International Journal of Engineering and Management Research
https://ijemr.vandanapublications.com
e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-13, Issue-2 (April 2023)
https://doi.org/10.31033/ijemr.13.2.31

Software Understandability using Software Metrics: An Exhaustive Review

Mohammad Saif Himayat[1] and Dr. Jameel Ahmad[2]
[1]PG Student, Department of Computer Science & Engineering, Integral University, INDIA
[2]Associate Professor, Department of Computer Science & Engineering, Integral University, INDIA

[2]Corresponding Author: jameel@iul.ac.in

Received: 21-03-2023 **Revised:** 12-04-2023 **Accepted:** 29-04-2023

ABSTRACT

Stability and Understandability are two important aspects of software architecture quality measurement. Stability refers to the degree to which software architecture is resistant to change. A stable architecture is one that can accommodate changes with minimal impact on the overall system. In other words, a stable architecture is one that can evolve over time while maintaining its integrity. There are several metrics used to measure the stability of a software architecture, including the number of dependencies between components, the number of changes required to implement a new feature or fix a bug, and the amount of time it takes to make changes to the architecture. A stable architecture will have fewer dependencies between components, require fewer changes to implement new features or fix bugs, and make changes faster. Understandability refers to the degree to which a software architecture can be easily understood by developers and other stakeholders. An understandable architecture is one that is easy to navigate and comprehend, and that clearly communicates the system's design and functionality. There are several metrics used to measure the understandability of software architecture, including the number of components and their relationships, the level of abstraction used, and the degree of consistency between different parts of the architecture. An understandable architecture will have a clear structure and logical organization, use consistent terminology and notation, and be documented with clear explanations of its components and their relationships. a software architecture that is both stable and understandable will be easier to maintain, evolve, and extend over time, reducing the risk of errors and improving the overall quality of the system.

***Keywords*—** Software Architecture, Software Engineering, Stability, Quality Attributes, Understandability

I. INTRODUCTION

Software evaluation using software matrices is a common practice to assess the quality of software. Understandability is one of the key aspects of software quality, which refers to the ease with which software can be understood by its users, developers, and maintainers.

Some software matrices that can be used to evaluate software understandability:

Halstead Software Metrics: These metrics were developed by Maurice Halstead in the 1970s and measure the complexity of software based on the number of operators and operands used in the code. These metrics include program length, vocabulary size, volume, difficulty, and effort. Higher values of these metrics indicate lower understandability.

Cyclomatic Complexity: This metric measures the number of independent paths through a software program. Higher values of cyclomatic complexity indicate a more complex program and lower understandability.

Maintainability Index: This metric measures the ease with which software can be maintained. It considers factors such as code size, complexity, and documentation. Higher values of the maintainability index indicate higher understandability.

Code Readability Metrics: These metrics evaluate the readability of the code, which is closely related to understandability. Some examples of code readability metrics include variable naming conventions, code indentation, and comments. Lower values of these metrics indicate lower understandability.

Cognitive Complexity Metrics: These metrics measure the cognitive load required to understand the software. They include metrics such as nesting level, control flow complexity, and method complexity. Higher values of cognitive complexity metrics indicate lower understandability.

Software matrices can be useful in evaluating the understandability of software. However, it is important to note that these metrics should be used in conjunction with other evaluation methods, such as user testing and code reviews, to obtain a comprehensive understanding of software quality.

II. METHODOLOGY

LOC in Software

Software development, LOC (lines of code) is a metric used to measure the size of a software program. It refers to the total number of lines of code in a software system, including comments and blank lines.

The LOC metric is often used to estimate the effort required to develop or maintain a software system and to track the productivity of software developers. However, it is important to note that the LOC metric has limitations and should be used in conjunction with other metrics to assess software quality.

One limitation of the LOC metric is that it does not necessarily indicate the complexity or functionality of a software system. Two programs with the same number of lines of code can have vastly different functionality and complexity, depending on the programming language, coding style, and design.

Moreover, some programming languages, such as Python, tend to have fewer lines of code than other languages like C or Java, even though they may have similar functionality. Thus, LOC may not always be a fair comparison between programs written in different programming languages.

LOC is a useful metric for measuring the size of a software system, but it should not be the only metric used to evaluate software quality. It is important to consider other factors such as functionality, complexity, maintainability, and performance when assessing the overall quality of a software system.

Resource Standard Metrics (RSM)

Is a software tool that provides a suite of software metrics to evaluate the quality and maintainability of software systems. RSM can be used to analyze code written in various programming languages, including C, C++, Java, and COBOL.

