Problem Statement – 1a: For non Scrum - Agile project

Programming standards give your code a more uniform approach and enhance its testability.

Keeping your project in mind, attempt the below questions:

1. Do you think Defensive Programming is applicable to your project?

Defensive programming is the creation of code for computer software designed to avoid problematic issues before they arise and make the product more stable.

The basic idea behind this approach is to create a program that is able to run properly even through unforeseen processes or when unexpected entries are made by users.

In our project, we plan to use assertive programming to verify assumptions as and when we code. Object-oriented programming languages enable developers to encapsulate resources and classes, or they can use code wrappers to check for details like resource state. We plan to use PHP and Java, which are object-oriented programming language. We learnt that another defensive programming practice is to accept responsibility for resources such as files, memory and devices. The person who allocates a resource should be the one to deallocate it.

• Follow up: Is Redundancy not applicable in the case where a single system is executing the program?

Many systems can provide internal hardware redundancy of components that are extremely prone to failure. The most common example of this in-built redundancy is systems or devices that have redundant onboard power in the event of a power supply failure.

2. As a developer, should you rely entirely on the user providing correct input?
• In your project, include statements to ensure that the program only continues with data if it is of the expected type.

No, we should not rely entirely on the user providing correct input. We plan to implement data type checking using try and except blocks. The developer needs to assume that the user has no idea about the data types of the variables and the user does not know the purpose of the variables.

3. While working on your project, the go to approach to debug is using print statements. Include these statements to highlight the location and values at the current point in the project.

```
<!DOCTYPE html>
<html>
<body>

<?php
// Prints the day
echo date("l") . "<br>";

// Prints the day, date, month, year, time, AM or PM
echo date("l jS \of F Y h:i:s A") . "<br>";

// Prints October 3, 1975 was on a Friday
echo "Oct 3,1975 was on a ".date("l", mktime(0,0,0,10,3,1975)) . "<br>";

// Use a constant in the format parameter
echo date(DATE_RFC822) . "<br>";

// prints something like: 1975-10-03T00:00:00+00:00
echo date(DATE_ATOM,mktime(0,0,0,10,3,1975));
?>

</body>
</html>
```

4. Build two valid and invalid test cases for any two functions in your project.

Valid test cases ensure that users can perform appropriate actions when using valid data.

Valid test cases in our project are implemented using NOT NULL constraints in SQL.

Invalid test cases are performed to try to “break” the software by performing invalid (or unacceptable) actions, or by using invalid data.

Invalid test cases in our project are-

- Booking a seat when there are no seats available
- Booking a seat for a journey when there are no flights from the desired source to the desired destination

As the name itself implies, boundary indicates limits to something. Hence this involves designing test scenarios that only focus on the boundary values and validate how the application behaves. Therefore if the inputs are supplied within the boundary values then it is considered to be positive testing and inputs beyond the boundary values is considered to be a part of negative testing.

Problem Statement – 2

Construction quality is of utmost importance when developing projects. With respect to your project, you are expected to document and attempt the below mentioned quality activities.

1. Evaluate your project according to the measures and metrics in "Proceeding as planned" and "Technical quality". Your team is also expected to evaluate and document the quality of the project according to FLURPS+.

Functionality:

Functional Requirements

- A user/agent can register with the application providing necessary details
- The user/agent logs into the application, with the valid username and password
- The user books airline tickets.
- The user can reserve seats for an airplane flight.
- The agent can view the number of people in a particular flight.
- To be able to run on desired OS
- To display the data to the user
- To be able to store a DB

Non-Functional Requirements:

- Interactive UI
- Performing advanced operations in regards to said data
- Future expansions

Localizability:

Currently the application is only going to support localizability in the English language. Further expansions may occur depending on final scope.

Usability:

Usability describes the simplicity and ease of use of the application. Most of our target audience will use general transaction functions which are shared by every other UPI based app. Besides these, we have a much more encapsulated view for the data collected. This leaves very little for the user to do themselves.

Reliability:

It means the extent to which the program performs with required precision. The website developed should be extremely reliable and secure so that info about any questions etc is not leaked. The system shall not be down more than 2 times in a year.
The reliability of the overall project depends on the reliability of the separate components. The main pillar of reliability of the system is the backup of the database which is continuously maintained and updated to reflect the most recent changes. Also, the system will be functional under a container. Thus the overall stability of the system depends on the stability of the container and its underlying OS.

