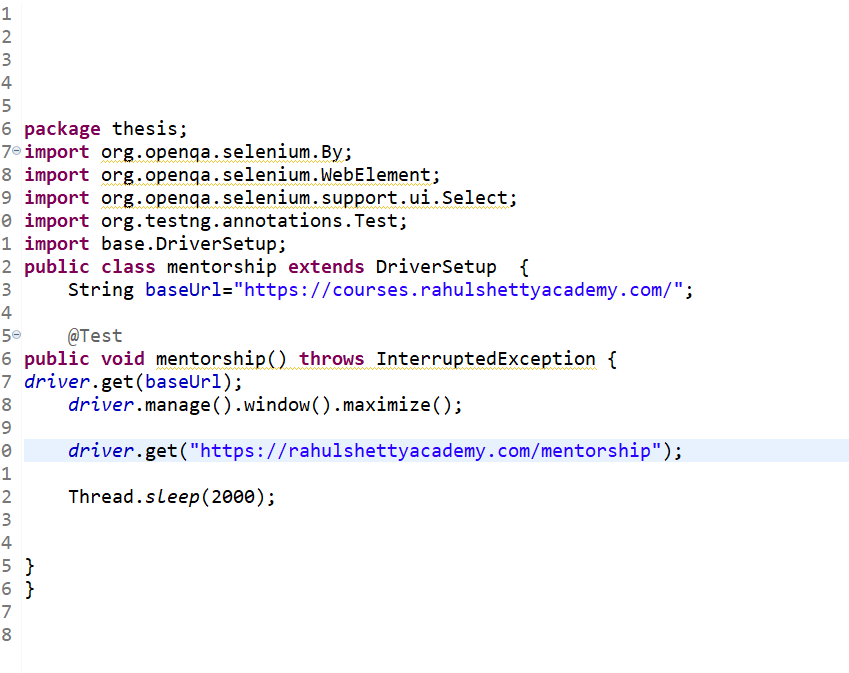


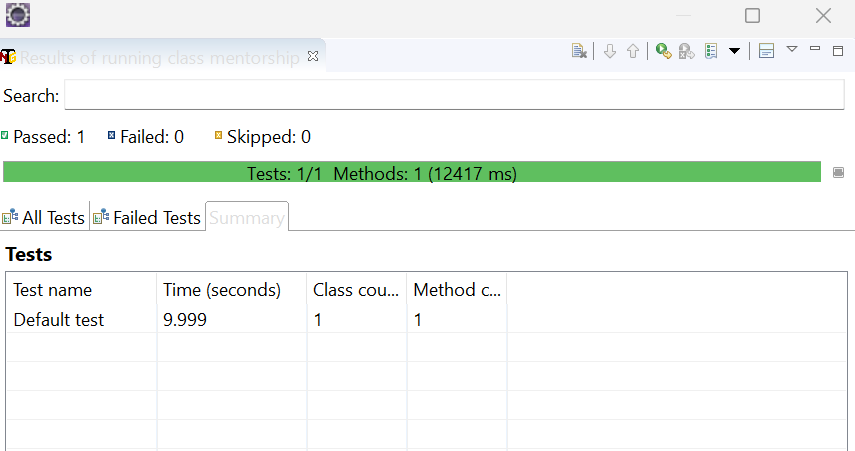




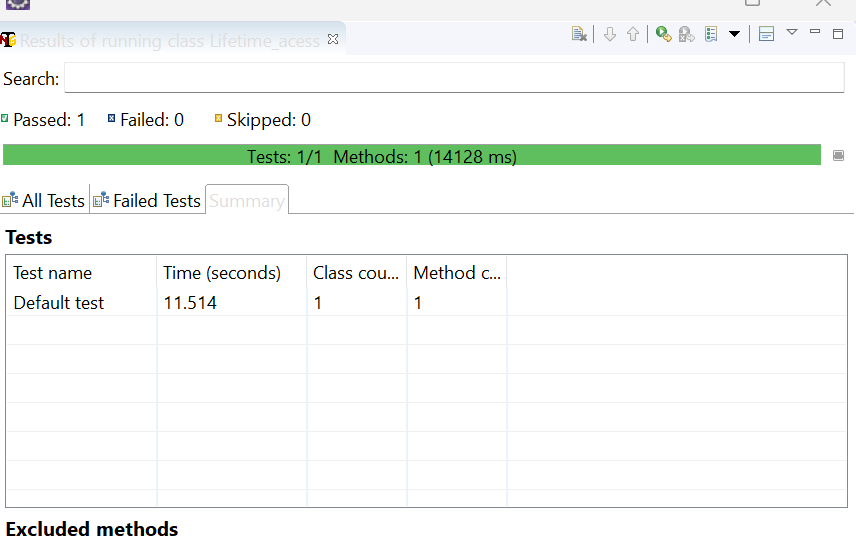


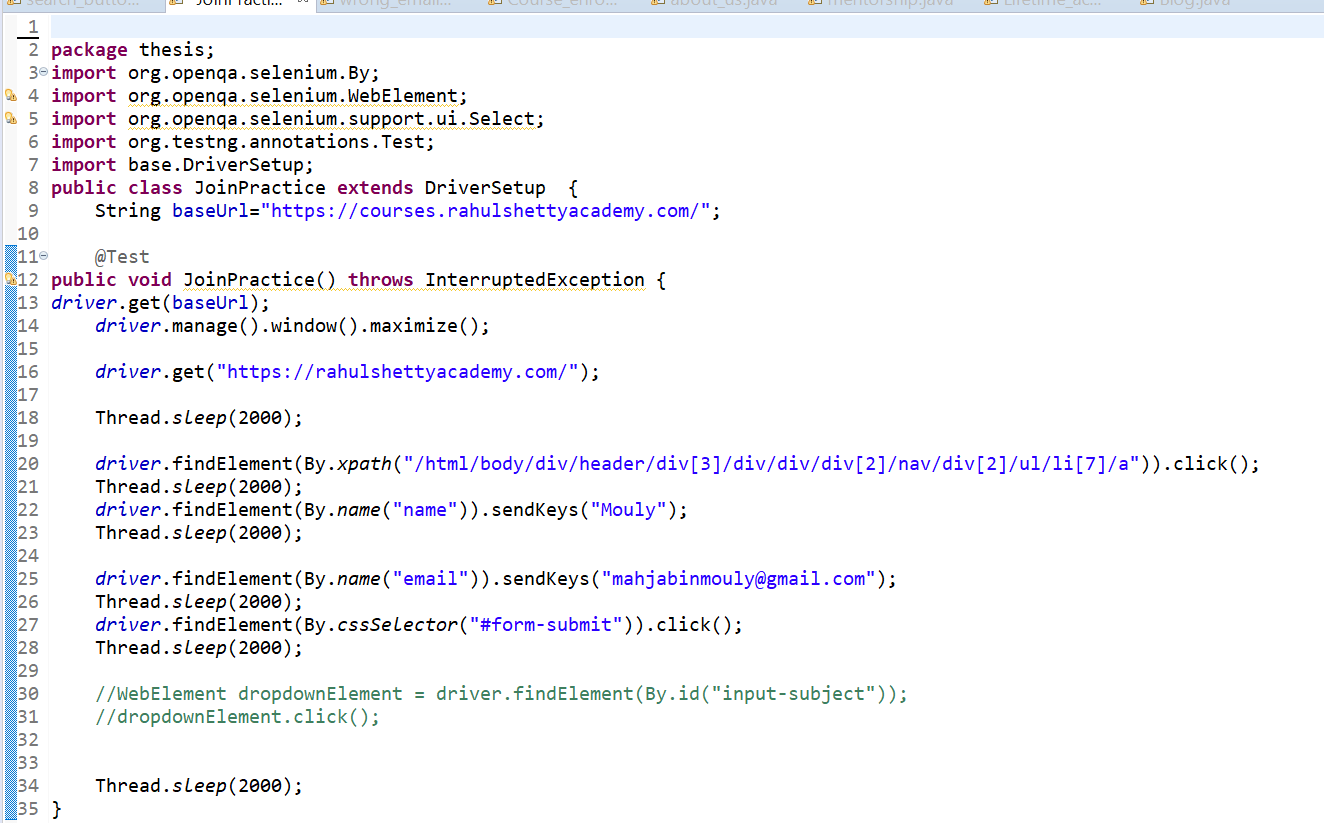


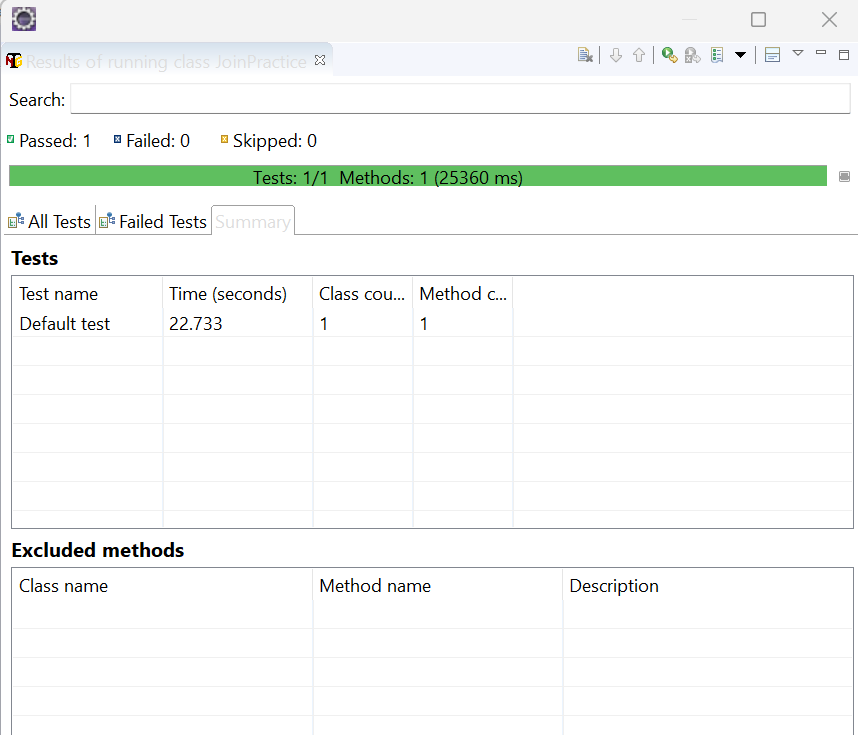