Some of the metrics provided by RSM include:

Lines of Code (LOC): Measures the number of lines of code in a software system, including comments and blank lines.

Cyclomatic Complexity: Measures the number of independent paths through a software program and can be used to estimate the testing effort required for a software system.

Halstead Complexity Measures: Measures the complexity of a software system based on the number of operators and operands used in the code

Code Maintainability Index (CMI): Measures the ease with which a software system can be maintained, based on factors such as code size, complexity, and documentation.

Depth of inheritance: Measures the number of levels in an inheritance hierarchy and can be used to evaluate the design and maintainability of object-oriented software systems.

RSM also provides various reports and visualizations to help developers and project managers interpret and understand the software metrics. For example, the "Code Structure Report" provides an overview of the software system's organization and complexity, while the "Function Metrics Report" provides detailed metrics for individual functions or modules.

Web document information extraction using the class attribute approach is a technique used to extract relevant data from HTML web pages. This approach involves analyzing the HTML structure of the page and identifying elements that are tagged with specific class attributes. Class attributes are used to group similar elements together and provide a way for developers to apply styles and formatting to specific elements [1]

III. PRIOR APPROACH

Software Engineering and Machine Learning

Software engineering and machine learning are two fields that have become increasingly intertwined in recent years, as machine learning techniques have become more prevalent in software development. Machine learning refers to the use of statistical algorithms and models to enable computers to learn and make predictions or decisions without being explicitly programmed.

Some examples of how software engineering and machine learning are being used together:

Predictive Maintenance: Machine learning algorithms can be used to analyze data from sensors in industrial equipment and predict when maintenance is needed. This can help Reduce downtime and improve efficiency.

Natural Language Processing (NLP): NLP techniques, which are used to analyze and understand human language, are often used in software applications such as chatbots, virtual assistants, and customer service systems.

Image and Video Recognition: Machine learning algorithms can be trained to recognize and classify images and videos, which can be used in applications such as security systems, autonomous vehicles, and medical imaging.

Fraud Detection: Machine learning algorithms can analyze transaction data and detect patterns that indicate fraudulent activity, which can be used in applications such as banking and e-commerce.

Personalization: Machine learning can be used to analyze user data and provide personalized recommendations and experiences in applications such as social media, e-commerce, and entertainment.

To effectively apply machine learning techniques in software engineering, developers need to have a solid understanding of both fields. This includes knowledge of

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-13, Issue-2 (April 2023)
https://doi.org/10.31033/ijemr.13.2.31

programming languages, software design principles, and machine learning algorithms and techniques. Collaboration between software engineers and data scientists is also important to ensure that machine learning models are effectively integrated into software applications and meet the requirements of end-users [2].

IV. SOFTWARE ENGINEERING WITH TASK SCHEDULING

Task scheduling is an important aspect of software engineering that involves the scheduling and allocation of tasks and resources within a software system. Task scheduling is particularly important in real-time systems and multi-tasking systems, where tasks need to be scheduled and executed in a timely and efficient manner.

Here are some key concepts related to task scheduling in software engineering.

Preemptive vs. Non-Preemptive Scheduling: Preemptive scheduling allows higher-priority tasks to interrupt lower-priority tasks, while non-preemptive scheduling does not allow interruptions. Preemptive scheduling is often used in real-time systems where responsiveness is critical

Round-Robin Scheduling: In round-robin scheduling, each task is allocated a fixed amount of time to execute, and tasks are executed in a circular order. This ensures that all tasks are executed fairly and can help prevent starvation

Priority scheduling: In priority scheduling, tasks are assigned a priority level, and higher-priority tasks are executed first. This can be useful in real-time systems where certain tasks have higher urgency than others [3].

Task Dependencies: In complex software systems, tasks may have dependencies on other tasks or resources. Task scheduling algorithms need to take these dependencies into account to ensure that tasks are executed in the correct order and that resources are allocated appropriately [4].

Effective task scheduling is essential for ensuring the reliability and performance of software systems. Software engineers need to carefully analyze the requirements of the system and design task scheduling algorithms that meet those requirements while considering factors such as task dependencies, resource constraints, and real-time responsiveness. There are also software tools available that can assist with task scheduling and resource allocation, such as scheduling algorithms and simulators [5].

Load balancing: Load balancing involves distributing tasks across multiple processors or nodes to ensure that resources are utilized efficiently, and tasks are executed in a timely manner [6].

