EasyChair Preprint

№ 10168

The Effectiveness of Software Complexity Metrics in Predicting Software Performance Bottlenecks

Pasindu Madhuwantha, Chamod Kodithuwakku,
Ashen Jayarathne, Manisha Karunanayake, Dilshan De Silva and
Samitha Vidhanaarachchi

May 15, 2023

The Effectiveness of Software Complexity Metrics in Predicting Software Performance Bottlenecks

Madhuwantha M.G.P, Kodithuwakku D.R.G.C.W, Jayarathne A.H.B, Karunanayake M.L, D. I. De Silva, Samitha Vidhanaarachchi
[1] *Department of Computer Science & Software Engineering*
Faculty of Computing, Sri Lanka Institute of Information Technology,
New Kandy RD, Malabe, Sri Lanka
[1]Corresponding Author: mpasindu72@gmail.com

***Abstract*—Customer demands for software services must be met in terms of reaction speed, productivity, accessibility, and availability. Software engineers refer to these issues as performance bottlenecks, particularly in terms of software systems when they refuse to operate at the required level of services. Software, and even hardware resource constraints, as well as badly written programming, may function as a bottleneck to software performance.**

***To address these challenges, this paper proposes the use of software complexity metrics to evaluate the complexity of software code and identify potential performance bottlenecks. The paper identifies three different types of complexity metrics, including McCabe's cyclomatic complexity metrics, NPATH complexity metrics, and the complexity of Halstead software science metrics. These metrics are used to quantify various aspects of software complexity, including data flow, control flow, and size.**

***The paper argues that software complexity metrics can be a valuable tool for identifying and managing performance bottlenecks in software systems. Excessive complexity can lead to poor software performance, and regular monitoring and management of complexity can help prevent performance bottlenecks. While high complexity can be an indicator of potential issues, other factors such as hardware limitations or external factors could also come into play.**

***In addition to identifying performance bottlenecks, software complexity metrics can be used to track complexity over time, which can help identify trends and potential areas for optimization. The paper recommends that software organizations regularly monitor and manage software complexity using these metrics to ensure that software systems perform at the required level of service.**

***The paper also discusses the importance of accurately measuring software complexity and the challenges involved in doing so. Different programmers might have different perceptions of what constitutes "complex" code, or different methods for measuring complexity might produce conflicting results. The authors suggest that integrating complexity metrics into existing development processes could help overcome some of these challenges.**

***Finally, the paper highlights the potential benefits of using software complexity metrics beyond identifying performance bottlenecks. For example, tracking complexity over time could help identify trends and potential areas for optimization. The paper concludes that software complexity metrics can be a valuable tool for software organizations looking to ensure that their systems perform at the required level of service.**

Keywords—complexity metrics, McCabbe's cyclomatic complexity metrics, NPATH complexity metrics, complexity of Halstead software science metrics

I. INTRODUCTION

Enhancing software quality is a quantifiable indicator of software code excellence. Given that the software complexity could be calculated on widely recognized software parameters, the testing stage of the procedure is likely to take less time and expense to estimate, particularly because that stage could only be operated after the completion of software coding. This may be done by defining metrics, the parameters of which may be determined by looking at how a programme or source code is written. There are nonetheless many software metrics that are employed often in the software business that are still poorly comprehended [1].

Software engineers may not have the skills and resources needed to guarantee the precision and dependability of the programs they develop. Software metrics have been created to give a standardized method of monitoring software development in order to meet this problem. These metrics may be used in a variety of software development fields, including healthcare, banking, and e-commerce, to make sure that software meets the necessary standards for functionality and quality. The fundamental objective of this software complexity metrics to ensure that bottlenecks are found during the SDLC process and clients are satisfied during the whole development process, rather than just at release. Thus, by comparing McCabbe's cyclomatic complexity metrics, NPATH complexity metrics, and the complexity of Halstead software science metrics, this paper aims to identify the metrics that can predict software bottlenecks earlier and faster.

