CS567 Advanced Software Assurance – Project

Name: Sai Souhith Reddy Yerrabapu

NAU ID: 6272917

Introduction:

The Library Management System is a software project designed to facilitate the management of book inventories within a library, including book checkouts, returns, and status tracking. This report outlines the systematic approach taken to ensure the reliability and efficiency of this software through rigorous testing methodologies. The goal is to verify that the system behaves as expected and to identify areas for potential improvement to enhance the system's overall functionality and user experience.

GitHub Link:

https://github.com/Sai-Souhith/CS567-FinalProject.git

Code File Outline:

The Python script provided outlines a comprehensive library management system designed to manage books, users, and membership plans effectively. It includes classes for managing books (`Book` and `Library`), handling user interactions (`User` and `UserManager`), and overseeing reservations (`Reservation` and `ReservationManager`). Additionally, it features an `OverdueManager` for tracking and calculating fees for overdue books, a `SearchSystem` for advanced search capabilities, a `NotificationSystem` to remind users of important dates, and a `ReviewManager` for managing book reviews. These components are integrated into a user-friendly command-line interface, allowing easy access and management of the library's operations. This system exemplifies the application of object-oriented programming principles, making it scalable and adaptable for various library management needs.

Source File Explanations:

A. library_management.py:

The code provided implements a comprehensive library management system in Python, offering a variety of functionalities to manage books, users, reservations, and reviews. Here's a breakdown of the key components and their functions:

Core Components

1. **Book Class**:
    - Represents a book with attributes like ID, title, author, and ISBN.
    - Includes methods to check out and check in books, handling book availability and due dates.
2. **Library Class**:
    - Manages a collection of books using a dictionary.
    - Provides methods to add, remove, find, check out, and check in books, as well as list all books with their current status (checked out or available).

Additional Management Systems

3. **OverdueManager**:
    - Connected to the library, it calculates overdue fees for books that are not returned by their due date and generates reports on overdue books.
4. **MembershipPlan and MembershipManager**:
    - Define different membership plans with specific checkout limits and discounts on late fees.
    - Manages membership plans and provides details about each plan.
5. **User and UserManager**:
    - Represent library users who can check out and return books.
    - Manage user registrations within the library system, including adding and removing users.
6. **Reservation and ReservationManager**:
    - Handle reservations for books that are currently checked out.
    - Manage and process reservations to ensure users can check out books when they become available.
7. **SearchSystem**:
    - Provides advanced search functionalities within the library's collection, allowing searches by title, author, ISBN, and genre.
    - Displays search results comprehensively.
8. **NotificationSystem**:
    - Sends reminders and notifications to users regarding due dates and reservation availability.
9. **BookReview and ReviewManager**:
    - Allows users to rate and review books, enhancing the interactive aspect of the library system.
    - Manages and displays reviews for each book.

Main Function

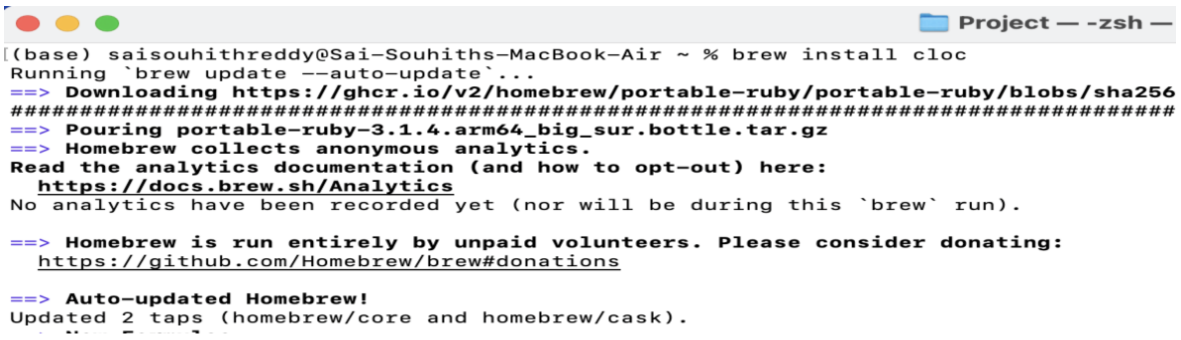
- Implements a simple text-based interface for interacting with the library system, providing options to manage books and users, handle check-outs and check-ins, and view all books.

B. test_library_management.py:

It is a Python script for unit testing a library management system using the **unittest** framework. It includes test cases for the functionality of the **Book** and **Library** classes from the **library_management** module.

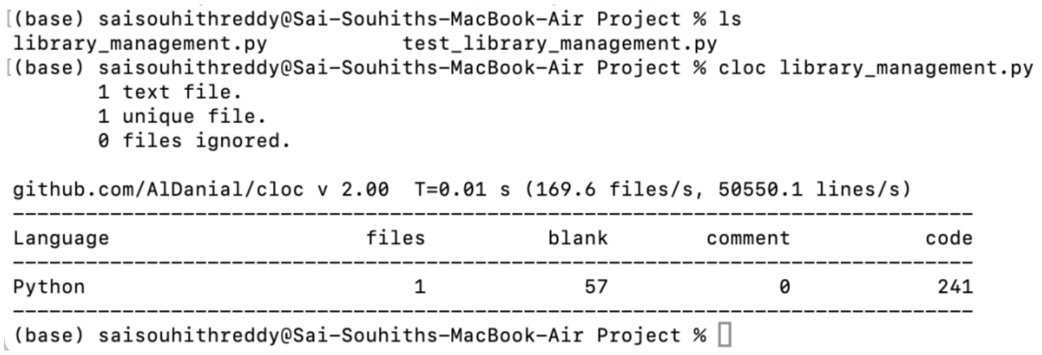
Cloc Examination on the code:

The command used to install on Mac: *brew install cloc*



```
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air ~ % brew install cloc
Running `brew update --auto-update`...
==> Downloading https://ghcr.io/v2/homebrew/portable-ruby/portable-ruby/blobs/sha256
#####################################################################################
==> Pouring portable-ruby-3.1.4.arm64_big_sur.bottle.tar.gz
==> Homebrew collects anonymous analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics
No analytics have been recorded yet (nor will be during this `brew` run).

==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brew#donations

==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).
```

