Topic C: Automation on Desktop Apps

Group 15:

20125075 - Hà Gia Khang
20125123 - Hoàng Thanh Tùng
21125131 - Huỳnh Hoàng Phúc

Table of contents

Automation on Desktop Apps

01 Theory

An introduction to theory, concepts, and strategies

Introduction

01. Manual vs. Automation

Manual Testing

- ✓ Performed by humans step-by-step
- ✓ Best for **Exploratory** & **UX** testing.
- ✓ Prone to fatigue & inconsistency.

Automation Testing

- ✓ Executed by scripts/tools.
- ✓ InBest for **Regression** & **Repetitive** tasks.
- ✓ Fast, precise, consistent.

"Automation does not replace humans; it liberates them."

Introduction

02. What is Automation Testing?

Software Verifying Software

Automation testing is the technique of using specific software (scripts) to control the execution of tests and compare actual outcomes with predicted outcomes.

- Eliminates repetitive manual tasks.
- Increases test coverage.
- Validates software reliability automatically.

Goal: Liberate testers from boring tasks to focus on difficult cases.

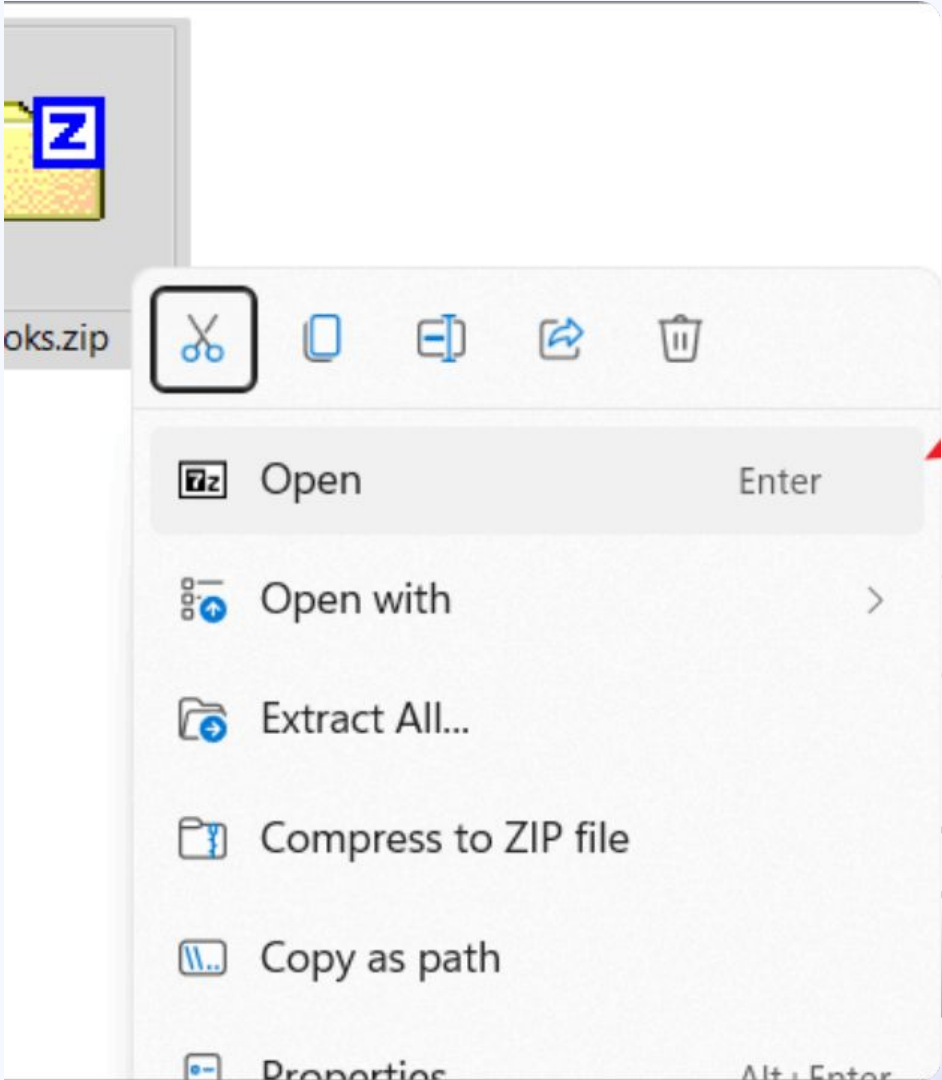
03. Desktop App Automation

Context Specifics

Verifying applications installed directly on the Operating System (unlike Web apps in browsers).

