PES UNIVERSITY

AUGUST – DECEMBER 2022 SEMESTER 5

SOFTWARE ENGINEERING LAB TASKS

UNIT 4 LAB TASK

SUBMITTED BY

Mihir Jayaprakash	**PES2UG20CS196**
Mitul Joby	**PES2UG20CS199**
Navyae Goyal	**PES2UG20CS219**
Noel Jacob Abraham	**PES2UG20CS234**

PROBLEM STATEMENT 1: UNIT TESTING

A unit is the smallest block of code that functions individually. The first level of testing is Unit testing and this problem statement is geared towards the same.

- **Discuss with your teammates and demarcate units in your code base**
  Note: Discuss why the code snippet you have chosen can be classified as a unit
- **Develop test cases for both valid and invalid data**
- **Ideate how you could further modularize larger blocks of code into compact units with your teammates**

There are 9 units in the project which is a stock price predictor. The units are:

LSTM Model: LSTM uses the dataset to predict the future prices of the stock.
Backend Server: The LSTM model is stored here and past stock predictions are kept, and frontend and terminal application connect to it.
Login page: The login page lets the user and admin login into the website email id and password.
Admin page: After logging in, the admin gets directed to the admin page where he can see the dashboard.
Dashboard: The dashboard lets the admin see the number of users and the number of stocks predicted.
User History Page: After logging in, the user can use the user history page to see the user's previous stock predictions.
Evaluation of previous stock prices: Users can evaluate the past prediction of the stock prices.
Listing of company stocks: Shows the current set of stocks of companies which are available to predict

We have chosen the **Login Page** unit for testing, the reason for this is that login page is one of the most crucial parts of the project's frontend for recognizing the user in order to store their predictions.

TEST CASES FOR BOTH VALID AND INVALID DATA

TEST ID	ACTION	INPUT	ACTUAL OUTPUT	EXPECTED OUTPUT	RESULT
1	Enter valid username and short password	Username: admin Password: pass	The website says password length is too short.	The website should say the password length is too short.	PASSED
2	Enter valid username and invalid password	Username: admin Password: pass12345	The website says password is incorrect.	The website should say the password is incorrect.	PASSED
3	Enter valid username and valid password	Username: admin Password: pass@3214	The website accepts the user and redirected to homepage.	The website should accept the user and redirect to homepage.	PASSED
4	Enter invalid username and password	Username: bob Password: pass	The website says user does not exist.	The website should say user does not exist.	PASSED
5	Enter valid username	Username: admin	Says password is mandatory field.	Should say password is a mandatory field.	PASSED
6	Enter only password	Password: pass@3214	Says username is mandatory field.	Should say username is a mandatory field.	PASSED

Modularization is an approach which enables developers to write code in manageable, independent parts. Each unit is self-contained or encapsulated. In essence, this indicates that programmers are creating little building components, each of which has a single limited duty or functionality. These more compact building units are designed to function separately.

The following can be used to modularize:

- Making functional use of already written code.
- Cutting back on variables or entities
- Functionality should only carry out one particular duty.
- Using the fewest possible arguments for each function to reduce confusion.
- Creating the code in a way that makes it simpler to understand.

We can further modularize the LSTM model into further parts such as importing and reading the dataset, choosing and scaling the data, feature selection, building the LSTM, predicting the test data

PROBLEM STATEMENT 2: DYNAMIC TESTING

Dynamic testing involves execution of your code to analyze errors found during execution. Some common techniques are Boundary Value Analysis and Mutation Testing.

2.A: BOUNDARY VALUE ANALYSIS

When it comes to finding errors in your code base, they are often found at locations where a condition is being tested. Due to this, developers often use Boundary Value tests to reduce defect density.

- **How would you define a boundary test?**
- **Note: Simple relational conditions are a basic example**
- **Build your boundary test cases and execute them**

Boundary value analysis is a type of black box or specification-based testing technique in which tests are performed using the boundary values. It is a software testing technique in which tests are designed to include representatives of boundary values in a range. The idea comes from the boundary.

