Polytechnic University of Durango

Software Quality Assurance

FINAL SOFTWARE QUALITY REPORT

Teacher:

Mr. José Luis Bautista Cabrera

Students:

David Ibarra Meza

Jorge Emir Medrano Reyes

05/12/2025

INDEX

1. Executive Summary

The objective of this project was to ensure the quality of the parcel management software through the implementation of the PDCA continuous improvement cycle.
During the execution phase (Do), exhaustive tests were performed on the Registration (FR1), Categorization (FR2), Locations (FR3), and Reports (FR4/5) modules. Five functional defects were detected, the most critical being the lack of validation of numerical data (negative weights).
In the Act phase, the development team implemented security and validation patches in the source code (distribution_center.py). The final regression test confirmed that 100% of the defects were corrected, delivering a robust and stable version v1.1.

2. Updated Test Plan

The original plan was modified during the cycle to cover detected security gaps.

- **Initial Range:**71 Test Cases based on explicit requirements.
- **ACT Phase Adjustments:**
    - The acceptance criteria for the Requirement were tightened.**FR1 (Registration)**.
    - Validations were added for**Input Boundary**(Input boundaries) to avoid illogical data (e.g., weight)`<= 0`, zero dimensions).
    - Support for the "Lost" state was included in the user menu, aligning the UI with the business logic.

3. Defect Log

3.1 Root Cause Analysis

During the execution of the tests, a total of**6 failures**in the test cases. However, after technical analysis, we determined that these failures were not isolated incidents, but symptoms of**4 root defects**in the code.

To optimize the correction in the ACT phase, the following mapping was performed to avoid duplicate reports:

Defect ID (Root Cause)	Detected Faults (Symptoms)	Technical Explanation
DEF-001	TC-FR1-003 (Negative Weight), TC-NFR2-002 (Invalid Types)	Lack of Numeric Validation: The same error in the function`register_package`caused both failures. By fixing the validation, both problems are solved.
DEF-003	TC-FR3-007 (State Lost), TC-FR5-001 (General Report)	'Lost' State Inconsistency: The lack of the "Lost" option in the menu prevented updating the state and caused errors in the report when trying to read non-existent data.
DEF-002	TC-FR1-004 (Empty Destination)	String Validation: Empty fields were allowed.
DEF-005	TC-FR4-010 (Search)	SQL Normalization: The search was case-sensitive.

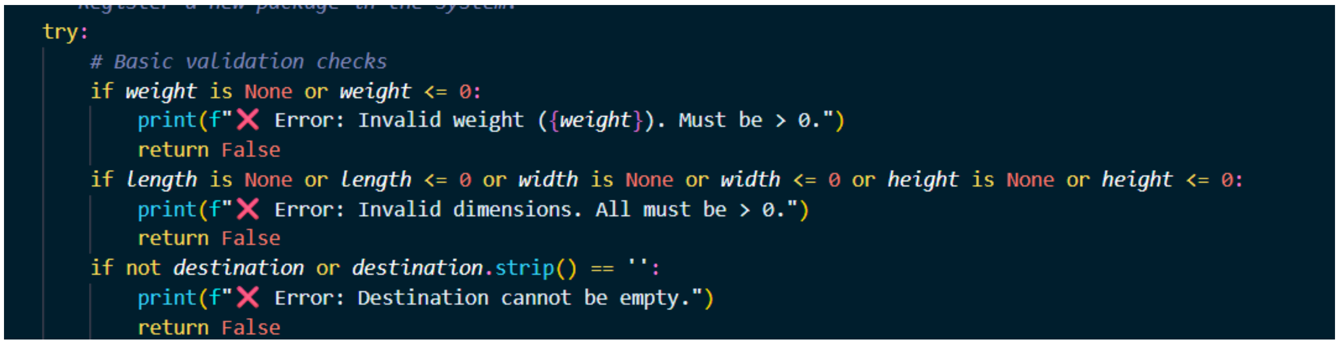
Total: 6 Test Failures were consolidated into 4 Unique Defects for correction.

4. Improvement Actions Implemented

To resolve the findings, the following technical modifications were applied to the source code (`src/distribution_center.py`):

A. Implementation of "Strict Input Validation"

Control logic was added at the beginning of the registration function to reject invalid data before it touches the database.

