Addis Ababa Science and Technology University

Review

College of Engineering

Department of Software Engineering

Title: “Key Software Metrics”

Course: Software Evolution and Maintenance in Software Engineering

Section: C

Name	**Id**
1. Mahlet Hailu	ETS0794/13
2. Nebyat Bekele	ETS1052/13
3. Naod Ararsa	ETS0956/13
4. Naod Zekeria	ETS0961/13
5. Naol Yadete	ETS0968/13
6. Natnael Tafesse	ETS1026/13

Submitted to:

Date: April 3, 2025

Table Of contents

Introduction

Software quality metrics play a crucial role in modern software development. These metrics provide valuable data points for assessing various facets of software quality, including reliability, performance, usability, and maintainability. This document will delve into specific metrics: Defect Density and Software Code Complexity, Code Churn, Test Coverage, Mean Time to Failure (MTTF), Mean Time to Repair (MTTR), Customer-Reported Incidents and Performance Metrics.

Key Software Metrics

1. Defect Density

Defect Density (DD) is a software quality metric that measures the number of defects in a software product relative to its size. It's typically expressed as the number of defects per unit of size, such as defects per thousand lines of code (KLOC) or defects per function point [1].

According to Rahmani and Khazanchi, several factors can influence defect density. The findings of Rahmani and Khazanchi suggest that, for open-source projects, the size of the project and the number of developers are key factors influencing defect density.

2. Code Complexity

Software code complexity is a measure of the intricacy of a software's code. It indicates how difficult it is to understand, maintain, and modify the code [2]. High code complexity can lead to increased errors, reduced reliability, and higher maintenance cost

Katipearachchi, Sandareka, and Weerasinghe [2] describe several metrics used to quantify code complexity including Cyclomatic Complexity (CC), Lines of Code (LOC), Halstead Complexity Measures, and Cognitive Complexity

High code complexity negatively impacts software quality by making code harder to understand, debug, and maintain. According to Katipearachchi, Sandareka, and Weerasinghe [2], it increases the likelihood of defects, reduces software reliability, raises maintenance costs, and lowers developer productivity. Conversely, low complexity improves code readability, reduces errors, enhances reliability, and boosts productivity. To manage code complexity, developers can use modular design, refactoring, coding standards, and design patterns. [2].

3. Code Churn

Code Churn is a software development metric that measures the amount of code that is added, modified, and deleted within a given period. It is often used to assess code stability, developer productivity, and potential technical debt. Code churn is typically expressed as a percentage of changed code relative to the total codebase size [3].

According to Jos Kraaijeveld, high code churn can indicate instability in the software, frequent rework, or issues in requirement clarity. Organizations can use code churn metrics to monitor project health, optimize development processes, and identify potential risks in software maintenance [3].

4. Test Coverage

Test coverage is a measure used in software testing to determine how thoroughly the testing process has exercised the software. It's expressed as the extent to which a particular structure, such as code, requirements, or functionalities, has been executed by the test suite, often represented as a percentage of the items covered. Essentially, it indicates how much of the software under test has been executed at least once by the tests, revealing which parts have been tested and which have not. This metric serves as an indicator of both the effectiveness and the completeness of the testing process, influencing confidence in the software's reliability [4].

5. Mean time to failure (MTTF)

Mean Time to Failure (MTTF) measures the average time an asset operates before failure. Effectively, MTTF measures the reliability of non-repairable assets by looking at similar assets over a period of time and measuring how long they perform until failure. The calculation uses time units—for example: hours, days, years—to express MTTF. A higher MTTF signifies a more reliable system, with longer intervals between failures. A lower MTTF warns of potential flaws or increased risk of breakdowns. MTBF is largely based on assumptions and definition of failure and attention to these details are paramount to proper interpretation.[5]

To calculate MTTF, divide the total number of hours of operation by the total number of assets in use.