Key aspects:

- **GUI Actions:** Clicking, dragging, typing on Windows/Dialogs.
- **CLI Actions:** Open CMD , type the command and press enter (MAC OS)
- **Environments:** Validating on Windows Forms, WPF, JavaFX, etc.
- **System Access:** Interacting with File System & OS permissions.



04. Why do we do it?

Repeatability

Scripts can be executed indefinitely without fatigue,

Accuracy

Eliminates human error. A machine will strictly follow the script, ensuring precise validation of complex GUI workflows.

Speed & Effort

Drastically reduces manual effort and accelerates regression testing cycles, allowing faster release of new versions.

05. Common Myths

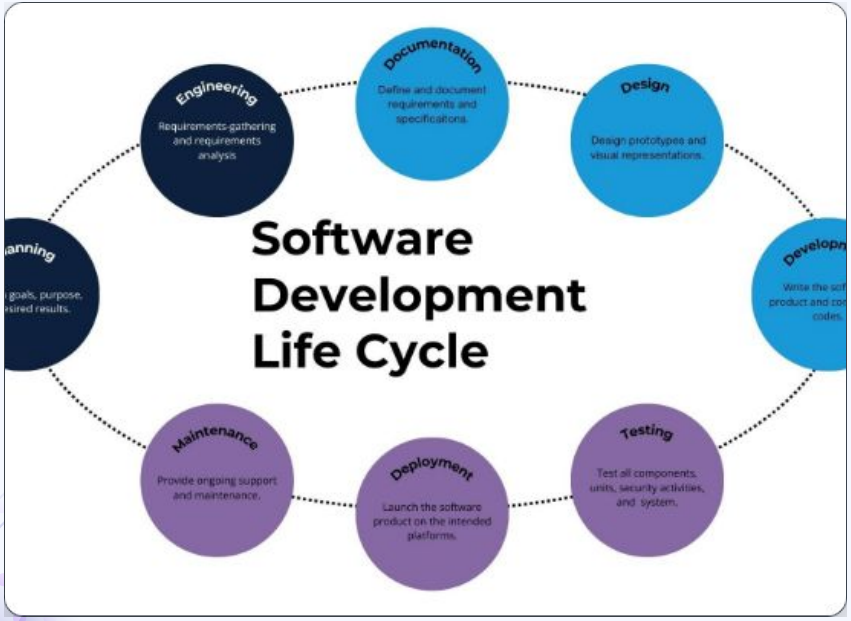
Myths

- "Automate **everything**."
- "Automation **replaces** Manual Testing."
- "Automation provides **immediate** ROI."

Reality

- Only automate stable, repetitive flows.
- It compliments manual testing.
- ROI is realized in the **long term**.

06. Strategy: When to Automate?



Crucial Phases

Automation is most effective when the application is stable.

- ✓ **Regression Testing:** Ensuring new code doesn't break existing 7-Zip features.
- ✓ **System Testing:** Validating the complete and integrated software.
- ✓ **Acceptance Testing:** Verifying business requirements.

Best for scenarios where GUI consistency is critical.

