



# Building a GUI for Unix Commands Execution on MacOS

CSCI 6917: Guided Research  
Sravya Kuchipudi  
08 August 2023

---

THE GEORGE  
WASHINGTON  
UNIVERSITY

---

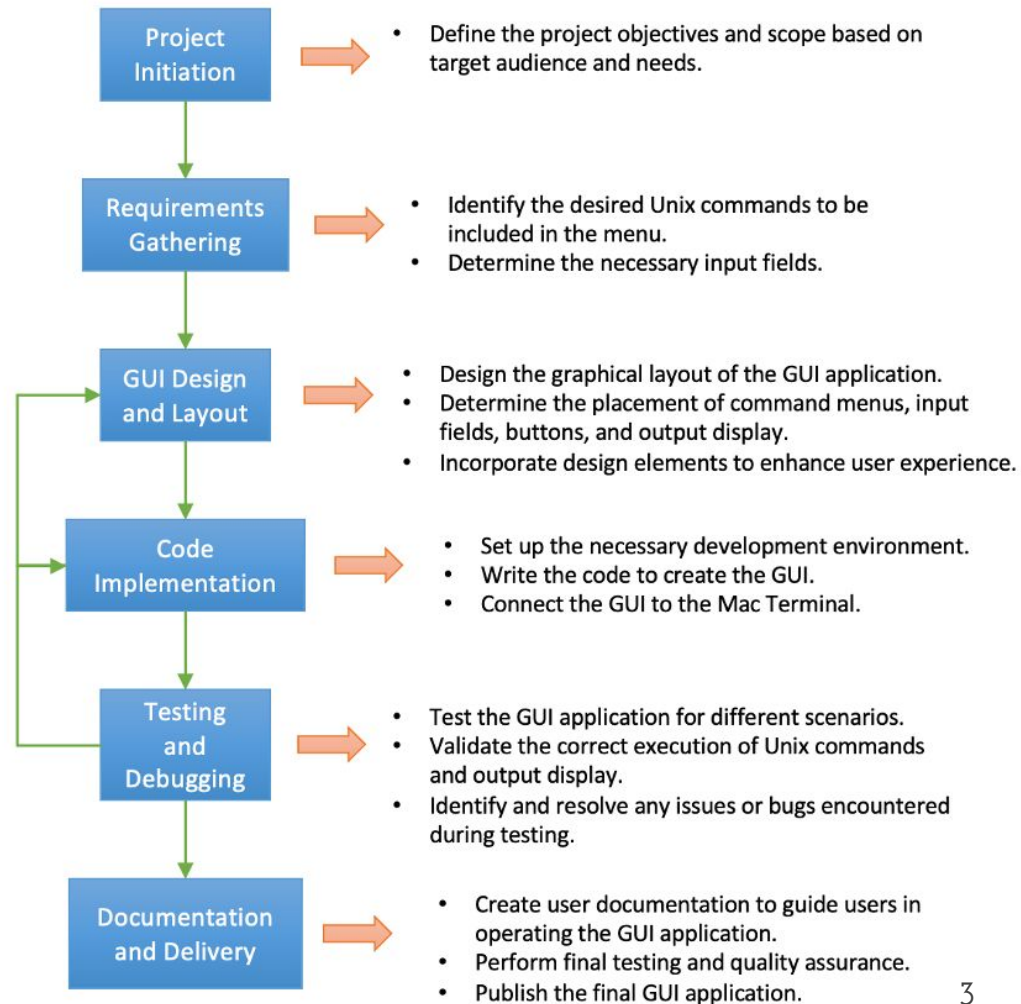
WASHINGTON, DC

# Project Objective

## Heilmeier Questions

- Objective (What was I going to do?):
  - Develop a user-friendly graphical user interface (GUI) for executing Unix commands on MacOS.
- How is it done today? Current Limitations?
  - Users have to work directly into the Terminal which requires a pre-existing knowledge of Unix Command syntax and comfort with the Terminal.
- What was my idea to do something better? Who will benefit from my work?
  - Empowers novice users by simplifying the execution of Unix commands, by making it easier to explore and use Unix commands through an intuitive GUI that has instructions, tips, and an aesthetic user interface which overall saves time and effort.
- What risks do I anticipate?
  - Security vulnerabilities in command execution and insufficient error handling.
    - These were addressed through methods such as input validation and specified error alert messages.
- Out of pocket costs? Complete within timeline?
  - No real cost due to existing access to all necessary tools and a reasonable plan was formulated to stay on track.