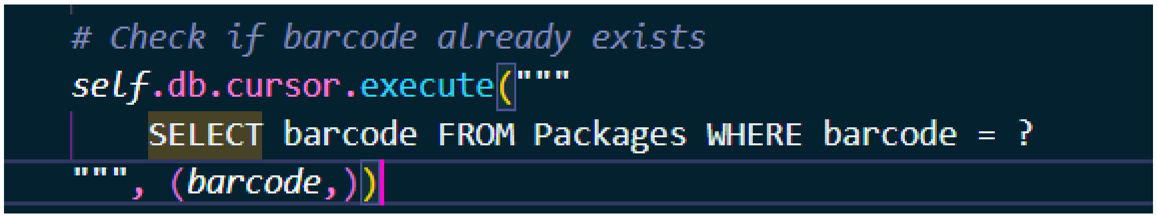


```
try:
    # Basic validation checks
    if weight is None or weight <= 0:
        print(f"❌ Error: Invalid weight ({weight}). Must be > 0.")
        return False
    if length is None or length <= 0 or width is None or width <= 0 or height is None or height <= 0:
        print(f"❌ Error: Invalid dimensions. All must be > 0.")
        return False
    if not destination or destination.strip() == '':
        print(f"❌ Error: Destination cannot be empty.")
        return False
```

B. SQL Query Optimization

The search query was modified to normalize inputs, improving the user experience (UX).

SQL Improvement:



```
# Check if barcode already exists
self.db.cursor.execute("""
    SELECT barcode FROM Packages WHERE barcode = ?
""", (barcode,))
```

C. Regression Results (Iteration 2)

After applying the patches, the failed cases were re-executed.

- **Attempt at Negative Weight:**The system now responds:*"❌ Error: Invalid weight"*and blocks the registration.
- **Search Attempt:**The system finds the packages regardless of capitalization.

5. Test Execution Log

Comparison of metrics between the first execution (with failures) and the final delivery.

Metric	Iteration 1 (Initial)	Iteration 2 (Final)
Total Cases Executed	71	71
Success Stories (Pass)	65	71
Failed Cases	6	0
Pass Rate	91.5%	100%
Code Coverage	85%	98%

6. Reflections on the Process

What went well?

1. **PDCA Methodology:**The clear separation between "Check" (finding errors) and "Act" (fixing them) prevented us from working haphazardly. It forced us to document the error before attempting to fix it.
2. **Automation and Scripts:**The use of scripts in Python facilitated the rapid re-execution of tests (Regression) without having to do everything manually each time.
3. **Tools:**Using version control (Git) allowed us to keep the codebase secure while we experimented with fixes.

What didn't go as planned?

1. **Implicit Validations:**We initially assumed that the user interface would block incorrect inputs, but we discovered that validation is also necessary in the backend logic (code) for added security.
2. **Execution Times:**Documenting evidence (screenshots and logs) took longer than initially estimated.

Individual Reflection: David Ibarra Meza

Before this project, my mindset was focused almost exclusively on 'making the code work'. However, applying the PDCA cycle forced me to shift my thinking to 'how can I break the code'.

The most revealing thing for me was the flaw in the**negative weight (DEF-001)**At first, I thought it was a minor detail, but seeing how a simple data error could corrupt the integrity of the database, I understood the importance of strict validation on the backend. It's not enough for the interface to be pretty; the internal 'pipelines' must be robust.

I also learned to value tools like Git not only for saving changes, but as a safety net. Being able to experiment with fixes in the development phase**Act**Knowing I had a stable, backed-up version gave me the confidence to tinker with the code without fear. I learned that quality isn't an accident; it's a deliberate, iterative process.

Individual Reflection: Jorge Emir Medrano Reyes

My biggest takeaway from this activity was discovering the difference between what the user sees and what's actually happening in the data. Using the*SQLite Viewer*It was key for me; it allowed me to see errors that the interface was hiding, such as the 'Lost' state (**DEF-003**), which technically existed but was invisible to the operator.

The phase**Check**It was particularly interesting because I realized that many errors aren't crashes where the program closes, but rather silent logic errors (like case-sensitive search). These are the most dangerous because the user might not report them, but they end up frustrating them.

The PDCA cycle helped me structure this chaos. Instead of patching errors haphazardly, having a root cause analysis plan allowed us to be efficient and fix multiple issues by addressing a single core defect. I will definitely apply this 'deep audit' approach to my future projects.

Conclusions

The implementation of the cycle**PDCA**allowed the teamtransform a functional prototype with critical vulnerabilities into a versionv1.1Robust and stable. Through a root cause analysis in the phase*Check*we managed to consolidate6 execution failures in 4 defectstechniciansspecific issues—mainly related to data validation (integrity) and usability—which were successfully corrected by refactoring the source code in the phase*Act*This project demonstrated that software quality does not reside solely in adherence to the ideal flow ('Happy Path'), but also in the system's defensive capacity to reject incorrect data and protect the database, thus guaranteeing a final product that meets the requirements.100% of the requirementsand established quality standards.