CHAPTER 3

ABSTRACT

Software complexity is a critical aspect of software development that directly impacts maintainability, reliability, and overall software quality. As software systems evolve, their complexity tends to increase, leading to potential challenges in understanding, debugging, and enhancing the codebase. This paper presents a comprehensive approach to code complexity monitoring, aimed at providing developers and project managers with valuable insights into the evolving complexity of their software systems.

The proposed methodology integrates static code analysis techniques to measure and monitor various dimensions of code complexity. Static analysis examines the source code without executing it, focusing on metrics such as cyclomatic complexity, code duplication, and class cohesion.

The code complexity monitoring system is designed to be easily integrated into existing continuous integration pipelines, enabling developers to receive real-time feedback on complexity changes introduced by their code commits.

CHAPTER 4

INTRODUCTION OF THE PROJECT

In the dynamic landscape of software development, managing and understanding code complexity is a crucial factor that significantly influences the success and sustainability of a software project. As software systems evolve and expand, their source code tends to grow in size and intricacy, making it challenging for developers to maintain, debug, and enhance the software efficiently. Code complexity, encompassing various dimensions such as structural intricacy, dependencies, and algorithmic intricacies, poses a constant concern for software maintenance.

Static code complexity involves characteristics inherent in the source code itself, including the organization of classes, methods, and the overall architecture.

To address these challenges, code complexity monitoring emerges as a proactive and strategic approach in software development. This involves the continuous assessment and analysis of code complexity metrics to provide developers and project managers with insights into the evolving intricacies of their software.

CHAPTER 5

MODULE DESCRIPTION

- Class Overview: The function names and levels of complexity are stored in an empty list when the `Calculator` class is first created.

- Get complexity level: It takes overall complexity as input and returns a complexity level based on certain conditions.

- Calculate complexity: It takes file path as input, reads the code and calculates the overall complexity and its level based on the functions.

CHAPTER 6

ALGORITHM

Step 1: Start

Step 2: Instantiate an object of the Calculator class.

Step 3: Main Menu Loop:

Enter a loop that continues indefinitely (while True).

Step 4: Print Menu Options:

Display the available options to the user.

Step 5: User Input:

If the choice is '1':

Take user input for the file name.

Create a new file and display a waiting message.

If the choice is '2':

Take user input for the file name to read.

Print the content of the file.

Calculate and display the complexity of functions.

If the choice is '3':

Take user input for the file path to delete.

Try to delete the file; handle exceptions if the file is not found or other errors occur.

If the choice is '4':

Print an exit message.

If the choice is invalid:

Print an error message.

Step 6: End

CHAPTER 7

RESULT

```
Welcome

Menu:
1. Create File
2. Read File
3. Delete File
4. Exit
Enter your choice: 1
Enter the file name (including extension): Sam.py
 Please Wait
Your File has been created. Please open the file and enter the code..

Menu:
1. Create File
2. Read File
3. Delete File
4. Exit

Enter your choice:
```

```
Menu:
1. Create File
2. Read File
3. Delete File
4. Exit
Enter your choice: 2
enter the file name sample.py
from radon.complexity import cc_visit
from pathlib import Path
import time
import os,sys
import warnings
warnings.filterwarnings('ignore')

class Calculator:
    def __init__(self):
```

```
######...Complexity of above File...######
Function                  Complexity
-------------------------------------
Calculator                4 - Easy
__init__                  1 - Easy
get_complexity_level      1 - Easy
calculate_complexity      7 - Moderate

Overall Complexity Level: Easy
Execution Time: 0.007001638412475586 seconds

Functions and Complexities:
[{'Name': 'Calculator', 'Complexity': 4}, {'Name': '__init__', 'Complexity': 1}, {'Name': 'get_complexity_level', 'Complexit
y': 1}, {'Name': 'calculate_complexity', 'Complexity': 7}]
```

```
Menu:
1. Create File
2. Read File
3. Delete File
4. Exit
Enter your choice: 3
Enter the file path you want to delete: sample.py
Your file is being deleted....
File 'sample.py' deleted successfully.

Menu:
1. Create File
2. Read File
3. Delete File
4. Exit
Enter your choice: 4
Exiting the program. Goodbye!
```

CHAPTER 8

CONCLUSION

In conclusion, code complexity monitoring emerges as a pivotal strategy in contemporary software development, offering a proactive means to address the challenges associated with the evolving intricacies of source code. As software continues to play a critical role in diverse industries, the importance of code complexity monitoring cannot be overstated. It is a strategic investment in the long-term viability of software projects, aligning development efforts with the goal of creating maintainable, scalable, and high-quality software.

CHAPTER 9

REFERENCE

1.https://www.researchgate.net/publication/333228587_A_Code_Complexity_Model_of_Object_Oriented_Programming_OOP

2. https://www.devopsschool.com/blog/code-complexity-tools/