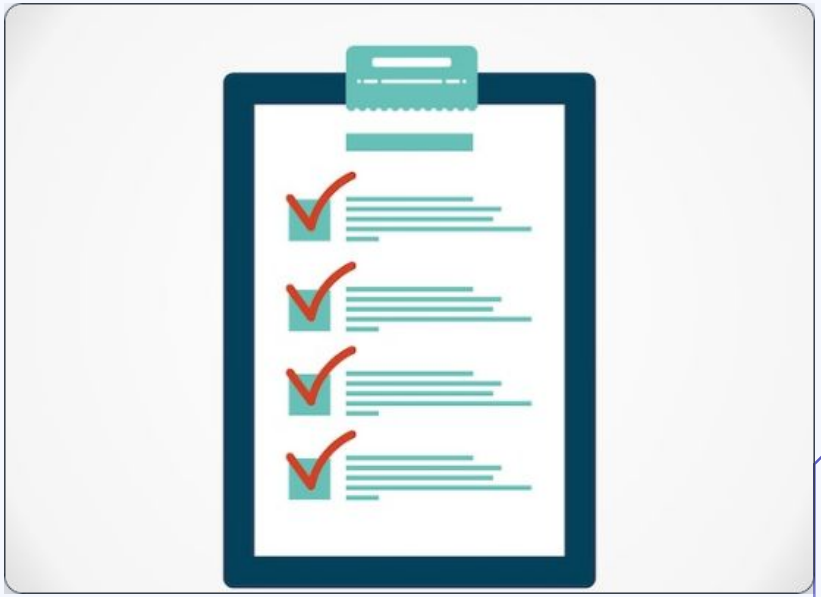
07. Strategy: What to Automate? (Where)

Target Areas

- ✓ **Functional UI Tests:** Core actions like "Add to Archive", "Extract Here".
- ✓ **Smoke Tests:** "Sanity" checks (Does the app launch? Do critical buttons work?).
- ✓ **End-to-End Workflows:** Complete user journey validations.

What to Avoid

- **Unstable UI:** Features currently in development.
- **Ad-hoc Tests:** One-time checks for edge cases.
- **Visual Aesthetics:** "Is this icon pretty?" (Leave for humans).



08. How do we do it?

1

Select Tool

pytest, pywinauto, etc.

2

Inspect UI

Identify IDs & Elements

3

Scripting

Write Action Code

4

Run Tests

Execute Framework

5

Review

Analyze Results

02

Framework & Tools

Introducing SUT, Approaches, Frameworks and Automation Tools used

SUT: 7-Zip

What is 7-Zip?

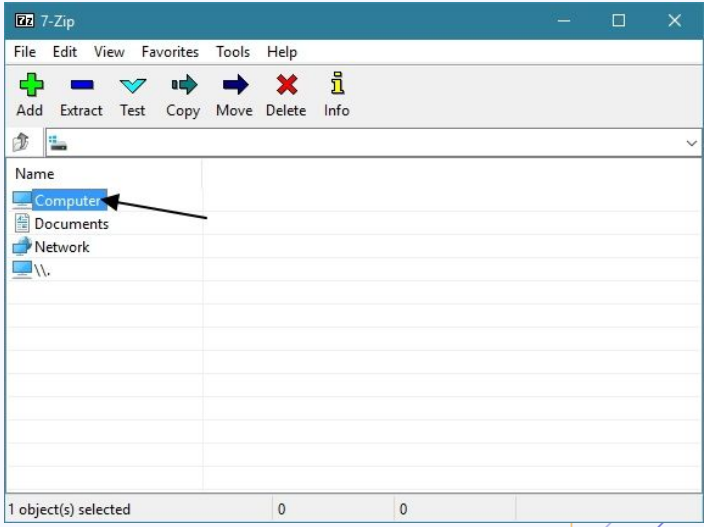
- Open-source file archiver
- Compress and Extract files, for example making .zip or .7z archives.

Why Choose 7-Zip as SUT?

- Stable UI → ideal for GUI automation
- Handles file operations (add, extract, list), easy to validate
- Offers **GUI (Windows)** + **CLI (macOS/Linux),** enabling multiple test types

GUI Automation Path (Windows)

We automate **7-Zip File Manager** on Windows.



Key tools:

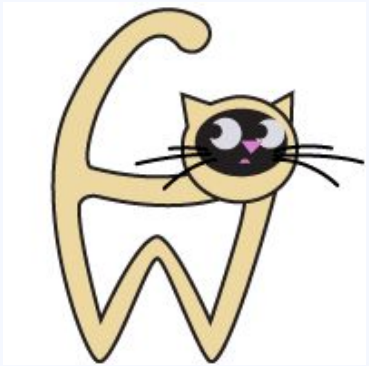
- **Pywinauto** – controls the 7-Zip window
  (click buttons, type text, choose options, open dialogs)
- **pytest** – runs the test functions and checks the results

Typical actions:

- Open 7-Zip File Manager
- Add files → create archive
- Extract archive → verify result

Automation Tool: Pywinauto

- Python library for Windows GUI applications
- Allows direct interaction with native Windows controls—such as buttons, menus, and text boxes
- Uses Win32 API or UI Automation backend.