Although software metrics have shown themselves to be a useful tool for controlling and monitoring software development, putting them into practice presents certain difficulties. For instance, some elements of software development, including intricate mathematics, images, and tables, could be difficult to quantify using conventional measurements. To make sure that these components are monitored and handled properly in such circumstances, software engineers may need to establish their own unique tailored metrics.

Software metrics should be versatile and adaptive to various development settings to assist overcome these issues. To guarantee that the metrics are appropriate for the current project, it is necessary for project managers, software developers, and other stakeholders to work closely together.

Programs for education and training can also assist developers in becoming more familiar with the application of software metrics and their possible advantages.

Software metrics are a crucial instrument for verifying the precision, dependability, and functionality of software products, to sum up. However, their implementation might be difficult, and developers may have to alter the metrics to meet certain requirements. Software metrics may be a useful tool for enhancing the effectiveness and efficiency of software development processes with the right planning, coordination, and training.

Despite this fact, certain software complexity measurements were put forward more than 30 years ago, while others came later. Sometimes, the complexity of the source code is used to measure software progress. Multiple measurements are employed, making it impossible to distinguish between methods and outcomes. Additionally, evaluating for a specific source code is not practicable or easy [2]. Software complexity refers to how challenging it can be to use and understand a programme [3]. The extent in which the qualities that make maintenance difficult have been identified and are influenced by programme complexity is known as software maintainability [3]. Figure 1 depicts these interdependencies.

Software complexity metrics are created to assess the degree of complexity and maintainability of software systems. These indicators can be used to spot possible performance bottlenecks in the software development cycle. Developers and testers may optimize software performance and improve the efficiency and caliber of their testing by being aware of the elements that affect it.

The purpose of this study is to evaluate the performance of three software complexity metrics—McCabbe's cyclomatic complexity metrics, NPATH complexity metrics, and the complexity of Halstead software science metrics—in identifying software performance bottlenecks. The number of separate pathways that may go through a piece of code is counted using McCabbe's cyclomatic complexity metrics, which can reveal both the complexity of the code and any possible performance problems. In order to provide a more thorough understanding of software complexity, NPATH complexity measurements count all potential routes through a program. Finally, the complexity of Halstead software science metrics assesses the code's complexity using variables such the quantity of operators and operands, adding still another degree of understanding to software complexity and possible performance problems.

This study offers useful insights into how software developers and testers may optimize software performance and enhance testing effectiveness and quality by looking at the usefulness of these three software complexity measures in anticipating performance bottlenecks. These indicators enable developers to see possible performance problems early on in the development cycle, cutting costs and raising the overall quality of the software product.

II. Literature Review

Software metrics are described as the on-going implementation of calculation-related methodologies to the entire SDLC procedure and its final products for providing valuable and promptly available information to management, in addition to the implementation of such methods to enhance the procedure and its merchandise [4].

Gill and Grover say that the level of complexity in assessing, testing, developing, and altering software is known as software complexity [5]. Grady emphasizes that the main application of software metrics pertains to decision-making processes [6]. According to him, software metrics are employed to gauge particular features of a software procedure or application. Metrics assist developers in choosing between two decisions. This term also identifies one of the issues with modern software advancement, which is a shortage of data for forecasting and assessing projects related to software.

The below figure shows the connection between software complexity metrics and software systems.