For e.g., any values between 1 and 100 are valid values, while anything below the minimum or above the maximum is invalid. The developer then engages in testing to ensure that the values 0 and 101 are invalid, while 2 and 99 are valid

In our project we used the boundary condition to create input for linear regression:

```
xtest= []
for i in range(60,80):
    xtest.append(test_input[i-60:i,0])
```

59 and 81 are invalid values in this case

SL.NO	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT
1	59	Not appended in the array	Not appended in the array
2	60	Appended in the array	Appended in the array
3	70	Appended in the array	Appended in the array
4	80	Appended in the array	Appended in the array
5	81	Not appended in the array	Not appended in the array

2.B: MUTATION TESTING

- **Using your isolated units from the first problem statement, ideate with your team mates on how to mutate the code**
- **Develop at least 3 mutants of the functioning code and test all 4 code bases using the test case from the first problem statement**

```
if len(password) < 8 or  ' ' in password:
        return jsonify({ "authenticated": False, "message": "Invalid Password" }), 400
```

INPUT	EXPECTED	ACTUAL	RESULT
Hello12345678	Accept Password	Accepted Password	PASS
Hello	Reject Password	Rejected Password	PASS
Hello world	Reject Password	Rejected Password	PASS

Mutant 1

```
if len(password) < 8 and  ' ' in password:
        return jsonify({ "authenticated": False, "message": "Invalid Password" }), 400
```

INPUT	EXPECTED	ACTUAL	RESULT
Hello12345678	Accept Password	Accepted Password	PASS
Hello	Reject Password	Accepted Password	FAIL
Hello world	Reject Password	Rejected Password	PASS

Mutant 2

```
if len(password) > 8 or  ‘ ’ in password:
        return jsonify({ "authenticated": False, "message": "Invalid Password" }), 400
```

INPUT	EXPECTED	ACTUAL	RESULT
Hello12345678	Accept Password	Rejected Password	FAIL
Hello	Reject Password	Accepted Password	FAIL
Hello world	Reject Password	Rejected Password	PASS

Mutant 3

```
if len(password) < 8 or  ‘ ’ not in password:
        return jsonify({ "authenticated": False, "message": "Invalid Password" }), 400
```

INPUT	EXPECTED	ACTUAL	RESULT
Hello12345678	Accept Password	Rejected Password	FAIL
Hello	Reject Password	Rejected Password	PASS
Hello world	Reject Password	Accepted Password	FAIL

All mutants got killed.

Mutant score = (killed mutants/total mutants) *100

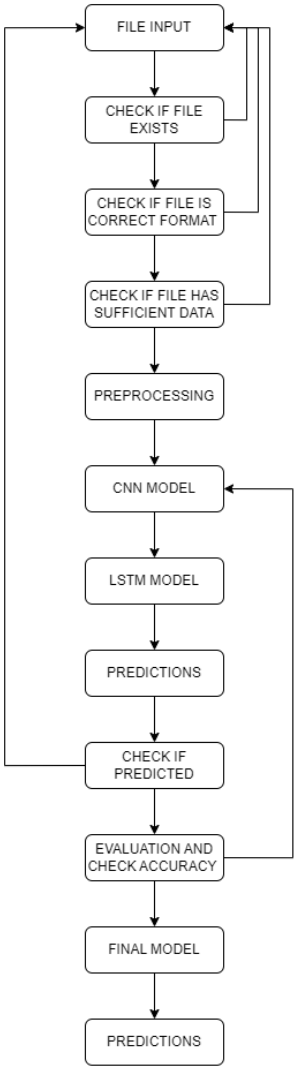
= (3/3) * 100 = 100%

PROBLEM STATEMENT 3: STATIC TESTING

Static testing involves validating your code without any execution. Under this problem statement, you will be expected to analyze and calculate the cyclomatic complexity of your code.

- **Using the unit you selected in the first problem statement as an example, develop the control flow graph of your problem statement.**
- **Using the Control flow graph, calculate the cyclomatic complexity of your code.**
- **Using the cyclomatic complexity as an indicator, Ideate and code your unit again to reduce complexity**

CONTROL FLOW DIAGRAM



CYCLOMATIC COMPLEXITY

Cyclomatic Complexity = E-N +2P
E = 16
N = 12
P = 1

Cyclomatic Complexity = 16 – 12 + 2 = 6

Code is written with the intention of decreasing time and space complexity and with an aim on performance efficiency and stability. To decrease the complexity, we can reduce if/else statements. Most often, we don't need an else statement, as we can just use return inside the 'if' statement.