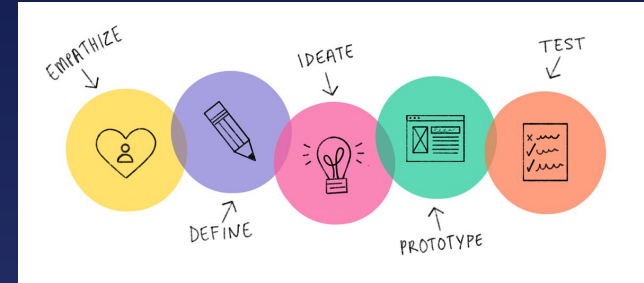
# Technical Approach Project Flow



# Technical Approach

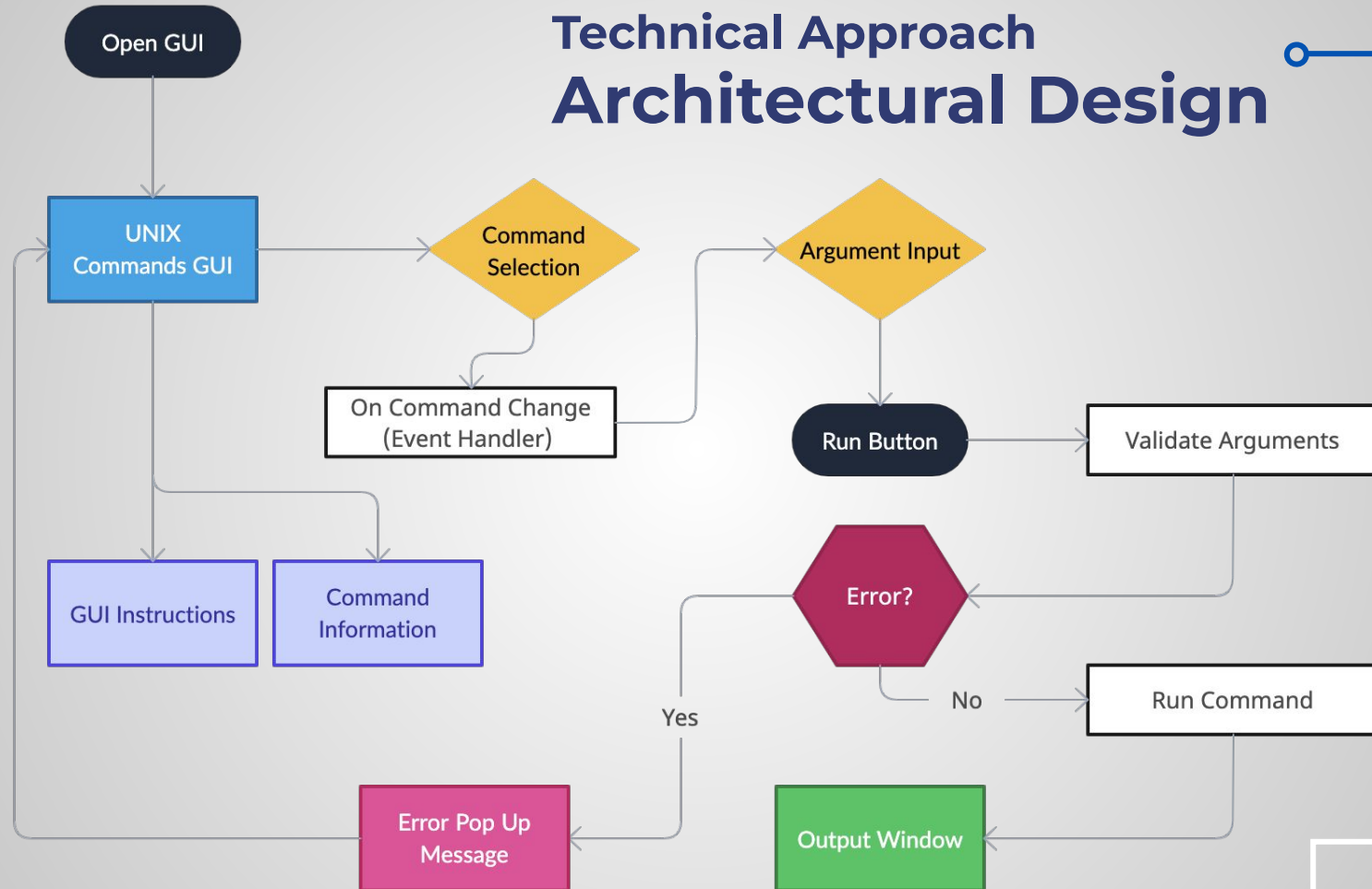
## Key Steps in Research

- Qualitative Approach
  - To focus on how to facilitate the user's needs and experience and improve through repeated testing.
- Explore GUI frameworks for MacOS.
  - PySimpleGUI and Tkinter.
- Choose relevant UNIX commands.
  - Research the most used and necessary for Terminal use.
- Design GUI layout and structure.
  - Format based on the needs of the commands.
- Implement menu-based command selection and argument input.
  - Develop command execution functionality.



# Technical Approach

## Architectural Design



# Technical Approach

## Code

```
commands = {
    'ls': {
        'description': 'List files and directories',
        'template': 'ls',
        'requires_arguments': False
    },
    'cd': {
        'description': 'Change directory',
        'template': 'cd <directory>',
        'requires_arguments': True
    },
    'mkdir': {
        'description': 'Create directory',
        'template': 'mkdir <directory>',
        'requires_arguments': True
    }
}
```

```
def validate_arguments(command, arguments):
    if not command:
        return False, "Please select a command."

    if command not in commands:
        return False, "Invalid command selected."

    if commands[command]['requires_arguments'] and not arguments:
        return False, "This command requires arguments. Please provide them."

    if not commands[command]['requires_arguments'] and arguments:
        return False, "This command does not require arguments. Please remove them."

    return True, ""
```

# Technical Approach

## Code

```
content_frame = ttk.Frame(root)
content_frame.pack(padx=20, pady=20)

command_label = ttk.Label(content_frame, text='Command:')
command_label.grid(row=0, column=0, sticky=tk.W)

command_combo = ttk.Combobox(content_frame, values=list(commands.keys()), state='readonly', width=30)
command_combo.grid(row=0, column=1, padx=5, pady=5)
command_combo.bind('<<ComboboxSelected>>', on_command_change)

arguments_label = ttk.Label(content_frame, text='Arguments:')
arguments_label.grid(row=1, column=0, sticky=tk.W)

