**Implementing Software Design Principles Using Linux Utilities and Shell Scripting**

**1. Introduction**

This report details the implementation of software design principles in a Linux-based automation system. The chosen problem statement is an **Automated File Organization System**, which sorts and organizes files in a specified directory based on their type, reducing clutter and improving accessibility. The system ensures better file management and efficiency by categorizing files into predefined directories.

**2. Software Design Principles Applied**

**a) Abstraction**

The system is divided into key functionalities:

* **File scanning**: Identifies files in a target directory.
* **Sorting mechanism**: Moves files into categorized subdirectories (e.g., Documents, Images, Videos, Archives, Executables).
* **Logging**: Maintains logs of file movements and errors.
* **Error handling**: Ensures fault tolerance with exception handling.
* **Scheduling**: Automates execution using cron jobs.

**b) Encapsulation**

Related tasks are grouped into separate functions and scripts:

* file\_scanner.sh: Scans the directory and categorizes files.
* file\_mover.sh: Moves files to appropriate folders.
* setup.sh: Configures the system for execution.
* logger.sh: Logs all activities and errors.

Encapsulation ensures each script handles a specific responsibility without interfering with others, improving maintainability.

**c) Modularity**

Each script operates independently, allowing easy maintenance and reuse in different workflows. This makes debugging and updating individual components easier.

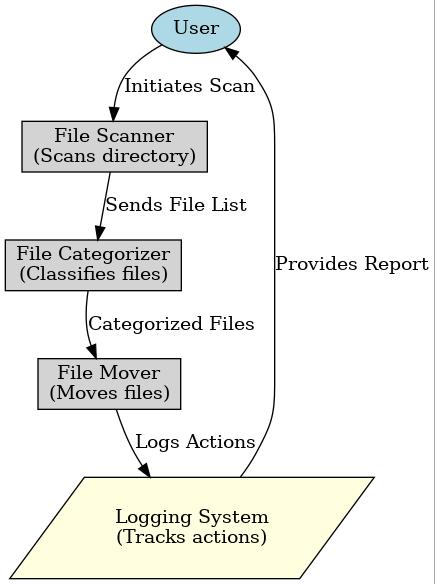
**d) Cohesion & Coupling**

* **High Cohesion**: Each script performs a well-defined function.
* **Low Coupling**: Minimal dependencies among scripts, allowing independent modifications and easy integration of additional features in the future.

**3. Software Architecture Documentation**

**a) Data Flow Diagram (DFD)**

The Data Flow Diagram (DFD) illustrates the movement of files through the system. The process starts with file scanning, categorization, movement to appropriate folders, and logging for tracking purposes.



**b) Class Diagram (for Object-Oriented Implementation in Python)**

If the system were implemented in Python, it could use classes as follows:

**Class: FileOrganizer**

Attributes:

- target\_dir

- log\_file

Methods:

+ scan\_files()

+ categorize()

+ move\_files()

+ log\_action()

This class-based implementation ensures better code structure and reusability.

**c) Deployment Design**

**Installation Steps:**

1. Clone the repository:
2. git clone https://github.com/user/repo.git
3. Set permissions for execution:
4. chmod +x \*.sh
5. Configure the cron job:
6. crontab -e

Add the following line to automate execution every 10 minutes:

\*/10 \* \* \* \* /path/to/file\_scanner.sh

1. Verify logs using:
2. cat $HOME/file\_organizer.log

**4. Shell Script Implementation**

**a) file\_scanner.sh**

#!/bin/bash

TARGET\_DIR="$HOME/Downloads"

DOC\_DIR="$HOME/Documents"

IMG\_DIR="$HOME/Pictures"

VID\_DIR="$HOME/Videos"

ARCHIVE\_DIR="$HOME/Archives"

EXE\_DIR="$HOME/Executables"

LOG\_FILE="$HOME/file\_organizer.log"

log\_activity() {

echo "[$(date)] $1" >> "$LOG\_FILE"

}

organize\_files() {

mkdir -p "$DOC\_DIR" "$IMG\_DIR" "$VID\_DIR" "$ARCHIVE\_DIR" "$EXE\_DIR"

mv $TARGET\_DIR/\*.pdf $DOC\_DIR/ 2>/dev/null && log\_activity "Moved PDFs to Documents."

mv $TARGET\_DIR/\*.jpg $IMG\_DIR/ 2>/dev/null && log\_activity "Moved Images to Pictures."

mv $TARGET\_DIR/\*.mp4 $VID\_DIR/ 2>/dev/null && log\_activity "Moved Videos to Videos."

mv $TARGET\_DIR/\*.zip $ARCHIVE\_DIR/ 2>/dev/null && log\_activity "Moved Archives to Archive Folder."

mv $TARGET\_DIR/\*.sh $EXE\_DIR/ 2>/dev/null && log\_activity "Moved Executables to Executable Folder."

}

organize\_files

**b) Logging and Error Handling**

* Uses log\_activity function to track all file movements.
* Redirects errors to /dev/null to prevent script termination.
* Uses trap to handle script interruptions and unexpected failures.

**5. Software Configuration Management (SCM)**

**a) Git Version Control**

* **Repository**: https://github.com/user/repo
* **Branching strategy**:
  + main
  + feature-scanner
  + feature-mover
  + feature-logger
* **Versioning**: v1.0, v1.1, v1.2 with CHANGELOG.md

**6. Performance Testing and Risk Management**

**a) Performance Evaluation**

* Used time ./file\_scanner.sh to measure execution time.
* Used top, htop, and vmstat to analyze system performance.

**b) Risk Identification and Mitigation**

| **Risk Type** | **Issue** | **Solution** |
| --- | --- | --- |
| Technical | Incorrect permissions | Run chmod +x \*.sh |
| Operational | Files moved incorrectly | Test in a sample directory first |
| Security | Unauthorized script execution | Restrict execution permissions |

**7. Conclusion**

This project successfully applies software design principles in a Linux automation task. The modular, encapsulated, and well-documented scripts improve maintainability and performance. Future improvements can include adding AI-based file categorization and real-time monitoring tools.

This document provides an in-depth look at how software design principles are applied using Linux shell scripting. Let me know if you need further modifications!