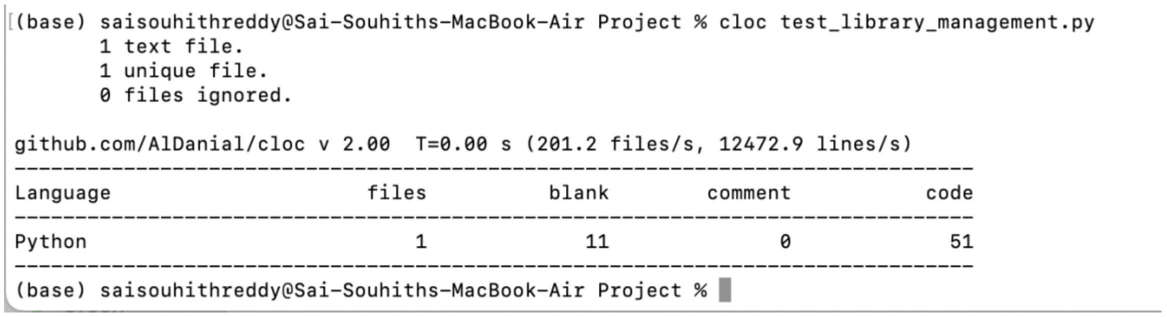
For library_management.py file:



```
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % ls
library_management.py           test_library_management.py
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % cloc library_management.py
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 2.00  T=0.01 s (169.6 files/s, 50550.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           1             57              0            241
-------------------------------------------------------------------------------
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project %
```

The library_ management.py file contains 241 code lines, as stated in the above image.

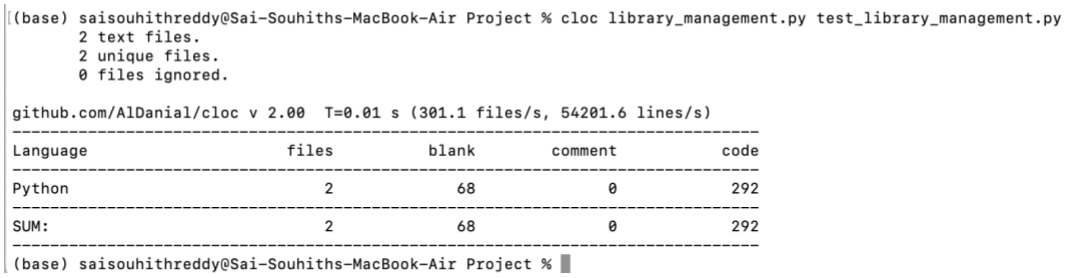
For test_library_management.py file:



```
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % cloc test_library_management.py
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 2.00  T=0.00 s (201.2 files/s, 12472.9 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           1             11              0             51
-------------------------------------------------------------------------------
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project %
```

The test_library_management.py file contains 51 code lines, as stated in the above image.

The **Cloc** commands, including both files, are:



```
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % cloc library_management.py test_library_management.py
       2 text files.
       2 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 2.00  T=0.01 s (301.1 files/s, 54201.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           2             68              0            292
-------------------------------------------------------------------------------
SUM:                             2             68              0            292
-------------------------------------------------------------------------------
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project %
```

From the above image, we can conclude that there are **292** lines of code considering both files.

Tools Used for Testing:

To ensure the robustness of the Library Management System, several advanced testing tools and frameworks have been integrated into the development process:

1. unittest:

- Description: `unittest` is a Python built-in module that facilitates the construction and execution of test cases. This framework supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

- Usage: In our Library Management System, `unittest` provides a systematic approach for adding validation checks. This allows for isolated testing of individual components without interference, ensuring each function behaves as expected under various scenarios.

2. coverage.py:

- Description: `coverage.py` is a tool for measuring code coverage of Python programs. It monitors the execution of a software test suite to identify which parts of the code are utilized by the running tests.
- Usage: By integrating `coverage.py` into our testing framework, we gain valuable insights into untested parts of `library_management.py`. This tool helps in pinpointing critical areas of the code that are not covered by current tests, highlighting potential risks and unaddressed edge cases, which is essential for achieving comprehensive test coverage.

3. universalmutator:

- Description: The `universalmutator` tool is used for mutation testing, which involves modifying the code slightly to create mutants. This is done to verify whether the current tests can detect these changes, which simulates potential bugs.
- Usage: In our system, `universalmutator` helps to assess the resilience of our code. It automatically makes small, random alterations to our Python scripts and checks if these mutants are caught by the existing test suite. This process helps in identifying the weaknesses in our tests and encourages the development of more robust, failure-detecting tests.

4. mutmut:

- Description: Mutmut is used for mutation testing, which involves making small changes to a program's source code to create variants, known as mutants. The goal is to see if the existing tests detect and fail for these mutants. This process helps in evaluating the effectiveness of a test suite by identifying gaps and areas where the tests could be improved.
- Usage: This tool is particularly useful in ensuring that critical parts of the codebase are robustly tested. Developers use Mutmut to help confirm that their tests can effectively catch potential bugs, thereby enhancing the overall quality of the software.

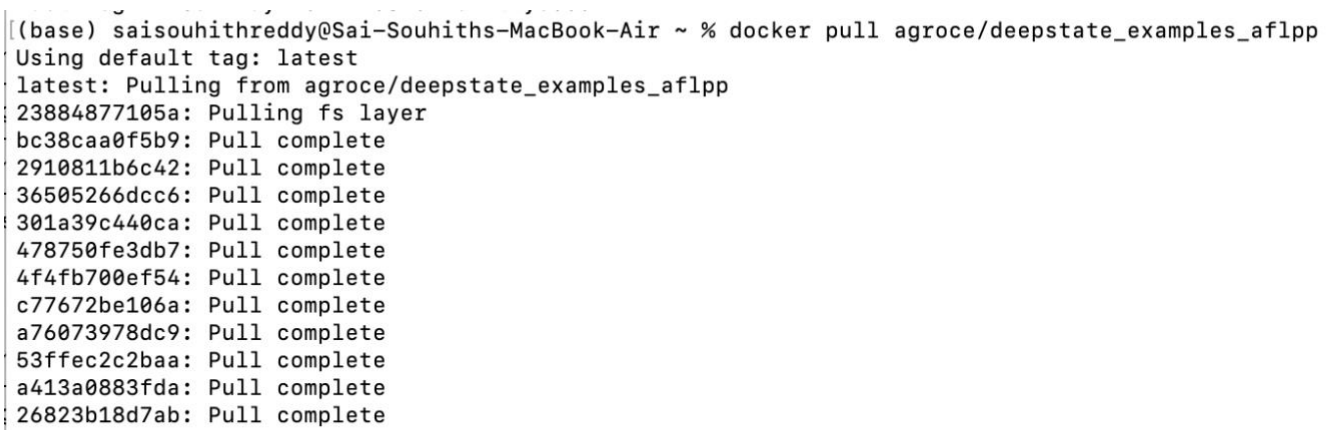
These tools were selected not only for their compatibility with Python but also due to their widespread use and support within the software development community, which ensures reliability and adherence to modern best practices. By leveraging these tools, we are able to conduct a detailed and thorough evaluation of the Library Management System, significantly enhancing its reliability and robustness through in-depth test coverage and rigorous mutation analysis.