arguments_entry = ttk.Entry(content_frame, width=30)
arguments_entry.grid(row=1, column=1, padx=5, pady=5)

run_button = ttk.Button(content_frame, text='Run', command=run_button_click)
run_button.grid(row=2, column=0, columnspan=2, padx=5, pady=10)
```

```
tooltip_toplevel = tk.Toplevel(root)
tooltip_toplevel.withdraw()
tooltip_toplevel.overrideredirect(True)
tooltip_toplevel.attributes('-topmost', True)

tooltip_label = ttk.Label(tooltip_toplevel, background="#ffffe0", borderwidth=0, relief=tk.SOLID, padding=(10, 5), wraplength=150)
tooltip_label.pack()
```

# Technical Approach

## Innovative Approach

- User-Friendly Experience using a Graphical User Interface (GUI):
  - The GUI interface makes Unix commands accessible and intuitive for users unfamiliar with the command-line interface.
- Menu-Based Command Selection:
  - Simplifying command execution by allowing users to choose commands from dropdown menus.
- Argument Prompting:
  - Guiding users to input required arguments for each command, reducing errors and improving command context.
- Robust Error Handling:
  - Providing informative and actionable feedback for users when errors occur during command execution.
- Security Considerations:
  - Emphasizing safety by warning users about potentially harmful commands (Ex: 'rm')



# Results Example

**Command Information**

**How to Use the Commands:**

**ls**  
Description: List files and directories  
Template: ls

**cd**  
Description: Change directory  
Template: cd <directory>

**mkdir**  
Description: Create directory  
Template: mkdir <directory>

**cp**  
Description: Copy file or directory  
Template: cp <source> <destination>

**mv**  
Description: Move file or directory  
Template: mv <source> <destination>

**rm**  
Description: Remove file or directory  
Template: rm <file>

**pwd**  
Description: Print working directory  
Template: pwd

**clear**  
Description: Clear screen  
Template: clear

**cat**  
Description: Concatenate and display files  
Template: cat <file>

**more**  
Description: View file content one screen at a time  
Template: more <file>

**Unix Commands GUI**

Command: **rm**

Arguments:

Run

Remove file or directory  
Template: rm <file>

To get more information about usage, click the buttons below.

