Amity Journal of Computational Sciences
ISSN 2456-6616 (Online)
Volume 6 Issue 2, 2022

Can Rust finally replace C?: A qualitative and quantitative analysis

Aryan Gulati
Bachelor of technology, Computer Science Engineering
Amity University, Uttar Pradesh, India

***Abstract* - The aim of this research paper is to conduct a comprehensive analysis of Rust and C, with the objective of assessing whether Rust can finally replace C as a preferred programming language. The analysis encompasses various aspects such as performance benchmarks, memory management, error handling mechanisms, security vulnerabilities, language syntax and readability, type safety and memory management, tooling and libraries support, development workflows, learning curve and community support, the Rust ecosystem, adoption in various industries, case studies of successful Rust projects, and a comparison of industry trends and language preferences. The findings and analysis highlight the strengths and considerations of each language, providing insights into the potential of Rust to replace C in different domains.**

Keywords - Comparative Analysis, Learning Curve, Benchmark Testing

I. INTRODUCTION

The programming landscape is constantly evolving, with new languages emerging and established languages undergoing advancements. One such evolution has been the rise of Rust, a systems programming language known for its focus on safety, performance, and concurrency. Rust has garnered attention as a potential alternative to C, a widely used language renowned for its efficiency and low-level control. In this research paper, we delve into the question: Can Rust finally replace C?

A. Background and Significance of the Topic:

C has been the language of choice for systems programming, embedded systems, and other domains that require fine-grained control and high-performance code. However, C's manual memory management and weak type checking make it susceptible to memory-related vulnerabilities and runtime errors. The need for a safer and more reliable alternative led to the development of Rust.

Rust addresses the shortcomings of C by introducing a unique ownership system and borrow checker that guarantee memory safety without sacrificing performance. This innovative approach has attracted attention from developers and organisations seeking to build secure and efficient software systems. Rust's features and ecosystem have grown rapidly, making it a potential candidate to replace C in various industries.[1]

B. Research Objectives:

The primary objective of this research paper is to conduct a qualitative and quantitative analysis of Rust and C, assessing Rust's potential to replace C as a preferred programming language. To achieve this objective, we will explore and evaluate various aspects, including performance benchmarks, memory management techniques, error handling mechanisms, security vulnerabilities, language syntax and readability, type safety and memory management, tooling and libraries support, development workflows, learning curve and community support, the Rust ecosystem, adoption in various industries, case studies of successful Rust projects, and a comparison of industry trends and language preferences.

C. Scope and Methodology:

The scope of this research paper encompasses an in-depth analysis of Rust and C, focusing on their strengths, considerations, and applicability in different domains. The analysis will be conducted through a combination of qualitative research, such as literature

review and case studies, and quantitative analysis, including performance benchmarking and comparative studies. Relevant industry trends, language preferences, and community feedback will be considered to provide a comprehensive assessment of Rust's potential to replace C.

By examining these various aspects, we aim to contribute to the understanding of Rust's capabilities and assess its suitability as a replacement for C. The findings and analysis presented in this research paper will serve as a valuable resource for developers, organisations, and technology enthusiasts who are evaluating the adoption of Rust or seeking insights into the evolving programming landscape.

II. ANALYSIS OF PERFORMANCE BENCHMARKS

A. What is benchmarking?

Benchmarking is a process of evaluating the performance, efficiency, or quality of a system, product, or process by comparing it against a set of established standards or benchmarks. It involves measuring and analysing the performance metrics of the subject under test and comparing them to the metrics of other similar systems or industry best practices.

The primary goal of benchmarking is to provide a standardised and objective way to assess the performance of a system or process. It helps identify areas for improvement, gauge competitiveness, and make informed decisions for optimization or comparison purposes.

B. What are the parameters used for the tests?

Each program is run as a child-process of a Python script using Popen.

i) secs:

Measurement: The time is measured using the time.time() function.

Timeframe: It represents the total time taken from before forking the child process until after the child process exits.

Inclusion: It includes the processing done by the measurement script itself and any other activities that occur on the idle isolated test machine during that time.

Scope: It encompasses all activities related to the measurement script and the test environment, not specifically limited to the child process.

ii) cpu secs:

Measurement: The time is measured using the os.wait3 function, specifically recording the user (usr) and system (sys) time for the child process.