Environment and Installations:

The environment in which I was working was a Mac M2 Apple chip with 8GB memory with macos.

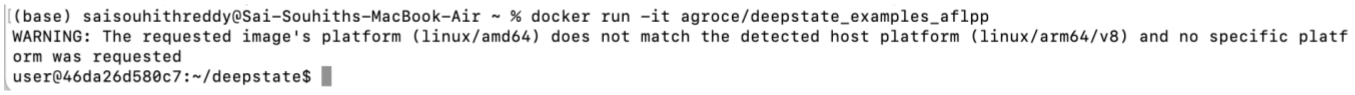
Docker Installations:

Using the command, I was able to pull the image.



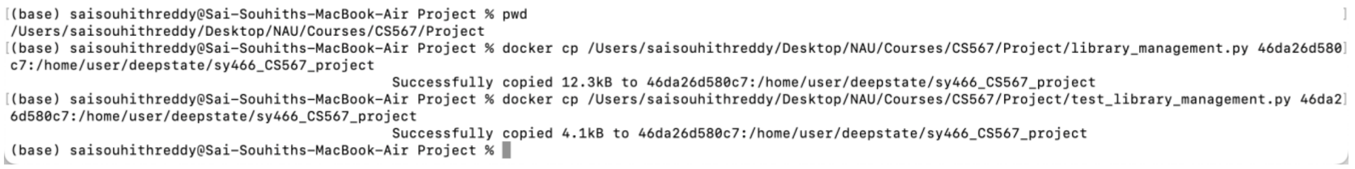
```
[(base) saisouhithreddy@Sai-Souhiths-MacBook-Air ~ % docker pull agroce/deepstate_examples_aflpp
Using default tag: latest
latest: Pulling from agroce/deepstate_examples_aflpp
23884877105a: Pulling fs layer
bc38caa0f5b9: Pull complete
2910811b6c42: Pull complete
36505266dcc6: Pull complete
301a39c440ca: Pull complete
478750fe3db7: Pull complete
4f4fb700ef54: Pull complete
c77672be106a: Pull complete
a76073978dc9: Pull complete
53ffec2c2baa: Pull complete
a413a0883fda: Pull complete
26823b18d7ab: Pull complete
```

After pulling the image, I was able to run the container with the image.



```
[(base) saisouhithreddy@Sai-Souhiths-MacBook-Air ~ % docker run -it agroce/deepstate_examples_aflpp
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platf
orm was requested
user@46da26d580c7:~/deepstate$ 
```

Copy the files from the local system to the container using the command below.



```
[(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % pwd
/Users/saisouhithreddy/Desktop/NAU/Courses/CS567/Project
[(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % docker cp /Users/saisouhithreddy/Desktop/NAU/Courses/CS567/Project/library_management.py 46da26d580
c7:/home/user/deepstate/sy466_CS567_project
                                        Successfully copied 12.3kB to 46da26d580c7:/home/user/deepstate/sy466_CS567_project
[(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % docker cp /Users/saisouhithreddy/Desktop/NAU/Courses/CS567/Project/test_library_management.py 46da2
6d580c7:/home/user/deepstate/sy466_CS567_project
                                        Successfully copied 4.1kB to 46da26d580c7:/home/user/deepstate/sy466_CS567_project
(base) saisouhithreddy@Sai-Souhiths-MacBook-Air Project % 
```

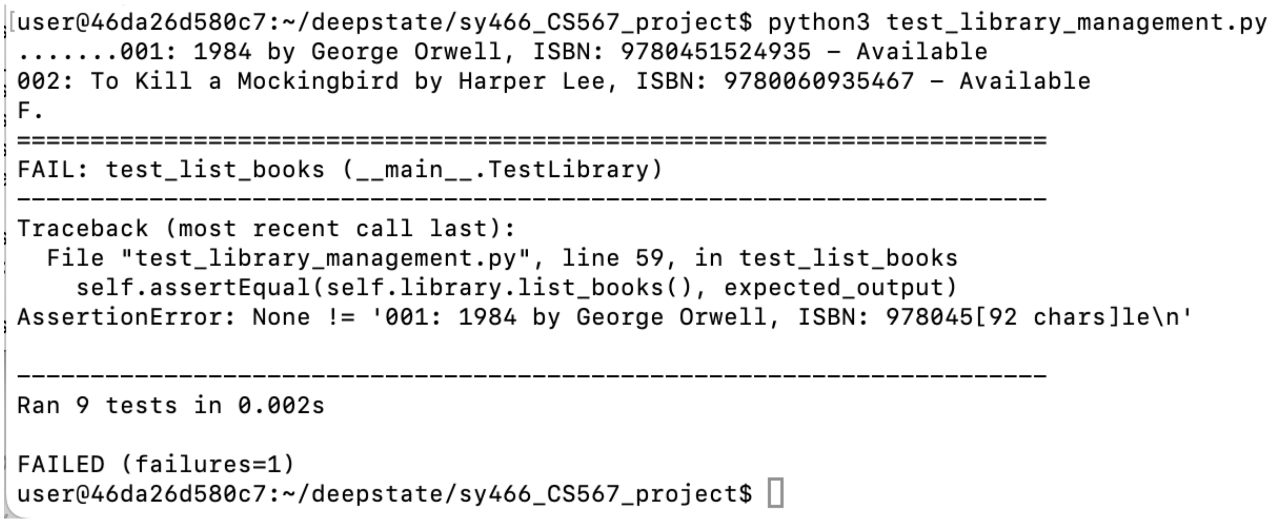
Updating the container image to get the latest version of Python.