V. SOFTWARE METRICES USED FOR ANOMOLY DETECTION

Software metrics can be used for anomaly detection in software development projects. Anomalies are deviations from the expected behavior or normal patterns in software development, and they can occur due to a variety of reasons such as coding errors, data quality issues, security breaches, and performance problems. software metrics and identifying anomalies, software developers can take corrective actions to address the underlying issues and improve the quality, reliability, and security of the software system. However, it's important to note that software metrics are just one tool for anomaly detection and should be used in conjunction with other techniques, such as code reviews, testing, and monitoring [7,8].

VI. SOFTWARE METRICES FOR INFORMATION SECURITY FRAMEWORK

Software metrics can be used to measure and improve an organization's Information Security Framework (ISF)[9]. Here are some common software metrics that can be used for an ISF, organizations can identify areas where their ISF needs improvement and take corrective actions to address the underlying issues [8]. However, it's important to note that software metrics are just one tool for measuring and improving an ISF and should be used in conjunction with other techniques, such as risk assessments, security audits, and employee feedback [10]

VII. VARIOUS SOFTWARE METRICS USED FOR SOFTWARE SYSTEM MODEL

Software developers can identify areas where their software system models need improvement and take corrective actions to address the underlying issues. However, it's important to note that software metrics are just one tool for measuring and improving software system models and should be used in conjunction with other techniques, such as peer reviews, testing, and documentation [11].

VIII. CONCLUSION

Software understandability refers to the ease with which software can be understood by its users, including software developers, maintainers, and end-users. There are several software metrics that can be used to measure software understandability. software metrics and taking

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-13, Issue-2 (April 2023)
https://doi.org/10.31033/ijemr.13.2.31

corrective actions to address issues, software developers can improve the overall understandability of their software programs. This, in turn, can improve software quality, reduce maintenance costs, and enhance end-user satisfaction. Software matrices can be used to monitor the software system model, it can be used to establish the security of the system. various anomaly can be detected and corrected and many more.

REFERENCES

[1] S. Srivastava, M. Haroon & A. Bajaj. (2013). Web document information extraction using class attribute approach. *4th International Conference on Computer and Communication Technology (ICCCT)*, Allahabad, India, pp. 17-22. DOI: 10.1109/ICCCT.2013.6749596.

[2] Haroon, M., Tripathi, M. M. & Ahmad, F. (2020). Application of machine learning in forensic science. In: *Critical Concepts, Standards, and Techniques in Cyber Forensics*, pp. 228-239.

[3] R. Khan, M. Haroon & M. S. Husain. (2015). Different technique of load balancing in distributed system: A review paper. *Global Conference on Communication Technologies (GCCT), Thuckalay, India*, pp. 371-375. DOI: 10.1109/GCCT.2015.7342686.

[4] M. Haroon & M. Husain. (2015). Interest attentive dynamic load balancing in distributed systems. *2nd International Conference on Computing for Sustainable Global Development (INDIA Com), New Delhi, India*, pp. 1116-1120.

[5] Haroon, M. & Husain, M. (2013). Analysis of a dynamic load balancing in multiprocessor system. *International Journal of Computer Science engineering and Information Technology Research*, *3*(1).

[6] Wasim Khan & Mohammad Haroon. (2022). An unsupervised deep learning ensemble model for anomaly detection in static attributed social networks. *International Journal of Cognitive Computing in Engineering*, *3*, 153-160. DOI: https://doi.org/10.1016/j.ijcce.2022.08.002.

[7] Khan, W. (2021). An exhaustive review on state-of-the-art techniques for anomaly detection on attributed networks. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(10), 6707-6722.

[8] Husain, Mohammad Salman & Haroon, Dr. Mohammad. (2020). An enriched information security framework from various attacks in the IoT. *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, *8*(3). Available at: https://ssrn.com/abstract=3672418.

[9] Husain & Mohammad Salman. (2020). A review of information security from consumer's perspective especially in online transactions. International Journal of Engineering and Management Research, 10(4). Available at: https://ssrn.com/abstract=3669577.

[10] A. M. Khan, S. Ahmad & M. Haroon. (2015). A comparative study of trends in security in cloud computing. *Fifth International Conference on Communication Systems and Network Technologies, Gwalior, India*, pp. 586-590. DOI: 10.1109/CSNT.2015.31.

[11] Haroon, M., & Husain, M. (2013). Different types of systems models for dynamic load balancing. *IJERT*, *2*(3).