**IN16/00060/21**

**ODHIAMBO FRANCIS ONGERE**

**SOFTWARE METRICS**

**CAT**

1. **(10 marks)** Discuss the significance and interpretation of the following software metrics in the context of this project: a. **Lines of Code (LOC)** b**. Cyclomatic Complexity (CC)** c. **Defect Density**

a. **Lines of Code (LOC)**

Significance: Lines of Code (LOC) is one of the most widely used software metrics. It measures the size of a software system by counting the number of lines of source code written, excluding comments and blank lines. LOC provides an overview of the software's size and can be an indicator of project complexity and development effort.

Interpretation in this Project: In this case, the project has 10,000 lines of code. While LOC provides a rough measure of the system’s scale, it must be interpreted with caution:

* **Productivity Indicator**: A higher LOC count can indicate a larger, more complex system, but it does not directly translate to quality or effort. For instance, excessive LOC may reflect inefficient or redundant code, while well-structured code may require fewer lines.
* **Quality Implications**: Larger systems, as indicated by more LOC, generally require more testing and maintenance efforts, as the larger the codebase, the higher the potential for defects.

For this project, 10,000 LOC suggests a mid-sized system. The key takeaway is that the LOC count alone does not determine software quality but provides insight into its overall size and complexity.

**b. Cyclomatic Complexity (CC)**

**Significance:** Cyclomatic Complexity (CC) measures the complexity of a program’s control flow. It counts the number of linearly independent paths through the program’s source code. Higher CC indicates more complex and potentially error-prone code.

**Interpretation in this Project:** With a Cyclomatic Complexity of 500, this project exhibits relatively high complexity. This suggests:

* **Testing Requirements:** High CC indicates that more test cases are necessary to cover all potential execution paths. In this case, with a CC of 500, thorough testing is critical to ensure all paths are adequately covered.
* **Maintainability Concerns:** High CC often correlates with code that is more difficult to maintain. This may lead to challenges when adding new features or fixing bugs, as understanding the code’s flow becomes harder as complexity increases.
* **Risk of Bugs:** Complex code is more susceptible to defects because it is harder for developers to predict all the interactions within the codebase.

The high Cyclomatic Complexity suggests that the team may need to refactor the code to improve maintainability and reduce potential risks.

**c. Defect Density**

**Significance:** Defect Density measures the number of defects (bugs) found in a software system per unit of size (typically per 1,000 LOC). It is used to evaluate software quality and identify potential problem areas in the code.

**Interpretation in this Project:** The defect density can be calculated as follows: Defect Density = (Number of Defects/Lines of Codes) \*1000

Given that there are 20 defects and 10,000 LOC, the defect density is:

Defect Density = (20/10,000) \* 1000

Resulting to **2 defects per 1,000 LOC**

A defect density of 2 is relatively low, indicating that the project has a good quality control process and that the codebase is relatively bug-free. This is a positive indicator of the software’s reliability, suggesting that the team has effectively managed to maintain a low rate of defects in the system.

1. **(10 marks)** Describe how the **Defects Reported After Testing** metric can be used to evaluate the quality and reliability of the software. Discuss how the number of defects correlates with overall project success.

**Using the Defects Reported After Testing Metric:**

The **Defects Reported After Testing** metric indicates how many defects were found after the testing phase of the software development process. This metric plays a crucial role in evaluating the quality and reliability of the software.

**Interpretation and Evaluation:**

* **Software Quality:** The 20 defects reported after testing suggest that the software is of good quality, as the number of defects is relatively low for a project with 10,000 LOC. This implies that the development and testing processes were effective in identifying and addressing issues before the software was released.
* **Reliability:** A lower number of defects correlates with higher reliability. This indicates that the software is less likely to fail under normal operation, which is a positive sign for its future use in production environments.

**Correlation with Overall Project Success:**

* **Project Success and Defects:** Generally, a high number of defects is an indicator of project difficulties, including poor planning, lack of adequate testing, or poor coding practices. In contrast, fewer defects point to a well-managed project with good quality control. Since this project has a relatively low number of defects, it suggests that the team is successfully managing quality and staying on track with the project's objectives.
* **Impact on Timeliness and Costs:** A lower defect count reduces the need for extensive rework and post-release patches, which can save time and reduce costs. Therefore, fewer defects directly contribute to the project's overall success by improving timeliness and maintaining budget efficiency.

1. **(10 marks)** Discuss how **Effort Estimation** (in person-hours) can be used to measure the efficiency of the development team. What insights can be gained by analyzing the productivity of the team based on the available effort estimation?

**Using Effort Estimation to Measure Efficiency:**

Effort Estimation, in terms of person-hours, represents the total time spent by the development team on the project. The project has an estimated total of **2,500 person-hours**. This metric provides insight into the efficiency of the development process and team productivity.

**Interpretation and Analysis:**

* **Average Effort per Developer:** The total effort can be divided by the number of developers to determine how much time each developer, on average, spent on the project. For 15 developers and 2,500 person-hours, the average effort per developer is:

**(2500/15) = 167 Hours Per developer**

This implies that each developer has worked an average of 167 hours during the six-month period.

* **Efficiency Insights:** Comparing the estimated effort with the output, such as the quality of the product (measured by defect density) and the completion time, allows for an evaluation of team efficiency. A low number of defects and a reasonable effort estimate suggests that the team is working effectively, likely balancing speed and quality in the development process.

**Productivity Insights:**

* If the development team was able to produce a relatively low defect density with 2,500 person-hours, this suggests that the team is highly productive. Effective use of effort and time, combined with low defect rates, points to a well-managed development process with appropriate work distribution and task management.
* The project’s success is also enhanced by the team's ability to maintain productivity while keeping the number of defects under control.