# 5. Security and Testing

## 5.1 Software Testing Practices and Security Measures

To ensure the reliability and robustness of the Project Review Scheduler (PRS), a comprehensive set of software testing practices and security measures were adopted throughout the Software Development Life Cycle (SDLC). These practices helped detect defects early, validate logic, and secure data handling.  
  
The project used a combination of black-box and white-box testing approaches:

• Black-box testing was employed to verify expected system behavior without knowledge of the internal code. For example, equivalence class partitioning and boundary value analysis were used when testing date-based review scheduling logic to ensure correct review intervals (Real Python, 2023).

Example from `calculate\_due\_date()` function:

if days\_until\_review < 0:  
 status = 'Overdue'  
elif days\_until\_review <= 30:  
 status = 'Due Soon'  
else:  
 status = 'Up to Date'

• White-box testing was applied using Python’s unittest framework to ensure internal paths, decision branches, and exception conditions were evaluated. Each test targeted specific logic paths within functions such as `assign\_reviewer()` and `calculate\_due\_date()` (BrowserStack, n.d.).

Example unit test for `calculate\_due\_date()` logic:

def test\_due\_date\_six\_months(self):  
 project = {  
 'Last\_Review\_Date': '2025-01-01',  
 'Review\_Frequency\_Years': '0.5'  
 }  
 result = calculate\_due\_date(project, current\_date='2025-05-01')  
 self.assertEqual(result['Next\_Review\_Date'], '2025-07-01')

Unit tests such as TC\_DR\_001 (Due Date Calculator) and TC\_RA\_006 (Graceful fallback for reviewer assignment) were implemented. Mocking was used for file operations and email server calls, enabling repeatable and isolated tests.

### Security Measures

In alignment with secure SDLC principles (Alenezi & Almuairfi, 2019), the following security measures were implemented:

• Input validation for CSV file fields such as dates and department names prevented malformed or malicious data.  
• Environment-based secrets management used a `.env` file to protect sensitive credentials.  
• Error handling ensured graceful fallback, for example using same-department reviewers only if necessary.  
• Modular design enabled testing each component independently and securely.

## 5.2 Test Cases and Their Contribution

Test cases helped ensure business logic correctness and system reliability:

• TC\_DR\_001 verified status categorization and scheduling date logic.

• TC\_RA\_006 validated fallback behavior when no suitable reviewers were available.

• TC\_NT\_002 confirmed overdue emails sent with urgency flags. Mock SMTP servers enabled validation.

• TC\_RP\_004 ensured CSV reports were formatted and sorted as required. This supported decision-making.

Each test improved trust in system outputs and resilience. Screenshots and outputs are included in the appendix.

# 7. Glossary

|  |  |
| --- | --- |
| Term | Definition |
| Black-box Testing | Testing functionality without internal code knowledge. |
| White-box Testing | Testing based on internal code and logic structure. |
| Unit Test | A test for a single function or unit. |
| Mocking | Simulating components such as file access or SMTP servers. |
| Secure SDLC | Integrating security at each SDLC phase. |
| SMTP | Protocol for sending email messages. |
| Boundary Value Analysis | Testing at boundary conditions of input ranges. |

# References

Alenezi, M., & Almuairfi, S. (2019). Security Risks in the Software Development Lifecycle. International Journal of Recent Technology and Engineering, 8(3), 7048–7055. https://doi.org/10.35940/ijrte.C5374.098319

BrowserStack. (n.d.). Unit Testing in Python: Best Practices. Retrieved May 12, 2025, from https://www.browserstack.com/guide/unit-testing-python

Real Python. (2023). Python Testing with unittest, pytest and more. https://realpython.com/python-testing/