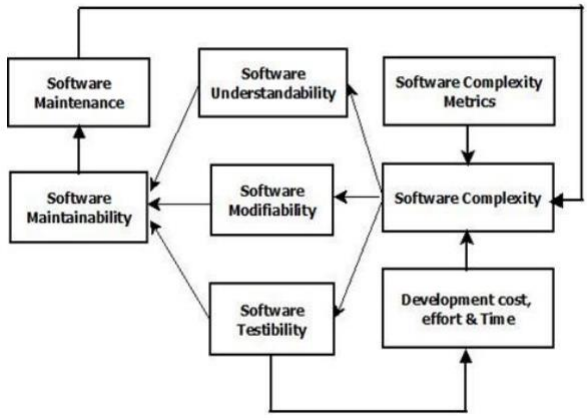

Fig. 1. Connection between software complexity metrices and software systems

Software complexity may either be seen as the resources required for the endeavor or as the efficacy of the work. According to this method, the level of difficulty of an issue may be described as the quantity of resources needed for an ideal resolution. The materials to carry out a particular approach can subsequently be utilized to determine how difficult the answer is. These materials include at least two components, which are time and space. Time refers to the man-hours or time used by the computer while space refers to the memory of the computer [4].

McCabe postulated that the cyclomatic frequency of the program's flowgraph may be used to gauge software complexity. One benefit of this metric is that it can be quite simple to calculate from the programme code and flow diagram. McCabe's cyclomatic value is supported by the majority of code measurement software equipment, which is an indication of how user-friendly it is [7].

The metric encourages a top-down approach to development to manage module difficulty at the conceptual stage, or before any coding is done. Additionally, it may be employed to calculate the complexity of programme modules in relation to McCabe's suggestion of a maximum of 10. Additionally, it may be used to discover the most basic

programme design and as an example for distributing testing resources in controlling alternative programme design [8].

It might be difficult for a person to comprehend a programme with an excessive amount of cyclomatic complexity, yet it might not be as difficult for a machine to comprehend it, according to some researchers who claim that it just evaluates psychological difficulty and not the amount of computing power [9].

Furthermore, it sees both iterative and selective predicates as adding the exact same level of difficulty. It continues to be one of the most often used metrics of software complexity regardless of all of its drawbacks, most likely since it is straightforward to comprehend and apply [9].

III. METHODOLOGY

Qualitative research is an important method for assessing software complexity measures since it enables a thorough review of the advantages and drawbacks of various metrics. Three distinct forms of software complexity measures—McCabbe's cyclomatic complexity metrics, NPATH complexity measurements, and the complexity of Halstead software science metrics—are compared, contrasted, and evaluated in this study using qualitative research.

Software developers frequently utilize McCabbe's cyclomatic complexity metrics because they offer a straightforward yet accurate technique to gauge the complexity of code. On the other side, NPATH complexity measurements provide a more thorough view of software complexity by assessing all potential pathways through a program. Finally, by counting the number of operators and operands in the code, the complexity of Halstead software science metrics adds another degree of understanding to the concept of software complexity.

The comparison and assessment of three distinct forms of software complexity metrics—McCabbe's cyclomatic complexity metrics, NPATH complexity metrics, and complexity of Halstead software science metrics—provided in this study will be very helpful to software developers and testers. This research gives helpful insights into the benefits and drawbacks of each statistic by contrasting and comparing these diverse methodologies.

The software development industry favors McCabbe's cyclomatic complexity metrics because they offer a quick and accurate approach to gauge the complexity of code. By analyzing all potential pathways through a program, NPATH complexity metrics provide a more thorough picture of software complexity. Last but not least, by assessing the quantity of operators and operands in the code, the complexity of Halstead software science metrics offers an extra degree of insight into software complexity.

Software testers and developers may select the measure that best fits their specific requirements and streamline their software development process by being aware of the advantages and disadvantages of each metric. As a result, testing efficiency and quality may increase while software development costs may decrease.

In addition to helping software developers and testers, this research advances the discipline of software engineering by illuminating the numerous software complexity measures and their applications in foretelling bottlenecks in program performance. Overall, this study emphasizes the value of periodically tracking software complexity using the right metrics and using the best statistic for the job.

Metrics for Software Lifecycle, sometimes referred to as management metrics, should be considered first when comparing various forms of software complexity metrics. This category entails evaluating a number of software development process elements, including methodology, reuse, effort, cost, and progress metrics. Metrics for methodology concentrate on the techniques and methods utilized in software development. The amount of code reused during the software development process is measured by reuse metrics, while the resources needed to finish the project are evaluated by effort metrics. Progress metrics track the project's advancement over time, whereas cost indicators assess the costs related to software development. In general, these indicators offer perception into the software development process and aid in making sure the project is on track to meet its objectives.

