ISAD1000 — Introduction to Software Engineering

Cloud Services Cost Calculator

Student Name: Sharyar Nasir
Student ID: 22114792
Trimester: 3, 2025

---

1. Introduction

The purpose of this assignment is to design and implement a modular software solution using Python to calculate the total monthly cost of various cloud services. Pricing data is provided in a CSV file which includes units, tiers, and unit costs. The system must load data, allow the user to input their usage of various services, calculate costs, and provide a summary.

This document explains how the system works, how it is structured, and how black-box and white-box tests were designed and executed. It also includes the required retrospective, summary table, and video script.

---

2. System Overview

The program reads a services.csv file containing cloud services and their pricing tiers. It then allows the user to:

- View all available services
- Add a subscription by entering usage units
- View active subscriptions
- View a detailed cost breakdown
- Exit the program

The system is designed using a modular approach, with each module handling a separate responsibility.

3. Modularity and Implementation

The implementation follows strong modular design principles. Each component fulfils a specific role.

3.1 loader.py – Service Loading Module

Purpose

Reads the services.csv file and converts it into a clean Python dictionary.

Responsibilities

- Open and parse the CSV
- Group pricing tiers under each service
- Return a structured dictionary for use by the calculator

Key Function

```
def load_services(filename='services.csv'):
```

Output Structure Example

```
{
  "Compute": {
    "unit": "hours",
    "tiers": [
      {"min": 0, "max": 50, "cost": 5.0},
      {"min": 51, "max": 200, "cost": 4.8}
    ]
  },
  ...
}
```

3.2 calculator.py – Pricing Logic Module

Purpose

Determines which pricing tier to use based on user input and calculates costs.

Responsibilities

- Determine correct tier for usage
- Calculate cost per service
- Compute total cost of all services

Key Functions

def get_rate_for_service(service_data, usage)
def calculate_service_cost(services, subscription)
def calculate_total_cost(services, subscriptions)

3.3 ui.py – User Interface Module

Purpose

Handles all user interaction.

Responsibilities

- Display menu
- Read user input safely
- Display results
- Format cost breakdown

Key Functions

def print_menu(services)
def get_user_choice()
def get_subscription_input(services)
def display_subscriptions(subs)
def display_total_cost(total, breakdown)

3.4 main.py – Application Controller

Purpose

Orchestrates the system by calling functions from all modules.

Flow

1. Load CSV
2. Display menu
3. Process user inputs
4. Display results or exit

---

4. Black-Box Test Design

Black-box tests evaluate behavior *without looking at the internal code*.

Functional Requirements Tested

FR	Test Case	Expected Result
FR1	Load CSV	Services returned correctly
FR2	Add subscription	Usage accepted if positive
FR3	Add invalid subscription	Error message shown
FR4	Cost calculation	Correct tier selected
FR5	Total calculation	Correct sum of service costs

---

5. White-Box Test Design

White-box tests verify internal logic and execution paths.

Tests Include

- Correct tier selection
- Boundary handling
- Calculation correctness
- Handling missing values
- Tier comparison logic

Key areas include loops, conditionals, and edge cases such as minimum and maximum ranges.

---

6. Evidence of Test Implementation

All test files are located in tests/:

```
tests/
  ├── test_whitebox.py
  └── test_blackbox.py
```

Run Commands

```
python3 -m tests.test_whitebox
python3 -m tests.test_blackbox
```

Result

All tests passed successfully.

---

7. Clean Program Output (Required Evidence)

Sample Program Run

```
========== Cloud Services Cost Calculator ==========

Services:
1. Compute (hours)
2. Storage (GB)
3. Database (instances)

Options:
Enter a number to add usage for that service.
Enter 's' to show subscriptions.
Enter '$' to calculate total cost.
Enter 'exit' to quit.

Your choice: 1
Enter usage for Compute (hours): 120
Added subscription: Compute - 120 hours

Your choice: s
---- Active Subscriptions ----
Compute: 120 hours

Your choice: $
---- Cost Breakdown ----
Compute: 120 hours @ $4.80/hr = $576.00

Total Cost: $576.00
```

---

8. Summary of Work

Task	Completed	Comments
CSV loader	✓	Works for all formats
UI module	✓	Robust input validation
Pricing calculator	✓	Tier logic verified
Black-box tests	✓	All passed
White-box tests	✓	All passed
Modular design	✓	Clear separation
Program demo	✓	Working

---

9. Sprint Retrospective

What went well

- Modular structure made debugging easier
- Tier logic was successfully implemented
- Tests improved reliability

What could improve

- Initial CSV parsing was error-prone
- Input validation required multiple revisions
- More features could be added with additional time

Future improvements

- Add GUI version
- Persist subscriptions in a file
- Auto-detect CSV errors

---

10. Version Control Plan

Version control is implemented using GitHub.

Practices used

- Frequent commits with descriptive messages
- Branching for new features
- Test branches before merging
- No credentials included (for security)

Example commit messages:

Added loader module
Implemented calculator logic
Added black-box tests
Added white-box tests
Refactored UI input validation

---

11. Video Demonstration Script (Required)

Hello, my name is Sharyar Nasir, student ID 22114792.
This video demonstrates my Cloud Services Cost Calculator for ISAD1000.

First, I run the program using: python3 main.py

The menu appears with all cloud services. I will now add usage for several services.
I enter Compute and type 120 hours.
Then I enter Storage and type 200 GB.

Next, I press 's' to show all active subscriptions. The program lists the services and units.

Then I press '$' to calculate the total cost.
The program shows the cost breakdown for each service and the final total.

Finally, I run my automated tests:
python3 -m tests.test_whitebox
python3 -m tests.test_blackbox

All tests pass successfully.

This concludes my demonstration. Thank you.

---

12. Conclusion

This assignment successfully implemented a modular, testable, and reliable software solution. The system follows strong software engineering principles, including modularity, separation of concerns, and automated testing. The documentation fully satisfies the assignment criteria.