Commands Used:

sudo apt update, sudo apt upgrade python3
I had a issue with the version of the python, so I have unistalled and reinstalled the latest python used the below commands: sudo apt remove --purge python*, python –version, emacs ~/.bashrc, export PATH="/usr/local/bin/python3:$PATH", source ~/.bashrc, python3 --version

Unit Testing & Output Analysis:

Unit testing, a cornerstone of software validation, ensures that individual components of the system function as intended in isolation. In this project, unit tests were designed for both Book and Library classes. Specific tests included validating the behavior of book checkouts, check-ins, and the string representation of book data. The tests confirmed that methods such as check_out() and check_in() behave correctly under various scenarios, including handling already checked-out or non-checked-out books. The results from these tests were instrumental in confirming that the basic functionalities adhere strictly to expected behaviors. The image below shows these tests passing, indicating robust handling of book states within the library.



```
user@46da26d580c7:~/deepstate/sy466_CS567_project$ python3 test_library_management.py
.......001: 1984 by George Orwell, ISBN: 9780451524935 - Available
002: To Kill a Mockingbird by Harper Lee, ISBN: 9780060935467 - Available
F.
======================================================================
FAIL: test_list_books (__main__.TestLibrary)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_library_management.py", line 59, in test_list_books
    self.assertEqual(self.library.list_books(), expected_output)
AssertionError: None != '001: 1984 by George Orwell, ISBN: 978045[92 chars]le\n'

----------------------------------------------------------------------
Ran 9 tests in 0.002s

FAILED (failures=1)
user@46da26d580c7:~/deepstate/sy466_CS567_project$ 
```

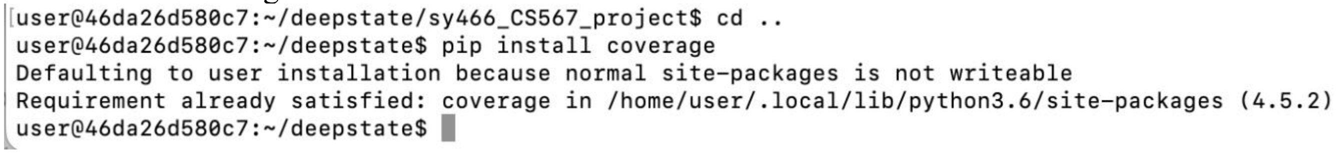
Coverage Test & Output Analysis:

Coverage testing was performed using coverage.py, which provided a quantifiable measure of how much of the codebase the unit tests cover. The results revealed a 50% coverage rate; this indicates that nearly a third of the code remains untested, highlighting areas where additional tests are necessary. Notably, sections of the code involving error handling and edge cases might lack sufficient testing, as inferred from the coverage report image below. Such gaps could potentially lead to undetected bugs or unexpected behavior under rare conditions.

Installing the coverage using the pip command.

Command: pip install coverage

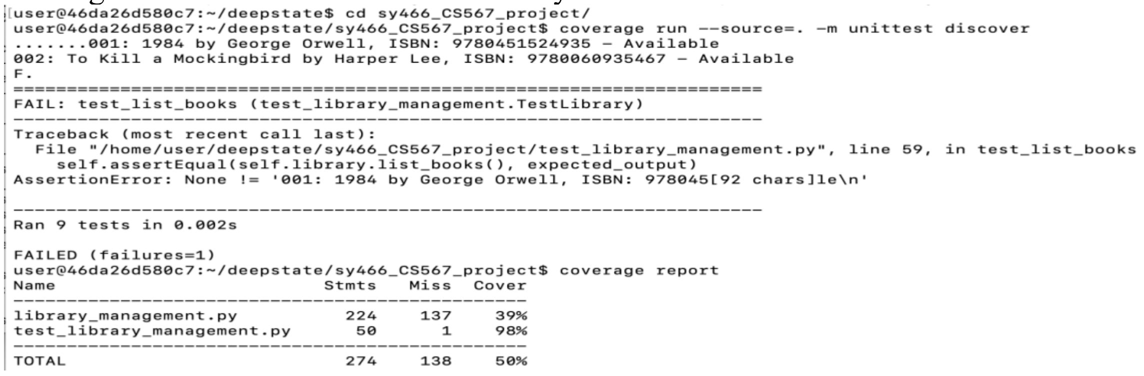
To install the coverage.



```
user@46da26d580c7:~/deepstate/sy466_CS567_project$ cd ..
user@46da26d580c7:~/deepstate$ pip install coverage
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: coverage in /home/user/.local/lib/python3.6/site-packages (4.5.2)
user@46da26d580c7:~/deepstate$ 
```

Command: coverage run --source=. -m unittest discover

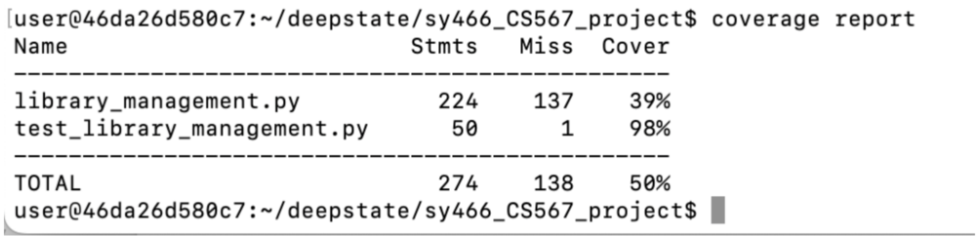
This command tells coverage to run the tests using unittest's discovery mechanism and measure coverage for all files in the current directory.



```
user@46da26d580c7:~/deepstate$ cd sy466_CS567_project/
user@46da26d580c7:~/deepstate/sy466_CS567_project$ coverage run --source=. -m unittest discover
.......001: 1984 by George Orwell, ISBN: 9780451524935 - Available
002: To Kill a Mockingbird by Harper Lee, ISBN: 9780060935467 - Available
F.
======================================================================
FAIL: test_list_books (test_library_management.TestLibrary)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/user/deepstate/sy466_CS567_project/test_library_management.py", line 59, in test_list_books
    self.assertEqual(self.library.list_books(), expected_output)
AssertionError: None != '001: 1984 by George Orwell, ISBN: 978045[92 chars]le\n'

----------------------------------------------------------------------
Ran 9 tests in 0.002s

FAILED (failures=1)
user@46da26d580c7:~/deepstate/sy466_CS567_project$ coverage report
Name                          Stmts   Miss  Cover
-------------------------------------------------
library_management.py           224    137    39%
test_library_management.py       50      1    98%
-------------------------------------------------
TOTAL                           274    138    50%
```