PROBLEM STATEMENT 4: ACCEPTANCE TESTING

Assume your neighboring team is the client for your code. Give them an idea of what your product is and the software requirements for the product.

- **Exchange your code base and test each other's projects to see if it meets user requirements**
- **If you identify a bug in the project you are testing, inform the opposing team of the bug**
- **As a team, based in clients experience, ideate modifications to the existing project that could improve client experience**.

The project being tested is a website to predict stocks. The user was greeted by the land page was prompted to login to continue. If they user did not have an account, they had the ability to create a new one. On successful login, the user lands on the home page. He has the ability to view a list of available stocks and get their corresponding data. They can also view visualizations of the data. The user has the ability to predict stocks and view the graph for it. The user can also view past predictions and evaluate against the actual value of the time and see how accurate it was. The user could reload the website and the session would be remembered successfully. The user could logout successfully and would be prompted to login again. There was also a terminal application which could get stocks and predictions which worked as expected.

The website had some compatibly issues for some browser sizes. The predictions took some time for some stock indexes, and would give a better user end experience if the predictions were made faster.

PROBLEM STATEMENT 5: MAINTENANCE ACTIVITIES

Once a product is completed, it is handed off to a service-based company to ensure all maintenance activities are performed without the added expenditure of skilled developers. However, a few tasks are performed by the maintenance team to gauge the product better. In this problem statement, you will be asked to experiment with your code.

- **Exchange code bases with your neighboring teams and reverse engineer a block of code in order to understand it's functionality**
- **After understanding the code block, Re-Engineer the code**
    - **Ideate how to refactor the code and the portion of the code base you would have to change**
    - **Discuss how the new changes would impact the time and space complexity of the project during execution**
- **After Reverse Engineering and Re-Engineering, the code, perform acceptance testing between the teams**

Reverse Engineering is the process or method through which one attempts to understand through deductive reasoning how a previously made device, process, system, or piece of software accomplishes a task with very little insight into exactly how it does so. It allows us to identify how the developer design a particular part of code so that we can recreate and create a replacement part of the product the code the engineers copy or mimic a design without the original blueprint

The project we are referring to for reverse engineering is a URL phishing detector.

- The URL phishing detector software that has been built works well but is not perfect since it is built on a specific dataset and is trained according to the attributes of the data set.
- The dataset has attributes which can be used to detect if a URL is unsafe or not but is not sufficient since there are more complex features which can be used to find out if a URL is a phishing URL like pattern matching or typo-squatting. Time and space complexity of the model isn't affected much.
- In order to make it better apart from the supervised model, we can add more functions to identify phishing scams on top of the models built. We

can define our own functions which will return a probability of safe/unsafe URL and this combined with the probability which we receive from the model prediction can be used to give more accurate results.

- When we do data visualisation, we see that the presence of HTTPS in the URL is the primary factor by which URLs are classified as phishing or non-phishing by a big margin. So, on adding more features and/or functions to check for complex phishing attacks, dependency on one particular feature (HTTPS) is reduced and the classification criteria for classification is diversified. This should improve the prediction accuracy.

Acceptance Testing for the Reverse Engineered code

Model building used (Gradient boosting)	Function for typo-squatting used	Function for pattern matching used	User input	Expected Output	Actual Output	Accuracy
Yes	No	No	www.go.gl.com	Phishing (Unsafe)	Safe	60%
Yes	Yes	No	www.go.gl.com	Phishing (Unsafe)	Phishing (Unsafe)	75%
Yes	No	Yes	www.go.gl.com	Phishing (Unsafe)	Phishing (Unsafe)	80%
Yes	Yes	Yes	www.go.gl.com	Phishing (Unsafe)	Phishing (Unsafe)	90%

The model is much more consistent and accurate when Functions for Typo-squatting and Pattern matching are included in the model. The software can be improved a lot by implementing these features. In cases like phishing attacks, accuracy of the detector is of the highest importance and hence these changes will be justified.