Performance:

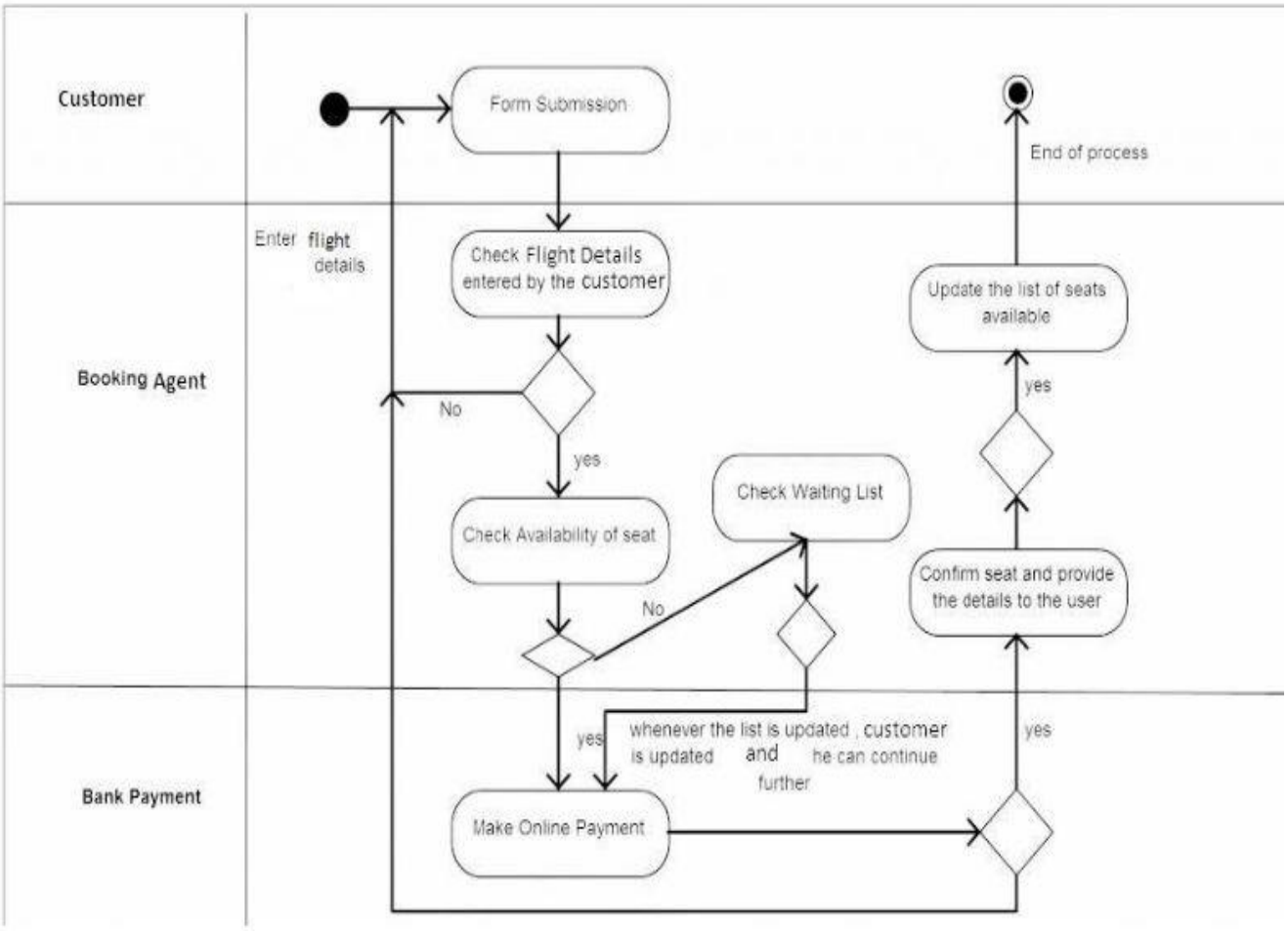
Untested on multiple systems as of yet, but seems smooth as compared to other similar services.

- Response Time: The response of all operations is good.
- Error Handling: Response to user errors and undesired situation has been taken care of to ensure that the system operates without halting.
- Safety and Robustness: The system is able to avoid or tackle disastrous action. In other words, it is foul proof.
- Portable: The software is not architecture specific. It is easily transferable to other platforms if needed.
- User Friendliness: The system is easy to learn and understand. A native user can also use the system effectively, without any difficulties
- User Satisfaction: The system is such that it stands upto the user expectations

Supportability:

The code and supporting modules of the system will be well documented and easy to understand. Online user documentation and system requirements will be provided.

2.Pick a function from your project with significant computations, use a pen and paper to statically analyze the flow of execution and the value of the variables for a specified input.



****