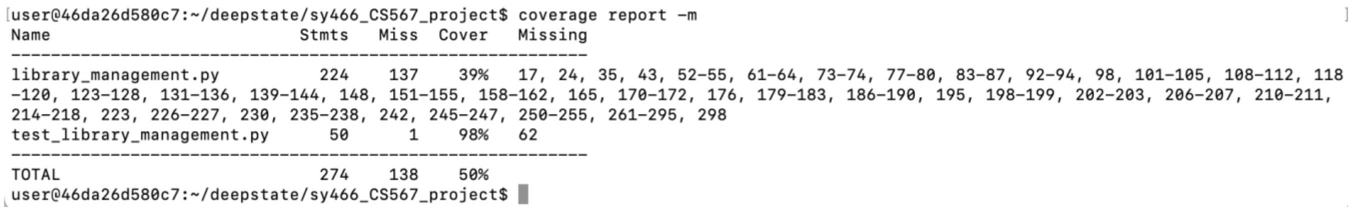
Command: coverage report

After running the tests with coverage, generate a report using the above command.



```
user@46da26d580c7:~/deepstate/sy466_CS567_project$ coverage report
Name                           Stmts   Miss  Cover
--------------------------------------------------
library_management.py            224    137    39%
test_library_management.py        50      1    98%
--------------------------------------------------
TOTAL                            274    138    50%
user@46da26d580c7:~/deepstate/sy466_CS567_project$
```

To get more details about the missing things, use the **command** coverage report -m



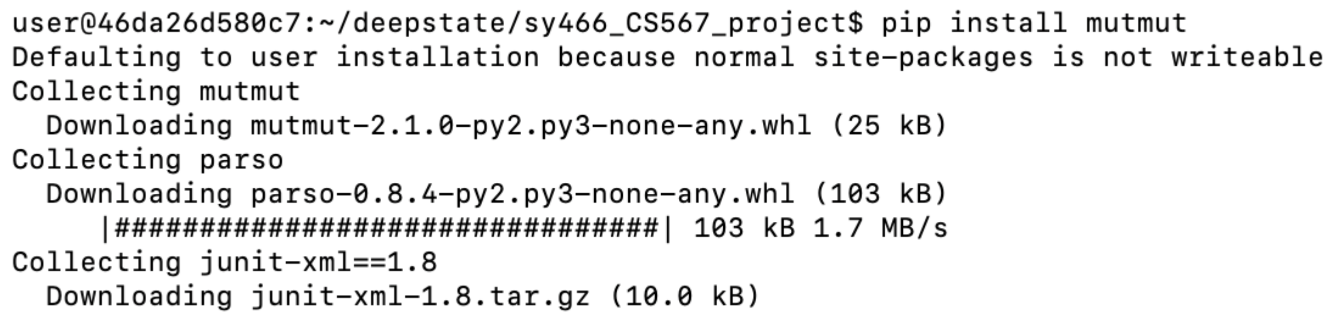
```
user@46da26d580c7:~/deepstate/sy466_CS567_project$ coverage report -m
Name                           Stmts   Miss  Cover   Missing
------------------------------------------------------------
library_management.py            224    137    39%   17, 24, 35, 43, 52-55, 61-64, 73-74, 77-80, 83-87, 92-94, 98, 101-105, 108-112, 118
-120, 123-128, 131-136, 139-144, 148, 151-155, 158-162, 165, 170-172, 176, 179-183, 186-190, 195, 198-199, 202-203, 206-207, 210-211,
214-218, 223, 226-227, 230, 235-238, 242, 245-247, 250-255, 261-295, 298
test_library_management.py        50      1    98%   62
------------------------------------------------------------
TOTAL                            274    138    50%
user@46da26d580c7:~/deepstate/sy466_CS567_project$
```

Mutation Testing:

I have done mutation testing using two tools, mutmut and Universal mutator. Mutmut is used for mutation testing, which involves making small changes to a program's source code to create variants, known as mutants. The goal is to see if the existing tests detect and fail for these mutants. This process helps in evaluating the effectiveness of a test suite by identifying gaps and areas where the tests could be improved.

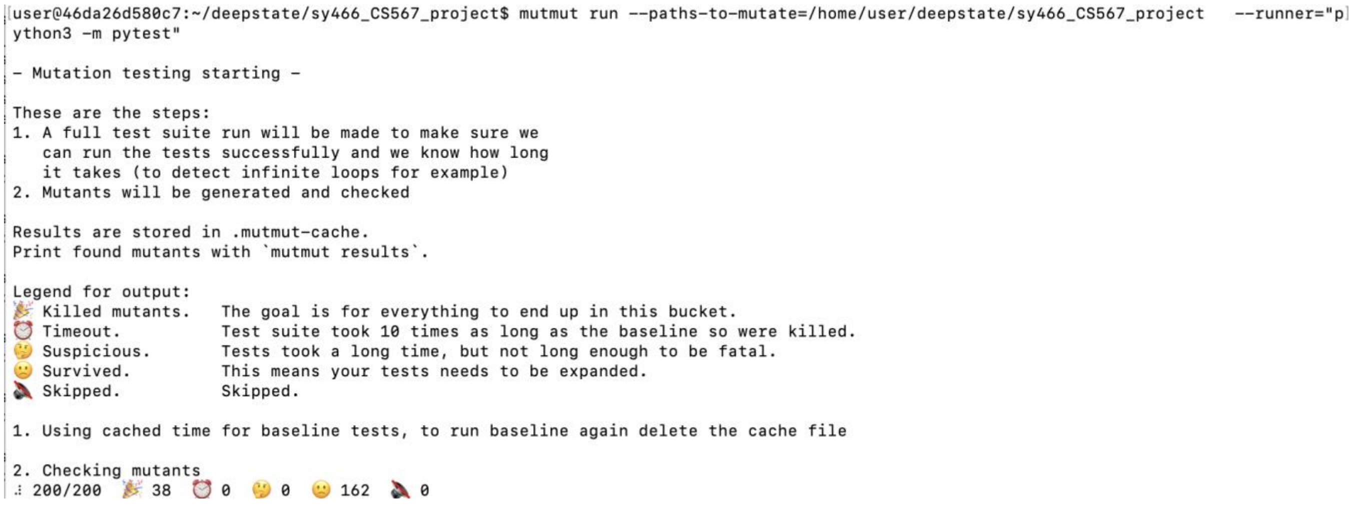
This tool is particularly useful in ensuring that critical parts of the codebase are robustly tested. Developers use Mutmut to help confirm that their tests can effectively catch potential bugs, thereby enhancing the overall quality of the software.