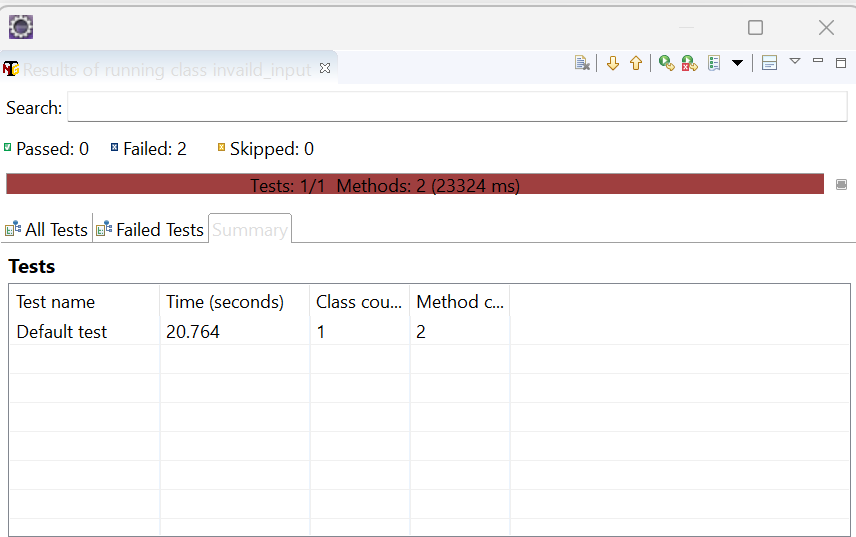








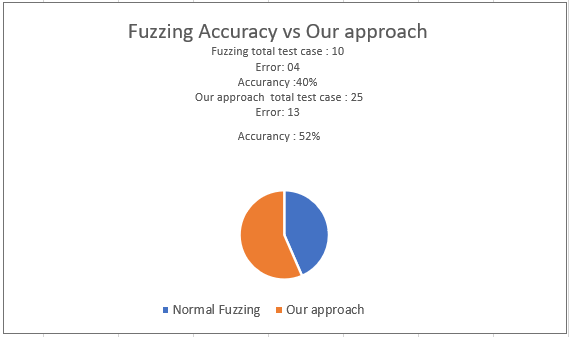




## Case summary:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case validity** | **Total case** | **Pass test** | **Fail test** | **Accuracy** |
| Fuzzing approach | 10 | 6 | 4 | 40% |
| Our Approach | 25 | 13 | 12 | 52% |

In this chapter we discus about the current Fuzzing test and our update approach at Fuzzing test and our suggestions for future work.



