# Project-1 Bonus Report

1. Introduction:

This report discusses a comprehensive approach to detecting design smells in code repositories through the integration of a GitHub workflow and two Python scripts. The workflow orchestrates the execution of the scripts, which utilize OpenAI Codex for code analysis and storage of detected smells.

2. Workflow Overview:

The workflow, located in the "workflows" directory, is triggered on push events to a specific branch or manually through a workflow dispatch. It coordinates the execution of the Python scripts located in the "scripts" directory. The workflow performs the following key tasks:

- Checks out the repository and sets up the execution environment.

- Analyzes the code for design smells using the first Python script (design_smell_detection.py).

- Fixes Code Smells for design smells using the second Python Scripy (fix_code_smells.py)

- Creates a pull request with the detected code smells and requests a review from a randomly selected reviewer. ( using create_pr.py)

3. First Script: Design Smell Detection

The first script, design_smell_detection.py, is responsible for detecting design smells in code files. It performs the following actions:

- Reads the GitHub token from a secured file.

- Creates a new branch dynamically and sets up the JDK and Python environment.

- Analyzes the code for design smells using OpenAI Codex ( gpt4-turbo-preview engine)

- Commits and pushes the detected smells to the repository.

- Creates a pull request to merge the changes into the base branch.

4. Second Script: Storing Smells and API Key Loading

The second script comprises several functions to facilitate the storage of detected smells and loading of the OpenAI API key. The functions include:

- store_smells(file_path, smells): Stores or updates detected smells for a file in a JSON file.

- load_api_key(file_path): Loads the OpenAI API key from a specified file.

- analyze_file_for_smells(file_path, api_key): Analyzes a given Java file for design smells using OpenAI Codex.

- find_java_files(directories): Recursively finds all .java files in specified directories.

5. Fix Code Smells Step:

- *Name*: Fix Code Smells

- *Description*: This step executes a Python script named fix_code_smells.py located in the .github/scripts/ directory. The script is responsible for analyzing the detected code smells and applying appropriate fixes to the source code.

- *Environmental Variables*:

  - OPENAI_API_KEY_PATH: Specifies the path to the OpenAI API key file (OpenAi_secret.txt) necessary for performing code smell fixes. This key may be utilized for accessing AI-based code analysis tools or other relevant services.

- *Script Functionality*:

  - The script is expected to process the detected code smells and automatically apply corrective actions to the affected code segments.

  - It should output either the files it modifies or a status indicating the success of the fixing process.

6. Commit Fixed Code Step:

- *Name*: Commit Fixed Code

- *Description*: This step commits the changes made by the code smell fixing script to the codebase.

- *Run Command*:

  - The script begins by configuring the global Git user email and name to ensure proper attribution of the commits made during the workflow execution.

  - It stages all changes in the directory using the git add . command. Optionally, specific paths can be specified to limit the changes staged for commit.

  - A commit is performed with a descriptive message indicating that fixes for detected code smells are being applied.

  - Finally, the changes are pushed to the origin repository under the newly created branch ${{ env.NEW_BRANCH }}.

- *Note*:

  - This step ensures that the fixes applied to resolve code smells are incorporated into the version-controlled codebase, facilitating further review and integration into the project.