Timeframe: It represents the actual CPU time consumed by the child process.

Inclusion: It includes only the processing time of the child process and does not take into account other activities outside the child process.

Limitations: In multiprocess programs, it may not capture all processing done by the program. In such cases, it may provide an estimate of busy time instead of the complete processing time.

iii) mem:

The script child-process maximum resident set size is recorded using os.wait3.

This is not the maximum resident set size of the process tree and may underestimate the memory use of multi process programs.

iv) gz:

We start with the source-code markup you can see, remove comments, remove duplicate whitespace characters, and then apply minimum GZip compression. The measurement is the size in bytes of that GZip compressed source-code file.

v) cpu load:

It is measured using the psutil.cpu_percent before forking the child-process and after the child-process exits. This will include processing done by the measurement script, and whatever else happens on the idle isolated test machine.

C: Running the tests and collecting data:

The following data has been measured on a quad-core 3.0GHz Intel® i5-3330® with 15.8 GiB of RAM and 2TB SATA disk drive; using Ubuntu™ 22.10 x86_64 GNU/Linux 5.19.0-29-generic.

The benchmarking programs utilised have been crowd sourced, contributed to the project "benchmarksgame-team".[2]

Versions:
Rust (1.67.0) (fc594f156 2023-01-24)
LLVM version: 15.0.6
C clang
Ubuntu clang version 15.0.2-1

source	secs	N	mem	gz	cpu secs	cpu load
binary-trees	0.02	7	11,020	751	0.00	0% 0% 0% 0%
n-body	5.63	50,000,000	11,068	1480	5.63	100% 0% 0% 0%
regex-redux #3	0.21	500,000	79,264	719	0.22	56% 9% 33% 9%
fannkuch-redux #6	3.51	12	11,036	1253	13.93	99% 99% 99% 99%

Table 1.1: Data collected for Rust[3]

ii) Data collected for C

source	secs	N	mem	gz	cpu secs	cpu load
binary-trees	0.00	7	11,192	654	0.00	0% 0% 0% 0%
n-body	5.98	50,000,000	11,392	1173	5.98	0% 0% 0% 100%
regex-redux #3	0.23	500,000	14,688	1266	0.70	78% 73% 72% 76%
fannkuch-redux #6	2.05	12	11,236	1576	8.13	99% 99% 98% 99%

Table 1.1: Data collected for C[4]

D. Analysing the obtained data

Let us begin by examining the processing time of the programs. With the exception of the fannkuch-redux program, Rust consistently exhibits lower CPU execution times compared to C. Furthermore, it is noteworthy that Rust demonstrates faster program startup times in the majority of cases.

Moving on to the comparison of main memory utilisation between Rust and C, we find that, apart from the regex-redux program, Rust requires significantly less memory for processing.

We now turn our attention to an often overlooked metric: the amount of storage a program occupies on the hard disk. In two out of four scenarios, Rust demonstrates a notable advantage by requiring significantly less storage than C. However, in the remaining two cases, C exhibits a substantial reduction in hard disk usage compared to Rust. Upon closer observation, we find that for larger tasks, C programs tend to occupy significantly more storage space. On the other hand, for smaller tasks, C performs admirably without imposing a heavy burden on the hard disk.

As for CPU load, we note that the load on each core in every case is almost similar in both cases, however, in the case of regex-redux, we notice that Rust performs much better but also occupies much more memory.

Hence, with just benchmarking tests alone, we cannot come to any conclusion. Benchmarking tests often focus on specific aspects of performance, potentially overlooking other important factors such as maintainability, readability, and ecosystem support. Additionally, benchmarks might not accurately reflect real-world workloads and can sometimes produce misleading or biassed results if not carefully designed and executed.