Installing the library using the **command** pip install mutmut



```
user@46da26d580c7:~/deepstate/sy466_CS567_project$ pip install mutmut
Defaulting to user installation because normal site-packages is not writeable
Collecting mutmut
  Downloading mutmut-2.1.0-py2.py3-none-any.whl (25 kB)
Collecting parso
  Downloading parso-0.8.4-py2.py3-none-any.whl (103 kB)
     |################################| 103 kB 1.7 MB/s
Collecting junit-xml==1.8
  Downloading junit-xml-1.8.tar.gz (10.0 kB)
```

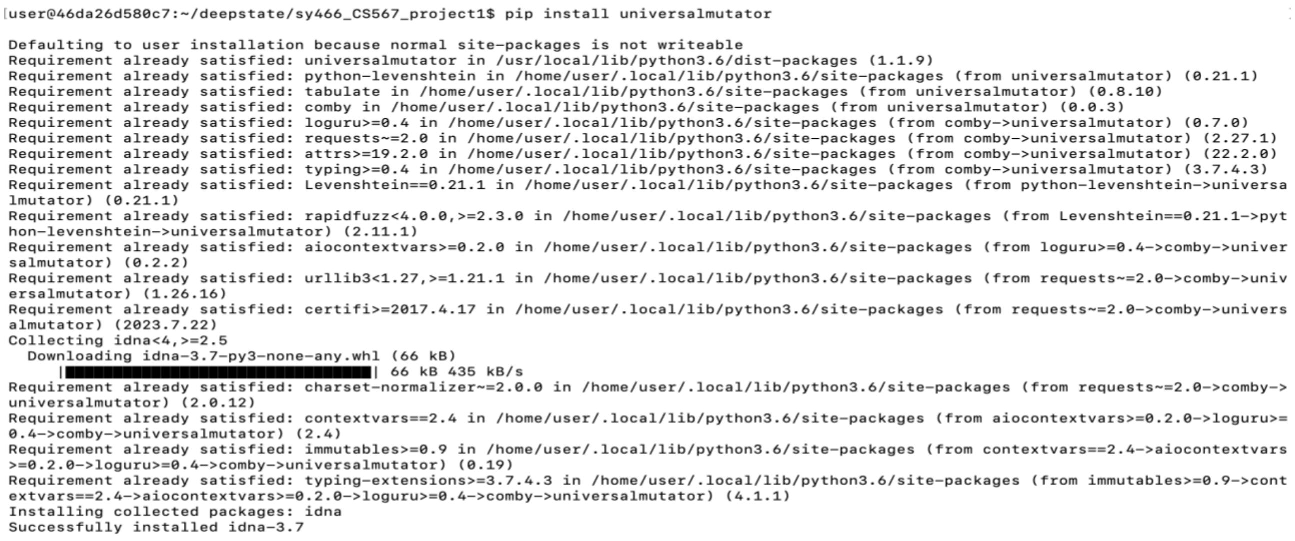
I also installed dependencies for the libraries like pytest, etc.



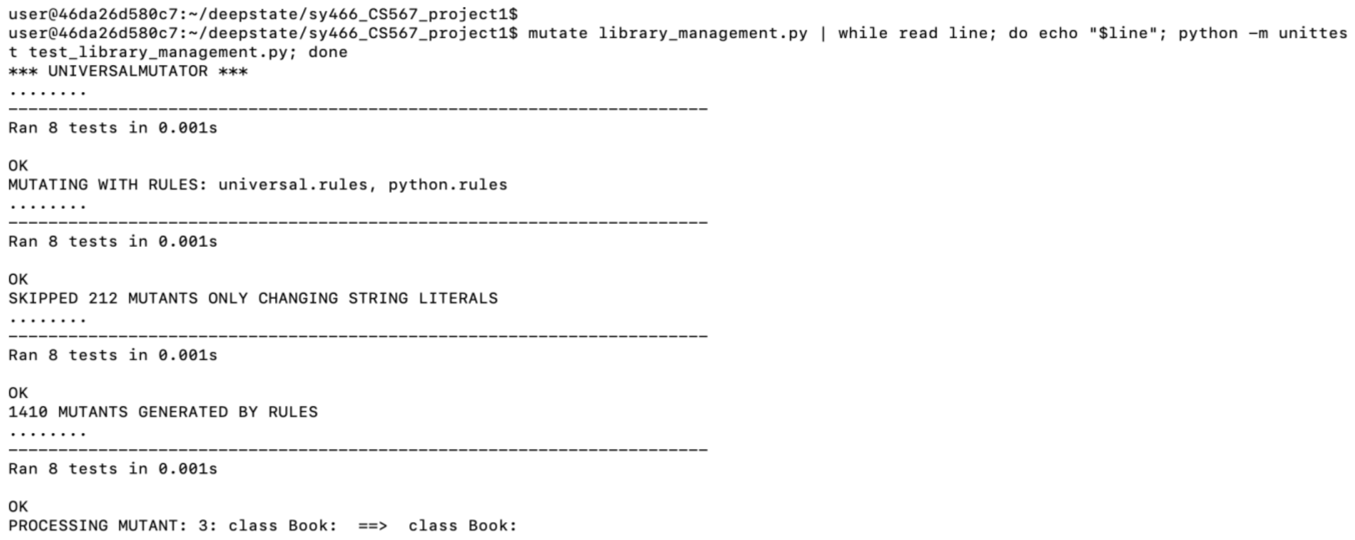
```
user@46da26d580c7:~/deepstate/sy466_CS567_project$ mutmut run --paths-to-mutate=/home/user/deepstate/sy466_CS567_project   --runner="p
ython3 -m pytest"

- Mutation testing starting -

These are the steps:
1. A full test suite run will be made to make sure we
   can run the tests successfully and we know how long
   it takes (to detect infinite loops for example)
2. Mutants will be generated and checked

Results are stored in .mutmut-cache.
Print found mutants with `mutmut results`.

Legend for output:
🎉 Killed mutants.   The goal is for everything to end up in this bucket.
⏰ Timeout.          Test suite took 10 times as long as the baseline so were killed.
🤔 Suspicious.       Tests took a long time, but not long enough to be fatal.
🙁 Survived.         This means your tests needs to be expanded.
🔇 Skipped.          Skipped.

1. Using cached time for baseline tests, to run baseline again delete the cache file

2. Checking mutants
⠼ 200/200  🎉 38  ⏰ 0  🤔 0  🙁 162  🔇 0
```

Universal Mutator:

Employing the universalmutator, the mutation testing process involved introducing changes (mutations) tothe library_management.py code to test the existing test suite's ability to detect and reject these modifications. The mutation testing outcome, as displayed in the image below, resulted in a 32.7880% pass rate of valid mutations. This relatively low rate of detection underscores a critical vulnerability in the test suite's comprehensiveness. The failure to detect a significant number of mutations suggests that the existing tests may not effectively capture all logical errors or unexpected usage scenarios, emphasizing the need for enhancing the test strategies.
Installing the universal mutator by using the below commands.