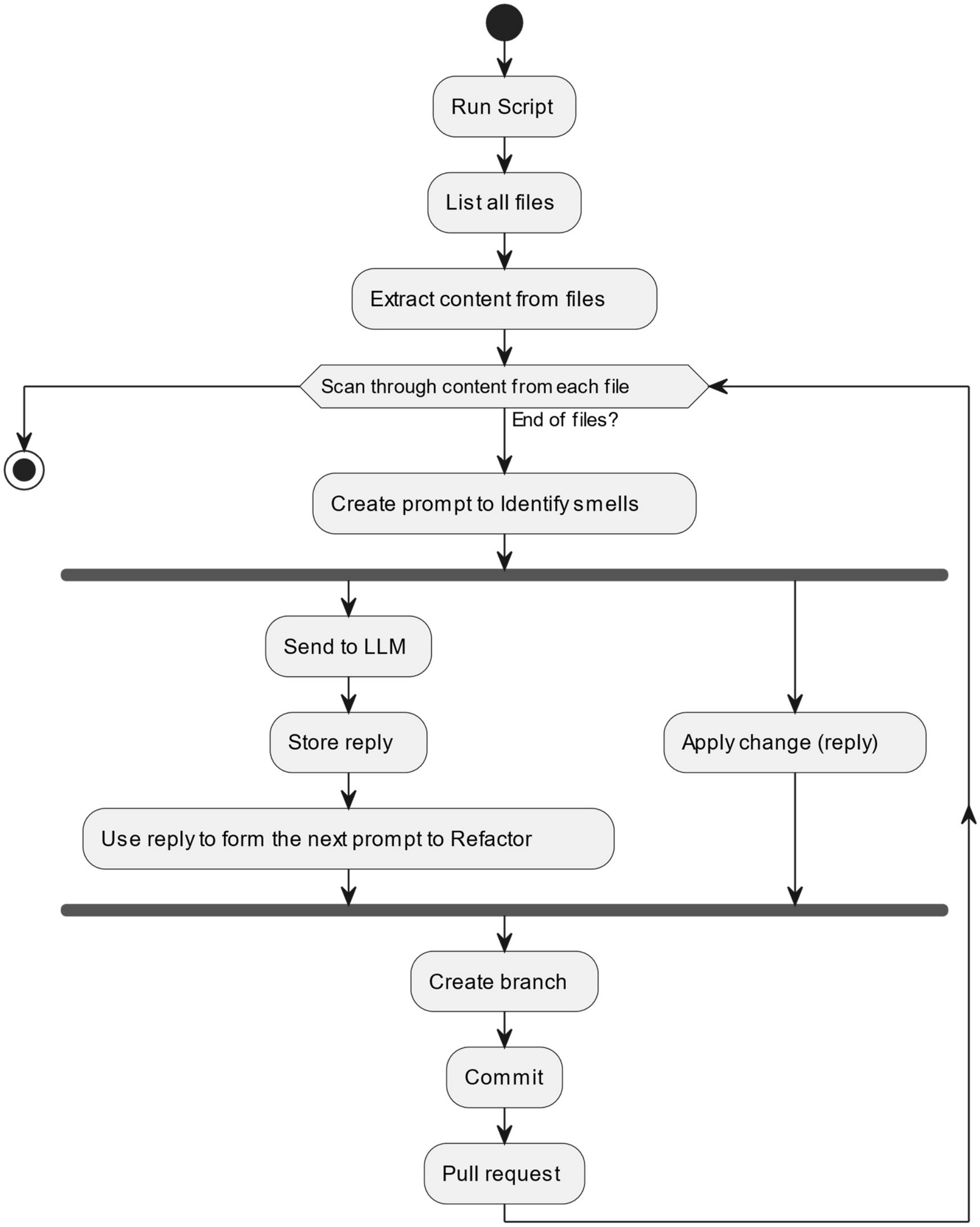
7. Execution Flow:

- The GitHub workflow triggers on push events and sets up the execution environment.

- The design_smell_detection.py script analyzes the code for design smells and creates a pull request.

- The second script loads the API key, analyzes Java files for design smells using Codex, and stores the detected smells.

**Process Diagram**


Run Script
List all files
Extract content from files
Scan through content from each file
End of files?
Create prompt to Identify smells
Send to LLM
Store reply
Use reply to form the next prompt to Refactor
Apply change (reply)
Create branch
Commit
Pull request

8. Conclusion:

The integration of the workflow and Python scripts provides an automated solution for detecting design smells in code repositories. By leveraging OpenAI Codex and GitHub Actions, developers can proactively identify and address potential issues, improving code quality and maintainability.

9. Recommendations for Improvement:

- Implement error handling mechanisms in the scripts to ensure robustness and graceful handling of exceptions.

- Enhance the detection algorithms to identify a broader range of design smells and provide more detailed analysis.

- Explore the possibility of incorporating feedback mechanisms to refine the detection process based on user input and feedback.

- Continuously monitor and optimize the workflow and scripts to adapt to evolving codebases and development practices.

**IN SUMMARY**

- **On Push to Master**
- **Create a Temporary Branch**
- **Read code of listed files**
- **Detect Smells using llm and store**
- **Commit**
- **Push**
- **Fix using LLM and store**
- **Commit**
- **Push**
- **Create Pull Request**