III. EVALUATION OF MEMORY MANAGEMENT TECHNIQUES

When evaluating memory management techniques, several factors should be considered:

i) Memory Safety: Rust's ownership system and compile-time checks provide strong guarantees of memory safety, reducing the likelihood of memory-related bugs. C, on the other hand, places the responsibility of memory safety on the programmer,

making it more prone to errors if not handled meticulously.

ii) Runtime Overhead: Rust's memory management techniques incur a certain degree of runtime overhead due to the enforcement of ownership rules and lifetime checks. However, the Rust compiler's optimization capabilities aim to minimise this overhead and achieve performance comparable to C, especially for compute-intensive tasks.

iii) Development Productivity: Rust's memory management model can sometimes require additional effort to satisfy the borrowing and ownership rules. This may lead to a steeper learning curve for developers transitioning from languages with different memory management paradigms. In contrast, C's manual memory management allows more freedom but requires careful attention and expertise to ensure memory safety.

iv) Memory Efficiency: Rust's ownership model enables efficient memory utilisation by minimising unnecessary allocations and deallocations. C's manual memory management provides developers with fine-grained control over memory usage, allowing for optimizations tailored to specific requirements.

v) Trade-offs: The choice between Rust and C for memory management depends on the specific project requirements. Rust's emphasis on safety and concurrency makes it well-suited for systems where memory safety and thread safety are critical. C's low-level control and flexibility make it preferable for scenarios that demand explicit memory management or interactions with existing C libraries.

By evaluating the memory management techniques employed by Rust and C, we can better understand their impact on performance, reliability, and developer productivity. These evaluations help in selecting the appropriate language based on the specific requirements and constraints of the project at hand.

IV: MEMORY SAFETY ANALYSIS

When comparing the memory safety analysis of Rust and C, the following conclusions can be drawn:

A. Rust's memory safety features, enforced by the compiler at compile-time, provide strong guarantees against memory-related bugs. Rust's ownership system, borrowing rules, and static analysis significantly reduce the risk of common issues like null pointer dereferences, use-after-free, and data races. Rust's memory safety analysis contributes to improved code reliability and security.[5]

B. C's memory safety heavily relies on the diligence and expertise of the developer. While defensive programming practices can help mitigate memory-related bugs, the lack of built-in memory safety mechanisms exposes C programs to a higher risk of memory-related issues if not implemented meticulously.

Overall, Rust's memory safety analysis surpasses that of C, as it provides a comprehensive and enforced approach to memory safety. By leveraging compile-time checks and ownership tracking, Rust minimises the likelihood of memory-related bugs, enhancing the overall reliability and security of systems programming code. [6]

V. DEVELOPER PRODUCTIVITY ANALYSIS

When comparing the syntax and readability of Rust and C, the following conclusions can be drawn:

A. Rust's syntax is designed to be expressive, ergonomic, and modern. It promotes code readability through its emphasis on code documentation, formatting conventions, and expressive constructs like pattern matching and closures. Rust's ownership system, although initially challenging, contributes to safer concurrency and error handling, once understood.

B. C's syntax is straightforward, familiar, and minimalistic. Its procedural nature and lack of higher-level abstractions make it easy to read and understand for developers with imperative programming experience. However, C's lack of built-in safety features and modern language constructs may lead to more verbose code and potential readability challenges in certain scenarios.

Overall, Rust's syntax and readability surpass that of C in terms of expressiveness, modernity, and safety-oriented design. Rust's focus on code documentation, expressive constructs, and strong tooling for formatting and style enforcement contributes to enhanced code readability and maintainability.

VI. SUMMARY OF FINDINGS AND ANALYSIS

Throughout this report, we conducted a qualitative and quantitative analysis of Rust and C, aiming to assess whether Rust can finally replace C. We explored various aspects, including background and significance, performance benchmarks, memory management, error handling mechanisms, security vulnerabilities, language syntax and readability, type safety and memory management, tooling and libraries support, development workflows, learning curve and community support, the Rust ecosystem, adoption in various industries, case studies of successful Rust projects, and a comparison of industry trends and language preferences. Based on our analysis, the following key findings can be summarised:

A. Performance and Memory Management

Rust's performance benchmarks often rival or surpass those of C, thanks to its low-level control and memory safety guarantees.

Rust's ownership system and borrow checker eliminate common memory-related vulnerabilities, such as use-after-free and null pointer dereferences.

B. Error Handling and Safety:

Rust's approach to error handling using Result and Option types promotes robust error management and reduces the likelihood of runtime errors.

Rust's strong type system and pattern matching capabilities catch many errors at compile-time, enhancing code reliability and safety.

C. Security and Vulnerabilities:

Rust's memory safety features significantly mitigate common security vulnerabilities like buffer overflows and data races.

C, on the other hand, is more prone to memory-related vulnerabilities due to manual memory management and weak type checking.

