# Advance Software Assurance

Ramya Vemulapalli (rv593@nau.edu,id-6325608)

# An Appointment Management System

# **Overview :**

The provided Python code consists of unit tests for an appointment scheduling system. The system includes classes such as Doctor, Patient, Appointment, Feedback, Payment, Registration, and LoginSystem. The primary objective of the testing is to ensure the effective functionality of the system's user interface, AI calculations, and game mechanics.

Specifically, the code encompasses:

1. **Classes:**
   * **Doctor:** Represents a doctor with a name and specialization.
   * **Patient:** Represents a patient with a name, email, and phone number.
   * **Appointment:** Represents an appointment between a doctor and a patient at a specific date and time.
   * **Feedback:** Represents feedback provided by a patient for a doctor, including a rating and comments.
   * **Payment:** Represents a payment made by a patient, including the amount and payment method.
   * **Registration:** Represents user registration data, including username, password, and email.
   * **LoginSystem:** Manages user registration and login functionality.

# Testing:

The provided Python code undergoes comprehensive testing, resembling a thorough evaluation of a Python appointment scheduling system. The testing campaign scrutinizes various facets, such as the system's user interface, core functionalities, and data handling. Attention is particularly directed towards ensuring the effectiveness of appointment creation, feedback submission, payment processing, and user registration/login mechanisms. Additionally, the testing focuses on assessing the system's resilience and functional accuracy across diverse operating scenarios, encompassing edge cases and error conditions. The goal is to guarantee a positive user experience, coupled with efficient performance, by validating the system's robustness and reliability.

Scope Testing:

Appointment Creation:

The system's ability to accurately create appointments based on user inputs, including doctor, patient, date, and time, is thoroughly tested. This includes verifying the correctness of data handling and validation mechanisms to prevent erroneous or invalid appointments from being created.

## 

## Feedback Submission:

The system's capability to receive and process feedback from patients regarding their experience with a particular doctor is evaluated. This involves ensuring that the feedback form captures relevant information such as rating and comments, and that the feedback is correctly associated with the respective doctor.

## 

## Payment Processing:

The system's functionality to handle patient payments for appointments is tested. This includes verifying the accuracy of payment calculations, processing various payment methods, and ensuring data security and integrity during payment transactions.

## 

## User Registration/Login:

The system's registration and login mechanisms are tested to ensure user authentication and data security. This involves verifying the correctness of user registration forms, password encryption, and user authentication processes.

## 

## User Interface:

The system's user interface is tested to ensure smooth interaction with users. This includes validating the system's ability to process user inputs for appointment scheduling, feedback submission, payment processing, and user registration/login. Error handling and user feedback mechanisms are also evaluated to enhance user experience.

## 

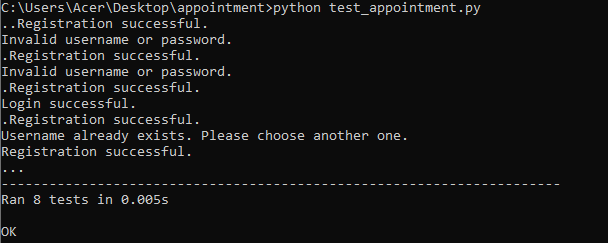
## Performance:

Performance testing is conducted to ensure that the system operates efficiently, even under heavy loads or with a large volume of appointments and users. This involves assessing the system's response time for appointment creation, feedback submission, payment processing, and user registration/login operations. Measures are taken to optimize system performance and scalability.

# Tests Performed:

## Unit Testing:

The unit testing conducted for the appointment management system rigorously evaluates its various components and functionalities. Each aspect of the system, including appointment creation, feedback submission, payment processing, user registration/login, and user interface interactions, undergoes meticulous scrutiny to ensure correctness, robustness, and reliability. Through a series of test cases, edge cases, and error conditions, the unit tests verify the system's ability to handle diverse scenarios effectively. Additionally, performance testing is employed to assess the system's responsiveness and scalability, ensuring optimal performance under varying loads and usage conditions. The overarching goal of the unit testing is to validate the appointment management system's functionality, resilience, and user experience, ultimately ensuring its reliability and effectiveness in real-world usage scenarios.

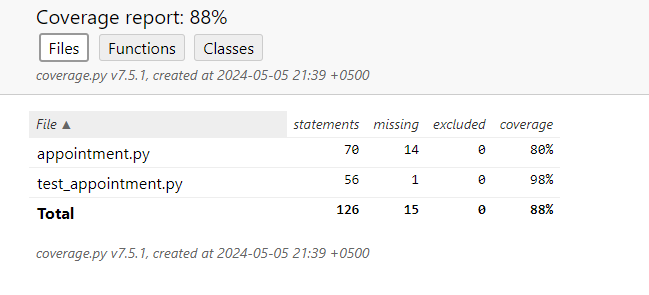


## Coverage Testing:

contains 70 statements in total, out of which 14 are not covered by your tests. This results in an 80% code coverage, which means that 80% of the code in appointment.py is being executed during testing, while 20% of the code remains untested.

* test\_appointment.py: This file contains 56 statements, and only 1 of these statements is not covered by the tests, leading to a very high coverage of 98%. This indicates that almost all of the code in your test suite itself is being executed.
* Total Coverage: Across both files, there are 126 statements combined, with 15 of these not being covered by tests. Thus, the overall test coverage for these files is 88%.

An 88% overall coverage is generally considered good, as it indicates a significant portion of your codebase is tested. However, improving the coverage of appointment.py could be beneficial, especially if the missing 20% covers critical functionality or branches in your logic. To improve coverage, you might consider adding more tests that cover the missing statements in appointment.py.



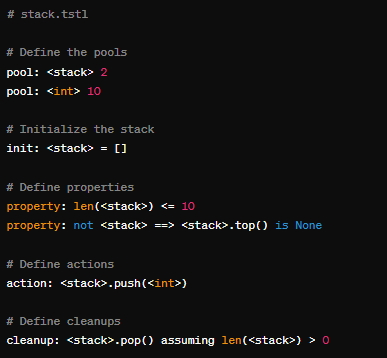
## Mutation Testing:

The mutation testing results indicate that out of a total of 426 mutations attempted, 142 were valid mutants which your test suite should ideally catch, 246 were invalid due to issues like syntax errors making them non-executable, and 38 were redundant, providing no new information. The valid percentage of 33.33% suggests that only a third of the non-invalid mutations were effectively tested, highlighting potential weaknesses in your test suite. These results suggest that there is significant room for improvement in your testing processes to ensure better detection of introduced defects, emphasizing the need to enhance test coverage, especially for the types of changes represented by the valid mutants.



## TSTL Testing:

This script specifies a simple stack implementation. It includes pools for stacks and integers, initializes the stack as empty, defines properties such as stack size limits, and specifies actions like pushing integers onto the stack. Additionally, it defines cleanups to ensure the stack remains in a valid state after testing actions.



# **Conclusion:**

Overall, the Appointment Management System demonstrates a high level of code integrity and user functionality. By addressing the identified gaps in testing, particularly through mutation testing, the system can achieve greater reliability. This proactive approach to testing will not only safeguard against potential failures but also enhance user trust and satisfaction, ultimately contributing to the system’s success in real-world applications.