- **MTTF =** Operating time to failure (in hours) / Number of Assets in Use
- **Failure** = when an asset is no longer able to perform its intended function
- **Operating time** = the timeframe that an asset or component is in operation

Calculating MTTF with a larger number of assets will lead to a more result as MTTF represents the average time to failure.

6. Mean time to repair (MTTR)

Mean time to repair (MTTR) Sometimes referred to as mean time to recovery, is a metric that is used to measure the average time it takes to repair a system or piece of equipment after it has failed.[6]

MTTR is often used alongside Mean Time Between Failures (MTBF) to measure the availability and reliability of assets. Tracking both metrics can help to identify system-wide failures and promote a more proactive approach to maintenance.

The MTTR formula is calculated by dividing the total unplanned maintenance time spent on an asset by the total number of failures that asset experienced over a specific period. Mean time to repair is most commonly represented in hours.

The MTTR calculation assumes that:

- Tasks are performed sequentially
- Tasks are performed by appropriately trained personnel

MTTR = Total maintenance time ÷ Number of repairs

What is considered a world-class MTTR is dependent on several factors, like the type of asset, its criticality, and its age. However, a good rule of thumb is an MTTR of under five hours.

7. Performance Metrics

Performance metrics in software engineering are quantitative measures used to assess the efficiency, speed, and reliability of a system. These metrics help developers evaluate whether an application meets user expectations and functions optimally. Key performance indicators include **response time, throughput, resource utilization, and error rate.** Continuous monitoring and optimization—through code improvements, hardware upgrades, and system refinements—are essential to maintaining performance. Regular testing and automated monitoring tools help detect bottlenecks early, ensuring software remains responsive and scalable as demands grow.[7].

8. Customer-Reported Incidents

Customer-reported incidents are a reality for anyone managing software systems. No matter how thoroughly an application is tested, users will eventually find issues that developers might have missed. These incidents range from minor inconveniences to critical failures that can severely impact user satisfaction.

Effectively managing these incidents means staying on top of a few important practices including **Incident Tracking and Classification, Root Cause Analysis, Response Time Metrics, and Post-Incident Reviews**

Ignoring or mishandling incidents can quickly damage a product’s reputation, while prompt and thorough responses help maintain user confidence. Building a robust incident management framework with automated tracking and monitoring can make all the difference. [8]

References

[1] Rahmani, Cobra & Khazanchi, Deepak. (2010). A Study on Defect Density of Open Source Software. Proceedings - 9th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2010. 679-683. 10.1109/ICIS.2010.11.

[2] Menusha Katipearachchi, Chathura Sandareka, Arosha Weerasinghe, The Relationship between Code Complexity and Software Quality: An Empirical Study, 2023, 10.1109/ICSE-SA4SE57176.2023.00011.

[3] J. Kraaijeveld, "Exploring characteristics of code churn," M.S. thesis, Delft University of Technology, Delft, The Netherlands, 2013.

[4] J. W. Hollen and P. S. Zacarias, "Exploring Code Coverage in Software Testing and its Correlation with Software Quality: A Systematic Literature Review," Bachelor of Science Thesis in Software Engineering and Management, https://gupea.ub.gu.se/bitstream/handle/2077/38588/gupea_2077_38588_1.pdf

[5] Wendy Torell and Victor Avelar, "Mean Time Between Failure: Explanation and Standards ," researchgate. https://www.researchgate.net/profile/Victor-Avelar/publication/251895269_Mean_Time_Between_Failure_Explanation_and_Standards/links/5a3c91d8a6fdcc21d8780af5/Mean-Time-Between-Failure-Explanation-and-Standards.pdf

[6]IBM, "MTTR," Ibm.com, Apr. 27, 2023. https://www.ibm.com/think/topics/mttr

[7] McKinsey & Company. (2023). Yes, You Can Measure Software Developer Productivity. Retrieved from McKinsey & Company

[8] Silva, L. and Almeida, J. (2024). A Framework for Managing Customer-Reported Incidents in Software Engineering. International Symposium on Technology, 2024.