Why choosing Pywinauto

- **Fast and simple setup:** installable with one command (pip install pywinauto) without external drivers.
- **Full Windows compatibility:** works seamlessly on Win32, WPF, and WinForms apps.
- **Readable API:** supports intuitive commands like .Edit.type_keys() and .MenuSelect().

Tools Comparison

Aspect	Pywinauto	WinAppDriver	AutoIt
License	MIT (open-source)	Open-source (Microsoft)	Freeware (closed)
Scripting Language	**Python**	C#, Java, etc. (WebDriver style)	AutoIt Basic (custom language)
Integration with pytest	**Native & direct (pure Python)**	Needs extra bindings + wrapper	Needs bridge (run AutoIt scripts from tests)

Tools Comparison

Aspect	Pywinauto	WinAppDriver	AutoIt
Best Use Case	**Automating classic Windows apps from Python test frameworks**	Enterprise .NET apps, large teams using WebDriver ecosystem	Small Windows-only scripts, admin tools
Learning Curve	**Easy** for anyone who knows basic Python	Steeper (WebDriver concepts, more setup/layers)	Need to learn a new proprietary language
Chosen?	**Yes**	No	No

Testing Framework: pytest

- Python testing framework used to execute and manage test cases.

- It automatically discovers functions whose names start with test_ in files test_*.py and reports results.

Why choosing pytest

- Clean and minimal syntax; easy to integrate with Pywinauto scripts.
- Generates readable command-line output and supports **HTML reporting** via plugins (pytest-html).
- Supports **core functionalities** such as **assertions**, **fixtures**, and **parameterization**

CLI Automation Path (macOS)

For platforms without 7-Zip GUI (like **macOS**), we automate using the **7z command-line**.

Main idea:

- Use `subprocess.run()` to execute commands like:
  - `7z a archive.7z file.txt`
  - `7z x archive.7z -pPASSWORD`

pytest is still the **runner**, but instead of Pywinauto, tests call the CLI, then check the return code and output.

CLI Libraries & pytest Integration

subprocess

- Runs `7z` commands and captures output / return code

openpyxl

- Reads multiple test data from Excel file where each line is 1 test case

zipfile and **py7zr**

- Programmatically open `.zip` and `.7z` archives
- Used to validate that archives created by 7-Zip are actually valid or not

CLI Libraries & pytest Integration

pytest integration

- All CLI tests are normal **pytest test functions** (with `assert`).

- **Fixture** (`setup_test`) → create temp folder + sample file before each test.

- **parametrize** (`@pytest.mark.parametrize`) → run the same test for every row in the Excel test cases data.

CLI Test Types Implemented

Data-Driven CLI Tests

- **Source**: Excel files (`openpyxl`).
- **Framework**: `pytest.mark.parametrize`.
- **Benefit**: Decoupling Test Data from Test Logic.

test_data.xlsx

	A	B	C	D	E	F
1	TestCaseID	ArchiveFormat	CompressionLevel	Password	InvalidName	Expected
2	1	zip	5	123		success
3	2	7z	9			success
4	3	zip	1	abc		success
5	4	zip	5	123	/bad.zip	invalid_filename
6	5	zip	5	123	/aux.zip	invalid_filename
7	6	zip	5			success
8	7	zip	9	999		success
9	8	zip	5	123		success

Happy Path / Functional CLI

- **Library**: `subprocess` module.
- **Dynamic Arguments**: Automatically handles flags (`-mx`, `-p`, `-tzip`).
- **Success Criteria**: `returncode == 0`.

```
CASE  1 | Format: ZIP  | Level: 5          | Password: 1
================================================================
EXECUTING → 7z a -mx=5 -tzip -p123 archive_1.zip sampl
```