Metrics for Software Product makes up the second category. Program characteristics are measured by programme process metrics, often known as quality metrics. These criteria include style, complexity, size, cost, reusability, portability, effectiveness, performance, functionality, usability, and dependability metrics. These criteria evaluate how challenging the program's size, scope, or written documentation are..

The number of Code Lines (LOC) in the program is the third category of software complexity measures. This statistic indicates the difficulty of the program's development. Many academics see it as an essential unit of measurement for software complexity. This measure is simple to grasp since it simply counts the number of source instructions necessary to solve a problem. However, lines used for blank commands and remark lines are not tallied when calculating the number of instructions. While this statistic is valuable, it is vital to evaluate other metrics as well, because some programs may have many code lines owing to lengthy comments or other non-functional code.

The scale and complexity of modern software systems need the use of efficient testing methods. Size characteristics are terms used for defining things like physical size and mass. Size measures include code lines and Halstead's software science [10], which is a measure that M. Halstead suggested.

A. McCabbe's cyclomatic complexity

One statistic that focuses on the control and data flow more than the programme size is this method, which supports a 1976 model of a specification flow graph created by Thomas J. McCabb. [11] In this model, control flow is illustrated via a programme graph. The borders indicate control flow between nodes, while nodes indicate processing tasks. A great instance of control flow measurements is McCabe's metrics [12]. The following approaches can be used to calculate the cyclomatic number via V (G). McCabb's Complexity has a flaw in that it cannot differentiate between several control flow architectures and its conditional statements. Moreover, it disregards the degree of layering across different control flow topologies. NPATH is superior to the McCabb's metric [13].

B. NPATH complexity metric for control flow

The management framework of a programme serves as the basis for the control flow difficulty measurements. The control

flow metric NPATH, developed by Nejmeh [14], quantifies the total amount of operation pathways that pass through a function's parameters. It quantifies acyclic execution pathways. One illustration of control flow indicators is NPATH. NPATH complexity (NC), one of the most used software complexity metrics. The drawbacks of McCabe's measure, which do not distinguish between distinct types of flow controls and nesting layers mechanisms, are solved by NPATH, another measure of software complexity.

C. The complexity of Halstead software science

Software science metrics for complexity measurements of software products were first proposed by M. Halstead [10]. The foundation of Halseatd's software science is an improvement on counting the lines of code when estimating programme size. The total amount of operands, operators, and their corresponding occurrence in the programme (code) are all counted using Halstead's metrics. When calculating the time, difficulty, effort, estimated length, volume, vocabulary and length of the programme, the following calculations should take these operands and operator combinations into account. Because control flow complexity is impossible to calculate during quick and simple computation, this complexity measurement has a significant flaw.

IV. Results and Discussion

The paper uses a static analysis of control flow and size measures to measure software complexity. This means that the analysis is done without actually running the program. Instead, it looks at the program's structure and measures its complexity based on that. The paper specifically looks at three types of complexity metrics commonly used in the literature: NPATH, McCabe complexity, and Halstead's complexity metrics. These metrics can help identify potential bottlenecks in the program and optimize its performance. It is important to note that the performance of the program's source code is not directly related to the statistical evaluation of metrics. However, using these metrics can help identify potential areas for improvement and guide the development and testing process.

When it comes to measuring software complexity, this paper adopts a static analysis approach that takes into account control flow and size metrics. The study examines three popular complexity metrics found in the literature, namely NPATH, McCabe complexity, and Halstead's complexity metrics, with the aim of identifying bottlenecks and optimizing the software development process.