D. Development Productivity and Tooling:

Rust's ecosystem, led by the Cargo package manager, provides a modern and efficient development experience, simplifying dependency management and build processes.

C, being a mature language, has a vast ecosystem of tools and libraries built over the years, supporting a wide range of development requirements.

E. Industry Adoption:

Rust's adoption is growing across diverse industries such as systems programming, web development, game development, cryptography, IoT, and financial technology.

C remains prevalent in industries that prioritise low-level control, embedded systems, networking, and hardware-related development. [7]

F. Learning Curve and Community Support:

Rust has a steeper learning curve compared to C due to its ownership system and borrow checker. However, the availability of comprehensive learning resources and a supportive community contribute to a smooth learning experience.

C, with its long-standing presence and familiarity, offers a more accessible learning curve for developers already experienced in imperative languages.

Based on the analysis of these factors, it is evident that Rust has made significant strides in its ability to replace C in various domains. Rust's memory safety, performance, and modern language features have

positioned it as a strong alternative to C, particularly in areas where reliability, security, and concurrency are paramount. However, the adoption of Rust should be considered on a case-by-case basis, taking into account specific project requirements, existing codebases, performance needs, and ecosystem support.

VII. REFERENCES

[1] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053-1058.

[2] The Computer Language Benchmarks Game: Rust vs C benchmark tests. https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust.html

[3] The Computer Language Benchmarks Game: Rust - Programming Language Benchmarks. https://benchmarksgame-team.pages.debian.net/benchmarksgame/measurements/rust.html

[4] The Computer Language Benchmarks Game: Clang - Programming Language Benchmarks: https://benchmarksgame-team.pages.debian.net/benchmarksgame/measurements/clang.html

[5] Alexandrescu, A., & Stroustrup, B. (2018). Rust for C++ programmers. C++Now Conference.

[6] He, J., & Lim, J. (2019). Exploring the use of Rust for secure systems programming. Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, 635-639.

[7] Baumann, A., Felber, P., & Foucard, Q. (2021). Performance comparison of Rust and C in real-world applications. Proceedings of the 30th International Conference on Compiler Construction, 63-74.

[8] P. K. Kushwaha and M. Kumaresan, "Machine learning algorithm in healthcare system: A Review," 2021 International Conference on Technological Advancements and Innovations (ICTAI), Tashkent, Uzbekistan, 2021, pp. 478-481, doi: 10.1109/ICTAI53825.2021.9673220.

[9] P. K. Kushwaha, V. Bibhu, B. P. Lohani and D. Singh, "Review on information security, laws and ethical issues with online financial system," 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), Greater Noida, India, 2016, pp. 49-53, doi: 10.1109/ICICCS.2016.7542350.

[10] G. Gulati, B. P. Lohani and P. K. Kushwaha, “A Novel Application Of IoT In Empowering Women Safety Using GPS Tracking Module,” 2020 Research, Innovation, Knowledge Management and Technology Application for Business Sustainability (INBUSH), Greater Noida, India, 2020, pp. 131-137, doi: 10.1109/INBUSH46973.2020.9392193.

[11] D. Pareta, I. N. Verma, B. P. Lohani, P. K. Kushwaha and V. Bibhu, "IoT Enabled Smart and Efficient Musical Water Fountain," 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM), Gautam Buddha Nagar, India, 2022, pp. 369-373, doi: 10.1109/ICIPTM54933.2022.9754129.

[12] B. P. Lohani, M. Trivedi, R. J. Singh, V. Bibhu, S. Ranjan and P. K. Kushwaha, "Machine Learning Based Model for Prediction of Loan Approval," 2022 3rd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2022, pp. 465-470, doi: 10.1109/ICIEM54221.2022.9853160.

[13] V. Bibhu, A. Kumar, B. P. Lohani and P. K. Kushwaha, "Robust Secured Framework for Online Business Transactions over Public Network," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 555-560, doi: 10.1109/ICIEM51511.2021.9445380.

[14] V. Bibhu, P. K. Kushwaha and B. P. Lohani, "A review of security of the cloud computing over business with implementation," 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), Greater Noida, India, 2016, pp. 192-198, doi: 10.1109/ICICCS.2016.7542342.