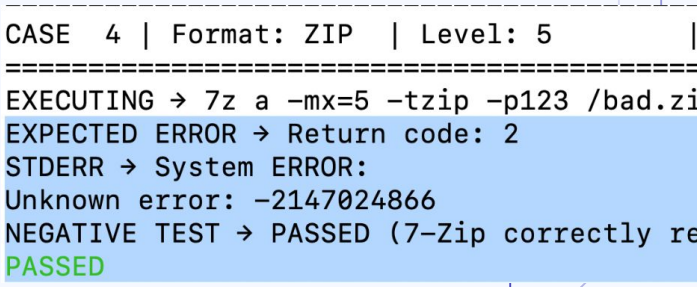
CLI Test Types Implemented

Validation Tests

- **Advanced Verification**: Beyond simple `os.path.exists`.
- **Cross-check**: Utilizing 3rd-party libs (`zipfile`, `py7zr`) to extract and verify.
- **Goal**: Ensure Data Integrity (Not corrupted).

Negative CLI Tests

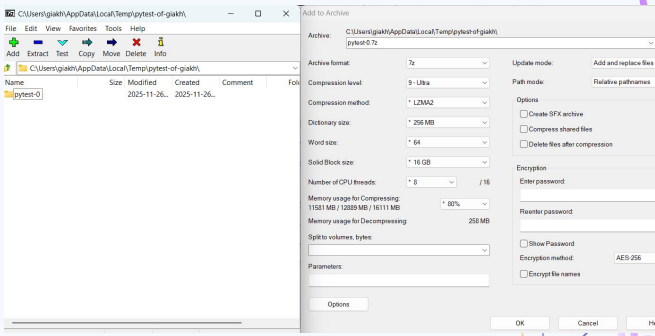
- **Error Capture**: Capturing `stderr` stream.
- **Validation**: Asserting `returncode != 0`.
- **Keyword Matching**: Scanning for "error" or "permission" in logs.



GUI Test Types Implemented

Data-Driven GUI Tests

- **Reusability**: Reusing `test_data.xlsx` from CLI.
- **Consistency**: Ensures consistent testing across interfaces.



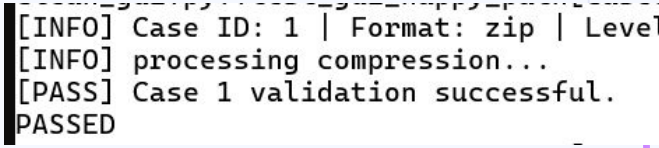
Happy Path / Functional GUI

- **Library**: `pywinauto` (UIA Backend).
- **Synchronization**: Smart `wait('visible')` instead of hard `sleep`.
- **Workflow**: Simulates user actions (Select File -> Add to Archive -> OK).

GUI Test Types Implemented

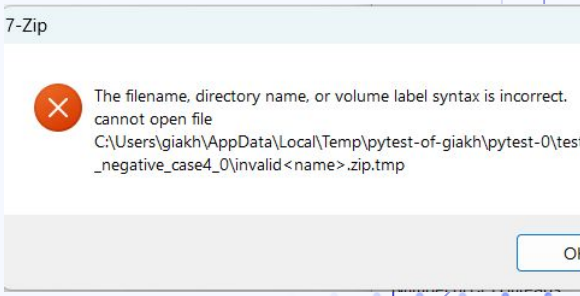
Validation Tests

- **Hybrid Approach**: UI execution with Backend verification.
- **Check**: Verify output file existence and integrity after UI processing.



Negative GUI Tests

- **Popup Detection**: Auto-detecting Error Windows (System & App level).
- **Assertion**: Verifying error message content against expected results.
- **Self-Healing**: Auto-closing popups to clean the environment.