GUI Instructions  
Command Info

**GUI Instructions**

**How to Use the Unix Commands GUI:**

Welcome to the Unix Commands GUI!

1. Select a command from the drop-down list.
2. If the selected command requires arguments, enter them in the 'Arguments' field.
3. Click the 'Run' button to execute the command.
4. The output will be displayed in a separate window.
5. To get information about each command, click the 'Command Info' button.

**Caution:**

- Some commands can make permanent changes to your file system.
- Be careful while using commands like 'rm', 'mv', 'mkdir', etc.
- Ensure that you have a backup or are certain of the command's effect.

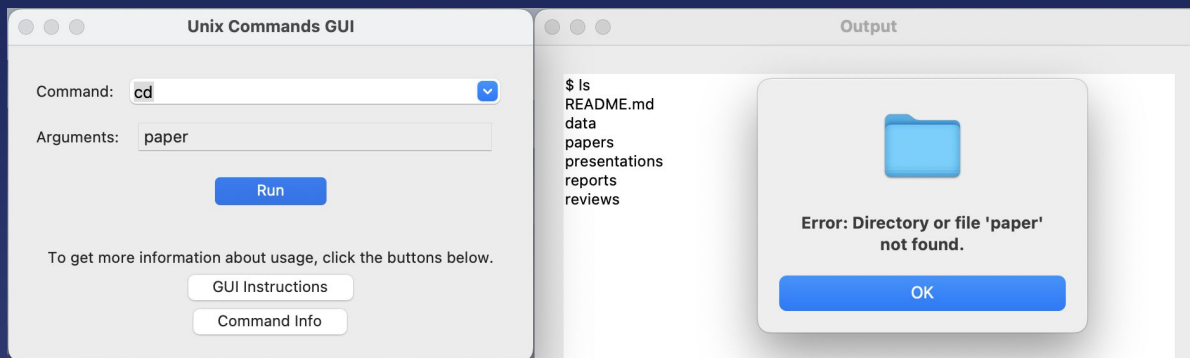
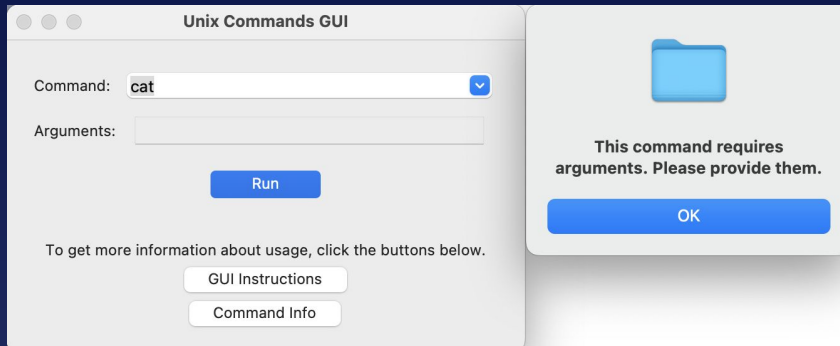
**Command Information:**

- Click the 'Command Info' button to view a list of available commands and their descriptions.