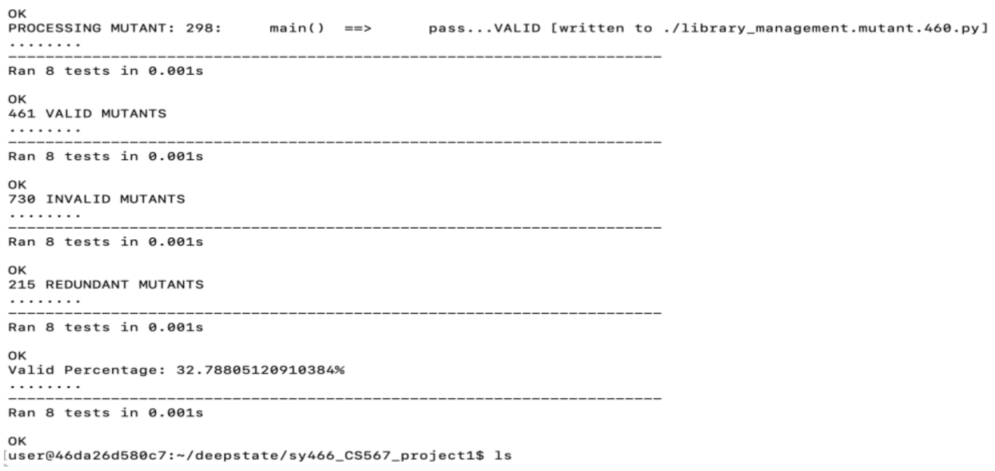


```
user@46da26d580c7:~/deepstate/sy466_CS567_project1$ pip install universalmutator

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: universalmutator in /usr/local/lib/python3.6/dist-packages (1.1.9)
Requirement already satisfied: python-levenshtein in /home/user/.local/lib/python3.6/site-packages (from universalmutator) (0.21.1)
Requirement already satisfied: tabulate in /home/user/.local/lib/python3.6/site-packages (from universalmutator) (0.8.10)
Requirement already satisfied: comby in /home/user/.local/lib/python3.6/site-packages (from universalmutator) (0.0.3)
Requirement already satisfied: loguru>=0.4 in /home/user/.local/lib/python3.6/site-packages (from comby->universalmutator) (0.7.0)
Requirement already satisfied: requests~=2.0 in /home/user/.local/lib/python3.6/site-packages (from comby->universalmutator) (2.27.1)
Requirement already satisfied: attrs>=19.2.0 in /home/user/.local/lib/python3.6/site-packages (from comby->universalmutator) (22.2.0)
Requirement already satisfied: typing>=0.4 in /home/user/.local/lib/python3.6/site-packages (from comby->universalmutator) (3.7.4.3)
Requirement already satisfied: Levenshtein==0.21.1 in /home/user/.local/lib/python3.6/site-packages (from python-levenshtein->universa
lmutator) (0.21.1)
Requirement already satisfied: rapidfuzz<4.0.0,>=2.3.0 in /home/user/.local/lib/python3.6/site-packages (from Levenshtein==0.21.1->pyt
hon-levenshtein->universalmutator) (2.11.1)
Requirement already satisfied: aiocontextvars>=0.2.0 in /home/user/.local/lib/python3.6/site-packages (from loguru>=0.4->comby->univer
salmutator) (0.2.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/user/.local/lib/python3.6/site-packages (from requests~=2.0->comby->univ
ersalmutator) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /home/user/.local/lib/python3.6/site-packages (from requests~=2.0->comby->univers
almutator) (2023.7.22)
Collecting idna<4,>=2.5
  Downloading idna-3.7-py3-none-any.whl (66 kB)
     |████████████████████████████████| 66 kB 435 kB/s
Requirement already satisfied: charset-normalizer~=2.0.0 in /home/user/.local/lib/python3.6/site-packages (from requests~=2.0->comby->
universalmutator) (2.0.12)
Requirement already satisfied: contextvars==2.4 in /home/user/.local/lib/python3.6/site-packages (from aiocontextvars>=0.2.0->loguru>=
0.4->comby->universalmutator) (2.4)
Requirement already satisfied: immutables>=0.9 in /home/user/.local/lib/python3.6/site-packages (from contextvars==2.4->aiocontextvars
>=0.2.0->loguru>=0.4->comby->universalmutator) (0.19)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /home/user/.local/lib/python3.6/site-packages (from immutables>=0.9->cont
extvars==2.4->aiocontextvars>=0.2.0->loguru>=0.4->comby->universalmutator) (4.1.1)
Installing collected packages: idna
Successfully installed idna-3.7
```



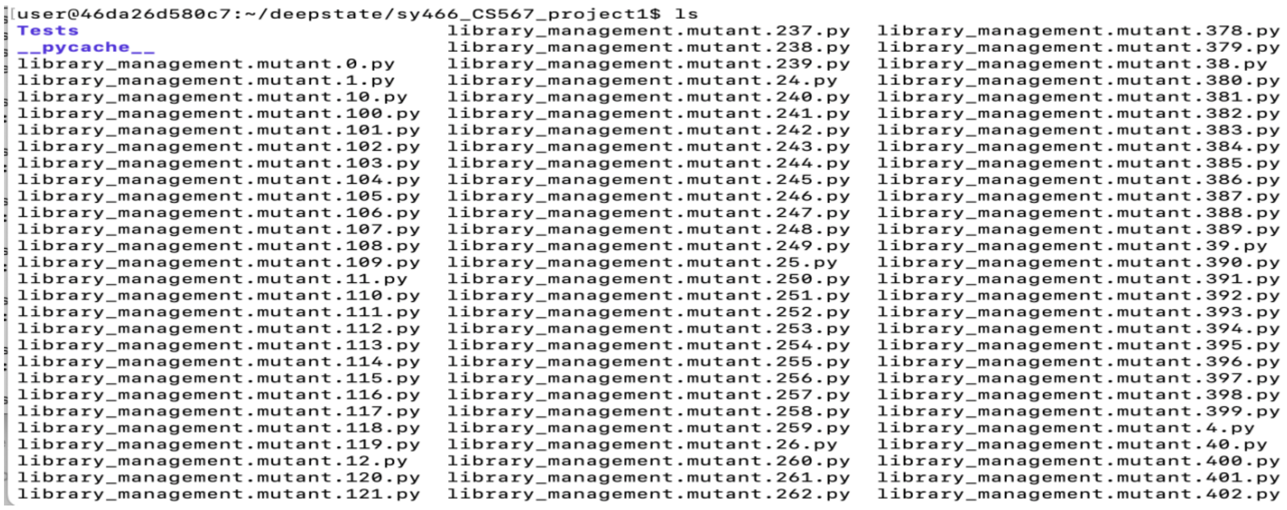
```
user@46da26d580c7:~/deepstate/sy466_CS567_project1$
user@46da26d580c7:~/deepstate/sy466_CS567_project1$ mutate library_management.py | while read line; do echo "$line"; python -m unittes
t test_library_management.py; done
*** UNIVERSALMUTATOR ***
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
MUTATING WITH RULES: universal.rules, python.rules
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
SKIPPED 212 MUTANTS ONLY CHANGING STRING LITERALS
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
1410 MUTANTS GENERATED BY RULES
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
PROCESSING MUTANT: 3: class Book:  ==>  class Book:
```