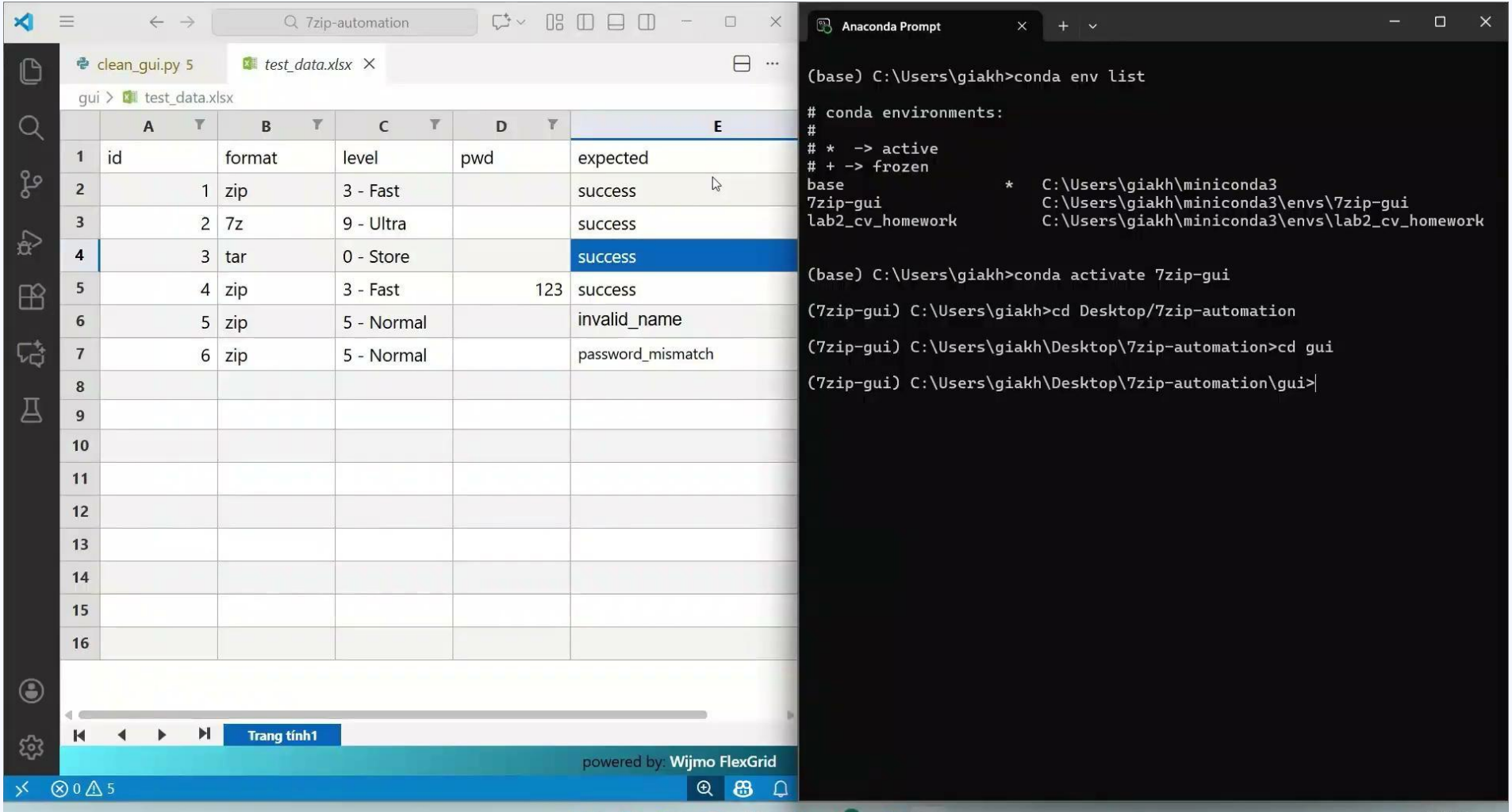


03

Demo

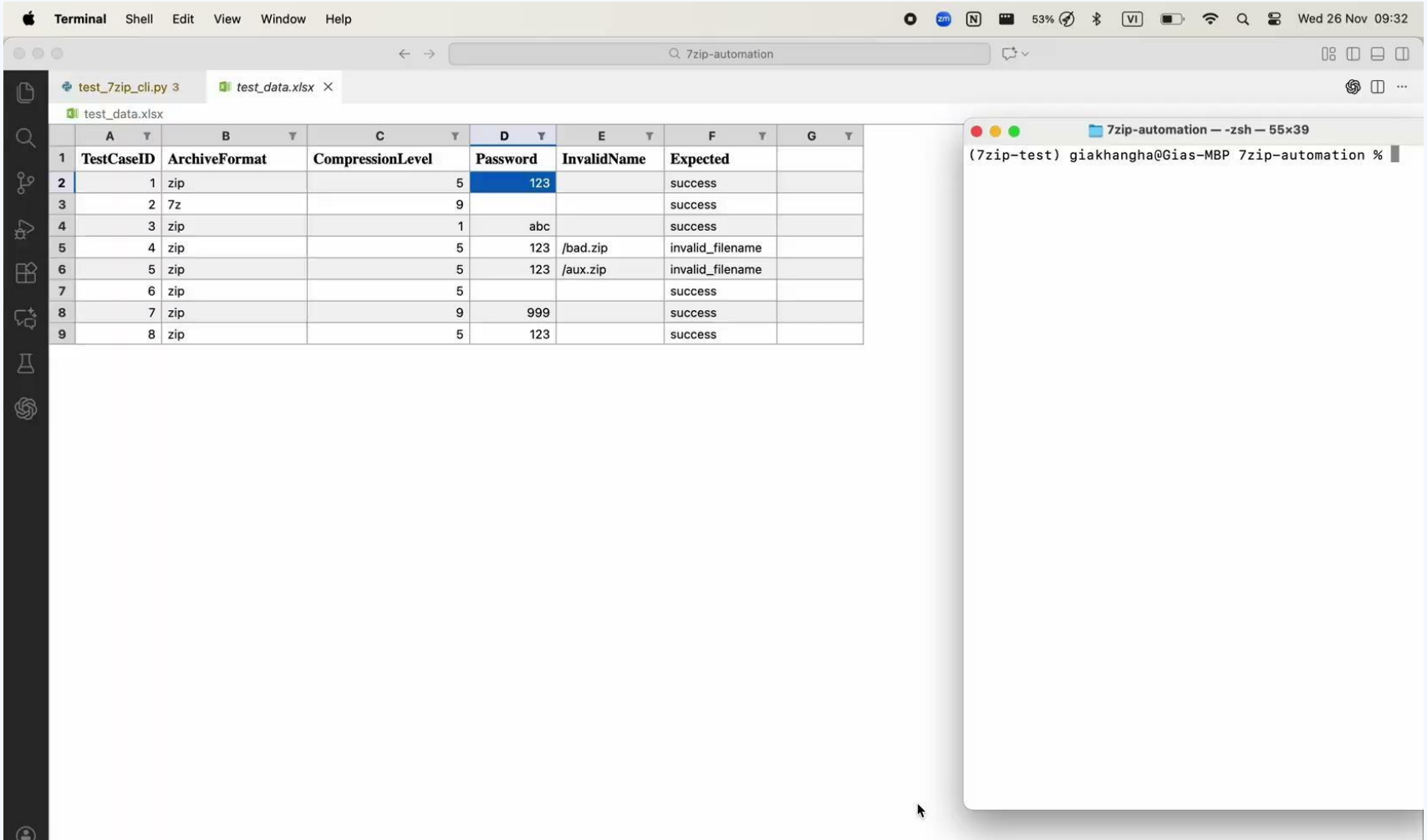
Demo with App 7-Zip on Windows & Mac Os

GUI



	A	B	C	D	E
1	id	format	level	pwd	expected
2	1	zip	3 - Fast		success
3	2	7z	9 - Ultra		success
4	3	tar	0 - Store		success
5	4	zip	3 - Fast	123	success
6	5	zip	5 - Normal		invalid_name
7	6	zip	5 - Normal		password_mismatch
8					
9					
10					
11					
12					
13					
14					
15					
16					

CLI



	A	B	C	D	E	F	G
1	TestCaseID	ArchiveFormat	CompressionLevel	Password	InvalidName	Expected	
2	1	zip	5	123		success	
3	2	7z	9			success	
4	3	zip	1	abc		success	
5	4	zip	5	123	/bad.zip	invalid_filename	
6	5	zip	5	123	/aux.zip	invalid_filename	
7	6	zip	5			success	
8	7	zip	9	999		success	
9	8	zip	5	123		success	

Thanks!

Do you have any questions?