## Findings:

|  |  |  |
| --- | --- | --- |
| **Traditional Fuzzing** | **Our approach** | |
| Before making test cases, no extra preparation is needed. | Before making test cases, extra effort is needed to understanding and analyzing the software. | |
| Here we found limited number of test cases. Because it only works on based on inputs fields. | Here we found many test cases comparing to traditional fuzzing. Because in this portion we have considered both input fields and non-input fields. | |
| Limited coverage of bugs. | Wide coverage of bugs. | |
| This approach takes less time and less effort. we can generate and execute test cases very easily and less time. | | In this approach, there needs much time and effort before and after generating test cases. So this approach is lengthy. |
| For quickly delivering a software this testing is applicable but can’t find all kinds of vulnerability. | | This approach can wide range of vulnerabilities but take long time to perform the test. |

|  |
| --- |
| Chapter 5: Discussion |

In this chapter, we discus about the traditional Fuzzing test and enhanced fuzzing approach and our suggestions for future work.

## Fuzzing testing to update analysis

There on of the most popular testing which fuzz test or Fuzzing. Where Fuzz testing targets in system that should be represent range of vulnerability system characteristics. There exist some limitations with the current method of Fuzzing test to a system to inject that system error and bug in less time or in cup of time to detect vulnerability .But in our method take fuzzing more time to detect vulnerability but get more efficient result to more than normal fuzzing which every important to any kind of system or webapp. In current situation any software developer need kind of this type of update testing method which step by step detect any software vulnerability to a suitable software for their requirement .In normal approach to fuzzing have kind of limitation to detect vulnerability but in our approach where find fuzzing more efficient result from the normal fuzzing which give any kind of software too much bug free and also which will be secure also. In that scenario fuzzing method improved need to test report because where error get any software system each and every module that more convenient from normal fuzzing that why in our approach we get update version of fuzzing more that normal fuzzing.

1. Summary

In this situation basis on approach in fuzzing testing which more useable to any software testing and there also need some updating in our approach fuzzing testing more convenient from normal fuzzing that will be best in any kind of software testing.

|  |
| --- |
| Chapter 6: Conclusion |

This paper proposes a fuzzing testing in any kind of software to detect vulnerability.

In traditional fuzzing, there have some limitation to make a perfect software vulnerability. In our research paper, we have discussed the background of fuzzing testing because of the enhancement of fuzzing testing. There we used a term in the fuzzing process which is static analysis. In static analysis, there has some set of protocols which are software analysis, identify each module and etc. By following these sets of protocols we are able to find more bug from tradition fuzzing and there is another advantage, which is fuzz testing can be integrated with software functional testing like black box testing. By integrated these testing now we will be test both functional testing and vulnerability testing together. Although we have enhanced the fuzzing testing, then there can be more research to improve it in future.

|  |
| --- |
| References |

1. <https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app>
2. <https://www.ringcentral.com/gb/en/blog/definitions/web-application/>
3. V. J. M. Manes, H. Han, C. Han, S. K. Cha, M. Egele, E. J. Schwartz, and M. Woo, “The art, science, and engineering of fuzzing: A survey,” IEEE, 2019.
4. ”Richard Amankwah,” “ Patrick Kwaku Kudjo” ,“Samuel Yeboah Antwi,” “Evaluation of Software Vulnerability Detection Methods and Tools: A Review” ,Vol. 169 – No.8, July 2017.
5. Hossain Shahriar and Mohammad Zulkernine , “Automatic Testing of Program Security Vulnerabilities” , August 2009.
6. “Willy Jimenez “, “Amel Mammar”,” Ana Cavalli”, “Software Vulnerabilities, Prevention and Detection Methods: A Review”, July 2010.
7. M. Bajer, M. Szlagor and M. Wrzesniak “Embedded Software Testing in Research  
   Environment” IEEE, 2017.
8. H.-M. Qian and C. Zheng “A Embedded Software Testing Process Model” IEEE, 2020.
9. Lj. Lazic and D. V. Sevi “Applying simulation and design of experiments to the embedded software testing process” IEEE, 2017.
10. W. Liu, H. Niu, W. Luo, W. Deng, W. Feng “Research on Technology of Embedded System Security Protection Component”, IEEE, 2020.
11. J. Seo, Y. Ki, B. Choi, K. La “Which Spot Should I Test for Effective Embedded Software Testing?”, IEEE, 2009.
12. <https://www.esds.co.in/blog/manual-testing-process-lifecycle/>
13. <https://www.techtarget.com/searchsoftwarequality/definition/automated-software-testing>
14. https://www.guru99.com/white-box-testing.html
15. <https://www.guru99.com/black-box-testing.html>
16. <https://www.synopsys.com/glossary/what-is-fuzz-testing.html#:~:text=Definition,as%20crashes%20or%20information%20leakage>.
17. https://github.com/SupRiyA-Osman/Enhancement-of-fuzzing-testing-method-on-web-apps.