**Output**

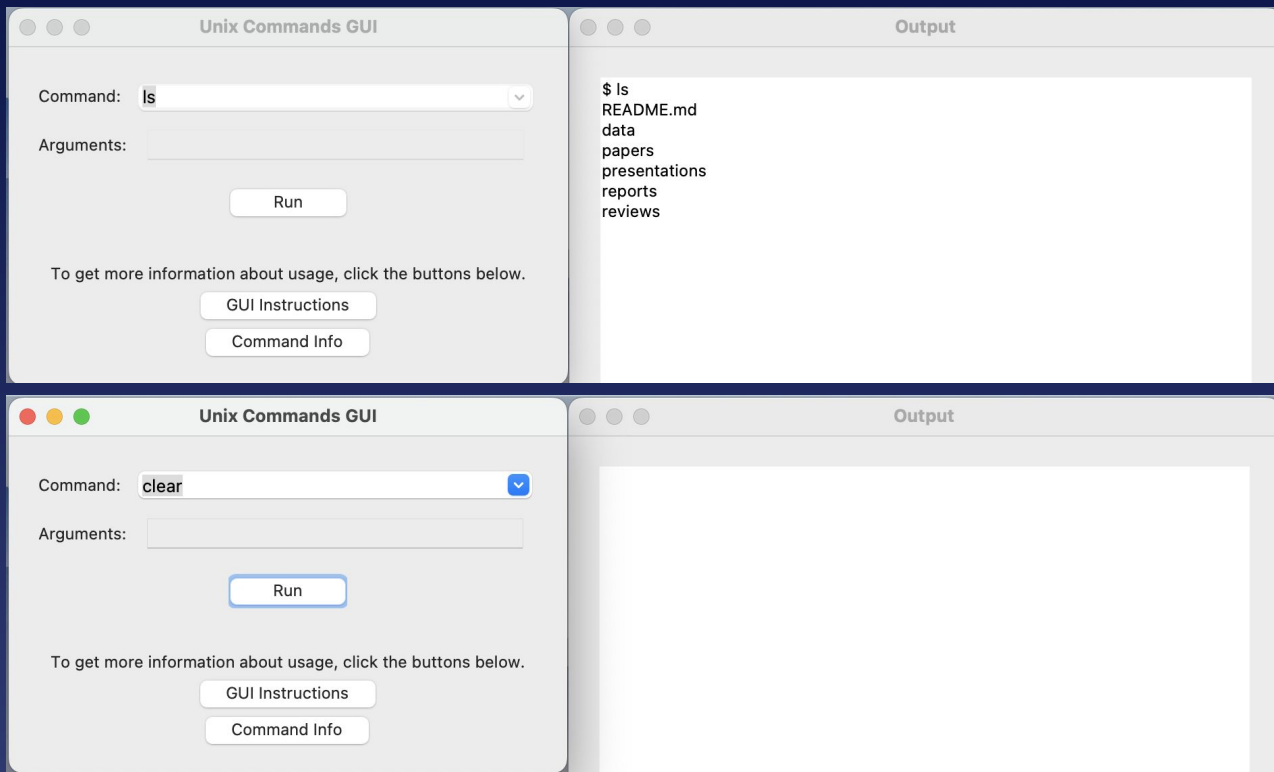
```
$ pwd
/Users/sravaya/Documents/GWU/GR6917/guidedresearchproject-sravayak13
$ ls
GR6917 Slides Draft.pdf
README.md
data
papers
presentations
reports
reviews
slides
$ cd papers
$ ls
1. Unix commands — Unix Guide documentation.pdf
GUIBasedApplicationforPDFProcessingToolsUsingPython&CustomTkinter.pdf
Python-GUI-Programming-with-Tkinter-Design-and-build-functional-and-user-frien
dly-GUI-applications-2nd-Edition-by-Alan-D.-Moore-preview.pdf
USING TKINTER OF PYTHON TO CREATE GRAPHICAL USER INTERFACE (GUI) FOR
SCRIPTS IN LNLS.pdf
articles
paperstest.txt
textbooks
```

# Results Example



# Results Example

```
def clear_output_window():  
    output_text.configure(state='normal')  
    output_text.delete(1.0, tk.END)  
    output_text.configure(state='disabled')
```



# Conclusion

- The development of a graphical user interface (GUI) for executing Unix commands on MacOS offers a user-friendly alternative to the traditional command prompt in multiple ways.
  - Menu-based command selection and argument prompting:
    - Simplifies the process of running Unix commands, making it accessible to novice users.
  - Error handling mechanisms and informative feedback:
    - Enhances the GUI's reliability and helps users understand and rectify potential errors.
  - Bridges the gap between technical complexity and user accessibility.\*\*
- Overall, this project has been successful in its mission to make Unix commands more user-friendly and efficient on MacOS.
  - There was some difficulty along the way with GUI development and integration with the Terminal, as well as with defining certain aspects of the research.
    - However, this was addressed by taking more time for research and discussing with the Professors as needed.

# Future Work

- Interactive Tutorial
- Expand Command Support
- Tracking Features
  - Save Output to File
  - Error Logging and Reporting
  - Command History
- Syntax Highlighting
  - Make different aspects more easily identifiable
- Macro Commands
  - Ability to bundle several commands together

UNIX<sup>®</sup>  
An Open Group Standard





# Thank you!

Questions?

