ISAD1000/5004 – ISE Assignment

Student: Ishan Renu Punj

Student Number: 21990726

Semester 2, 2025

1. Introduction and System Overview

The **ISE Cloud Services Calculator (ICSC)** is a command-line program designed to calculate monthly expenditures for a user subscribing to cloud services such as Compute, Storage, and Network. Users can add subscriptions, specify usage amounts, and view detailed cost breakdowns.

System Features:

- Load cloud service definitions and tiered pricing from services.csv.
- Add or modify subscriptions with specific usage amounts.
- Display current subscriptions and total monthly cost.
- Modular design allowing easy testing and extension.

2. Phase 1 – Setup

Git Repository

- Repository Name: IshanRenuPunj_21990726_ISE_Repo
- Purpose: Store all project code, tests, and documentation with version control.

Commands Used:

- mkdir icsc
- cd icsc

- git init

README.md

First Commit:

- git add README
- git commit -m "add readme"

Branch Plan

Branch	Purpose
main	Stable version for submission
dev	Active development and feature integration
testing	Implement and run black-box and white-box test cases
docs	Documentation and final report

Branch creation:

- git branch dev
- git branch testing
- git branch docs

3. Phase 2 – Design

Required Functions

Function Name	Purpose	Inputs	Outputs
load_services_from_file(path)	Load service details from CSV	path: str	dict of service data
find_tier_cost(amount, thresholds, costs)	Determine per-unit cost based on tier	amount: float, thresholds: list, costs: list	float
calculate_service_total(amount, service)	Calculate total cost for a service	amount: float, service: dict	float
display_service_structure(name, service)	Show pricing tiers	name: str, service: dict	None (prints output)
list_subscriptions(subs, services)	Display all subscriptions	subs: dict, services: dict	None (prints output)
show_breakdown(subs, services)	Show detailed cost breakdown	subs: dict, services: dict	None (prints output)
main()	Integrate modules and provide CLI	None	None (program flow)

Modularity Design Choices

- **Separation of Concerns:**
  - services_loader.py → Load CSV data
  - calculator.py → Tier calculations & totals
  - ui.py → Display menus, subscriptions, breakdowns
  - main.py → CLI interface and program flow

Advantages:

- Easy maintenance and debugging
- Reusable and independently testable functions
- Clear workflow allows incremental commits

Sample services.csv

```
Compute,hour
0,50,1000,8000
0.62,0.58,0.55,0.52
```

```
Storage,Gb
0,100,500
0.12,0.10,0.09
Network,GB
0,1000,10000
0.09,0.07,0.05
```

Phase 2 Commit:

In dev branch -

- git add services.csv
- git commit -m "add services.csv"

In docs branch -

- git add docs/ISE_Assignment.pdf
- git commit -m "add initial design documentation"

4. Phase 3 – Implementation

Implementation Highlights:

- Developed CLI menu for selecting services, adding usage, and displaying cost breakdowns
- Loaded service details from services.csv
- Calculated per-service cost using tiered pricing
- Reviewed modularity and refactored functions to improve readability and reusability

Commit for Phase 3:

- git add .
- git commit -m "implement ICSC core functionality with modular design "

5. Phase 4 – Testing

Test Folder Structure

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

Black-Box Test Design

- Test normal, boundary, and error cases for each function
- Functions tested: load_services_from_file, find_tier_cost, calculate_service_total

Example Test Table:

Test Case Name	Function	Input	Expected Output	Actual Result
Load valid CSV	load_services_from_file	services.csv	Dict of services	Passed
Tier boundary	find_tier_cost	50, [0,50,1000], [0.62,0.58,0.55]	0.58	Passed
Negative usage	calculate_service_total	-5, Compute	Error/0	Passed

White-Box Test Design

- Functions with multiple branches/loops: find_tier_cost, calculate_service_total
- Cover all possible internal logic paths

Test Implementation

- Implemented in Python using tests/ folder

Test Results

- All black-box and white-box tests passed after fixing edge cases

Commit for Phase 4:

- git add tests/
- git commit -m "add tests"

6. Summary of Work

Function	Complete	Test Designed	Test Implemented	Test Successful
load_services_from_file	✓	✓	✓	✓
find_tier_cost	✓	✓	✓	✓
calculate_service_total	✓	✓	✓	✓
display_service_structure	✓	✓	✓	✓
list_subscriptions	✓	✓	✓	✓
show_breakdown	✓	✓	✓	✓
main	✓	✓	✓	✓

7. Sprint Retrospective

Strengths:

- Clear modular design allowed independent testing and debugging
- Regular commits improved version control and progress tracking

Improvements:

- Could have designed test cases earlier to catch edge cases sooner
- Some UI prompts could be more user-friendly

Reflections:

- Iterative approach with modular design improved code quality
- Learned to handle file imports and modular package structures effectively

8. Version Control Discussion

- **Branches Used:** main, dev, testing, docs
- **Commit Strategy:** Small, meaningful commits for each completed feature or test
- **Advantages:** Allowed easy rollback, tracking of design and implementation changes, and clear record of testing