The three elements that can effect software performance are program size, data organization, and control flow. Bottlenecks can be predicted by each of these criteria in different ways. NPATH, for example, counts the total number of processing pathways via a function and evaluates their acyclic nature. Halstead's metrics, on the other hand, count the number of operators, operands, and their occurrences in the code to assess the complexity, effort, expected length, volume, vocabulary, and length of the program.

McCabb's complexity metrics, on the other hand, use a program graph to represent the control flow. The graph has nodes that represent processing tasks, each consisting of one or more code statements, and edges that indicate the control flow between the nodes.

While these indicators can aid in identifying possible bottlenecks and improving software development, they do not directly evaluate program performance. Nonetheless, by examining these data, software developers and testers may get useful insights into their code's strengths and shortcomings and adjust their development process appropriately.

V. Conclusion

Software complexity metrics play a crucial role in software development by assessing the quality of the software being produced. With the increasing complexity of software systems, the need for efficient testing methods is becoming more crucial. Software complexity metrics can help testers by counting the number of acyclic execution paths through a program, leading to better testing and higher quality software.

In this context, software complexity measurements such as NPATH, McCabb's complexity metrics, and Halstead's Software Science Complexity can aid in the identification of bottlenecks and the improvement of software development. NPATH counts the number of acyclic execution pathways in a function, assessing their acyclic nature and assisting testers in identifying potential issues. McCabb's complexity metrics reflect control flow using a program graph, with edges denoting control flow between nodes and nodes representing processing jobs. Potential bottlenecks can be found by examining the complexity of the control flow. Halstead's Software Science Complexity counts the number of operators, operands, and their occurrences in the program, enabling for the calculation of time, difficulty, effort, predicted length, volume, vocabulary, and length.

Developers and testers can detect bottlenecks and enhance software quality by comparing the results of certain software complexity indicators. This can assist to reduce software development costs while also enhancing testing effectiveness and software quality. In conclusion, software complexity measures are critical for measuring software quality, and their application may assist guarantee that software development is productive, efficient, and generates high-quality outputs.

References

[1] T. J. M. Cabe, A complexity measure, IEEET Ransactions on Software Engineering, vol. 2, 1976

[2] I. Herraiz, J. M. G. Barahona, and G. Robles, Towards a theoretical model for software growth, in 29th International Conference on Software Engineering Workshops (ICSEW'07).

[3] W. Harrison, K. Magel, R. Kluczny, and A. Dekok, Applying Software Complexity Metrics to Program Maintenance Compute, vol. 15, pp. 65-79, 1982

[4] Goodman, P. Practical Implementation of Software Metrics. London: McGrawHill, 1993

[5] Grover, P.S. & Gill, N.S. Composite Complexity Measures (CCM). In Lee, M., Barta, B.-Z., & Juliff, P. (Eds.), Software Quality and Productivity: Theory, Practice, Education and Training (pp. 279-283). London: Chapman & Hall, 1995

[6] Grady, R. B. Practical Software metrics for Project management and Process Improvement. New Jersey: Prentice Hall, 1992.

[7] McCabe, T. A software complexity measure, IEEE Transactions on Software Engineering, SE-2(4), 308-320, 1976.

[8] Fenton, N. E. & Pfleeger, S. L. Software Metrics – A Rigorous and Practical Approach. London: International Thomson Computer Press, 1996.

[9] Zuse, H. Software Complexity – Measures and Methods. Berlin: Walter de Gruyter & Co, 1991

[10] M. Halstead, Elements of Software Science. North Holland, 1977

[11] T. A. McCabe, A complexity measure, IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, December 1976.

[12] A. Fitzsimmons and T. Love, A review and evaluation of software science, Computing Survey, vol. 10, no. 1, March 1978.

[13] E. E. Millis, Software metrics, SEI Curriculam Module SEI- CM. vol. 12, no. 2.1, Dec, 1988.

[14] B. A. Nejmeh, NPATH: A measure of execution path complexity and its applications, Comm. of the ACM, vol. 31, no. 2, pp. 188-210, February 1988