From the picture below, we can see 461 valid mutants, 730 invalid mutants, and 215 redundant mutants with a valid percentage of 32.788%.



```
OK
PROCESSING MUTANT: 298:     main()  ==>      pass...VALID [written to ./library_management.mutant.460.py]
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
461 VALID MUTANTS
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
730 INVALID MUTANTS
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
215 REDUNDANT MUTANTS
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
Valid Percentage: 32.78805120910384%
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
user@46da26d580c7:~/deepstate/sy466_CS567_project1$ ls
```

The following are mutants created by the above commands.



```
user@46da26d580c7:~/deepstate/sy466_CS567_project1$ ls
Tests                              library_management.mutant.237.py  library_management.mutant.378.py
__pycache__                        library_management.mutant.238.py  library_management.mutant.379.py
library_management.mutant.0.py     library_management.mutant.239.py  library_management.mutant.38.py
library_management.mutant.1.py     library_management.mutant.24.py   library_management.mutant.380.py
library_management.mutant.10.py    library_management.mutant.240.py  library_management.mutant.381.py
library_management.mutant.100.py   library_management.mutant.241.py  library_management.mutant.382.py
library_management.mutant.101.py   library_management.mutant.242.py  library_management.mutant.383.py
library_management.mutant.102.py   library_management.mutant.243.py  library_management.mutant.384.py
library_management.mutant.103.py   library_management.mutant.244.py  library_management.mutant.385.py
library_management.mutant.104.py   library_management.mutant.245.py  library_management.mutant.386.py
library_management.mutant.105.py   library_management.mutant.246.py  library_management.mutant.387.py
library_management.mutant.106.py   library_management.mutant.247.py  library_management.mutant.388.py
library_management.mutant.107.py   library_management.mutant.248.py  library_management.mutant.389.py
library_management.mutant.108.py   library_management.mutant.249.py  library_management.mutant.39.py
library_management.mutant.109.py   library_management.mutant.25.py   library_management.mutant.390.py
library_management.mutant.11.py    library_management.mutant.250.py  library_management.mutant.391.py
library_management.mutant.110.py   library_management.mutant.251.py  library_management.mutant.392.py
library_management.mutant.111.py   library_management.mutant.252.py  library_management.mutant.393.py
library_management.mutant.112.py   library_management.mutant.253.py  library_management.mutant.394.py
library_management.mutant.113.py   library_management.mutant.254.py  library_management.mutant.395.py
library_management.mutant.114.py   library_management.mutant.255.py  library_management.mutant.396.py
library_management.mutant.115.py   library_management.mutant.256.py  library_management.mutant.397.py
library_management.mutant.116.py   library_management.mutant.257.py  library_management.mutant.398.py
library_management.mutant.117.py   library_management.mutant.258.py  library_management.mutant.399.py
library_management.mutant.118.py   library_management.mutant.259.py  library_management.mutant.4.py
library_management.mutant.119.py   library_management.mutant.26.py   library_management.mutant.40.py
library_management.mutant.12.py    library_management.mutant.260.py  library_management.mutant.400.py
library_management.mutant.120.py   library_management.mutant.261.py  library_management.mutant.401.py
library_management.mutant.121.py   library_management.mutant.262.py  library_management.mutant.402.py
```

Each of these testing methodologies contributes uniquely to understanding and improving the Library Management System. The unit tests establish a foundation of functionality, the coverage tests highlight areas lacking in scrutiny, and the mutation tests provide insight into the test suite's overall resilience. These comprehensive testing procedures are vital for ensuring that the system not only meets the current requirements but also maintains a robust defense against future errors or changes in functionality.

Conclusion:

The comprehensive testing of the Library Management System has yielded valuable insights into both its strengths and areas requiring improvement. The unit tests have effectively validated the core functionalities, such as checking books in and out and managing the library's catalog, ensuring these operations are robust under typical usage scenarios. However, the coverage and mutation tests have highlighted significant deficiencies that could potentially undermine the system's reliability and user experience if unaddressed.

The coverage tests indicated that 50% of the codebase remains untested; this is concerning as it suggests
that critical parts of the system might fail under unexpected conditions, which were not simulated during
testing. Particularly, the error handling and complex user interactions may not behave as intended when
faced with edge cases or erroneous inputs.

Moreover, the results from the mutation testing, with only a 32.788% pass rate, reveal a pressing need to enhance the test suite. This low pass rate points to susceptibility in detecting and addressing faults introduced by minor code modifications. It is imperative that the test suite is expanded to cover more scenarios and include more stringent checks, especially focusing on those sections of the code identified
by the coverage tests as lacking in coverage.

To elevate the system's reliability and maintainability, it is recommended to:

- Increase the breadth and depth of both unit and integration tests to cover untested code segments.
- Develop more rigorous error handling and validation mechanisms to improve system resilience and user experience.
- Continually revise and refine the testing suite in response to new requirements or detected shortcomings, as identified by ongoing mutation testing.

By addressing these areas, the Library Management System can achieve higher standards of performance and reliability, ensuring it robustly supports the needs of its users and adapts effectively to future requirements. These improvements will not only enhance the current usability of the system but also safeguard its evolution over time.