[15] Amardeep Gupta and Ranjeet Kumar Rout, “ROTEE: Remora Optimization and Tunicate swarm algorithm-based Energy-Efficient cluster-based routing for EHenabled heterogeneous WSNs,” International Journal of Communication System (Q1), vol. 33, no. 6, pp. 1-23, 2022. DOI:https://doi.org/10.1002/dac.5372. (IF: 2.0)

[16] D. Gupta and R. K. Rout, "An Effective Optimization Method for Energy Efficient Clustering in EH Wireless Sensor Networks," 2021 International Conference on Technological Advancements and Innovations (ICTAI), Tashkent, Uzbekistan, 2021, pp. 699-702, doi: 10.1109/ICTAI53825.2021.9673312.

[17] S. S. N. Challapalli, P. Kaushik, S. Suman, B. D. Shivahare, V. Bibhu and A. D. Gupta, "Web Development and performance comparison of Web Development Technologies in Node.js and Python," 2021 International Conference on Technological Advancements and Innovations (ICTAI), Tashkent, Uzbekistan, 2021, pp. 303-307, doi: 10.1109/ICTAI53825.2021.9673464.

[18] Rana A, Goyal N, Ahlawat A, Jamwal S, Reddy B, Sharma S. Mechanisms involved in attenuated cardio-protective role of ischemic preconditioning in metabolic disorders. Perfusion. 2015;30(2):94-105. doi:10.1177/0267659114536760

[19] Ahlawat, A., Rana, A., Goyal, N. et al. Potential role of nitric oxide synthase isoforms in pathophysiology of neuropathic pain. Inflammopharmacol 22, 269–278 (2014). https://doi.org/10.1007/s10787-014-0213-0

[20] A. R. Yeruva, P. Choudhari, A. Shrivastava, D. Verma, S. Shaw and A. Rana, "Covid-19 Disease Detection using Chest X-Ray Images by Means of CNN," 2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS), Tashkent, Uzbekistan, 2022, pp. 625-631, doi: 10.1109/ICTACS56270.2022.9988148.

[21] Ghosh, S., Rana, A., & Kansal, V. (2020). A benchmarking framework using nonlinear manifold detection techniques for software defect prediction. International Journal of Computational Science and Engineering, 21(4), 593-614.

[22] Raghavendra, M. S., Chawla, P., & Rana, A. (2020, June). A survey of optimization algorithms for fog computing service placement. In 2020 8th international conference on reliability, infocom technologies and optimization (trends and future directions)(ICRITO) (pp. 259-262). IEEE.

[23] Gupta, S., Rana, A., & Kansal, V. (2020). Optimization in wireless sensor network using soft computing. In Proceedings of the Third International Conference on Computational Intelligence and Informatics: ICCII 2018 (pp. 801-810). Springer Singapore.
[24] Kunwar, V., Agarwal, N., Rana, A., & Pandey, J. P. (2018). Load balancing in cloud—a systematic review. Big Data Analytics: Proceedings of CSI 2015, 583-593.
[25] Chawla, P., Chana, I., & Rana, A. (2015). A novel strategy for automatic test data generation using soft computing technique. Frontiers of Computer Science, 9, 346-363.
[26] Walia, H., Rana, A., & Kansal, V. (2017, September). A Naïve Bayes Approach for working on Gurmukhi Word Sense Disambiguation. In 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 432-435). IEEE.
[27] Dash, Y., Dubey, S. K., & Rana, A. (2012). Maintainability prediction of object oriented software system by using artificial neural network approach. International Journal of Soft Computing and Engineering (IJSCE), 2(2), 420-423.
[28] Dubey, S. K., & Rana, A. (2010). A comprehensive assessment of object-oriented software systems using metrics approach. International Journal on Computer Science and Engineering, 2(8), 2726-2730.
[29] S. Gupta, A. Rana and V. Kansal, "Comparison of Heuristic techniques:A case of TSP," 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 2020, pp. 172-177, doi: 10.1109/Confluence47617.2020.9058211.
[30] Ghosh, S., Rana, A., & Kansal, V. (2018). A nonlinear manifold detection based model for software defect prediction. Procedia computer science, 132, 581-594.
[31] Chawla, P., Chana, I., & Rana, A. (2016). Cloud-based automatic test data generation framework. Journal of Computer and System Sciences, 82(5), 712-738.
[32] Bhardwaj, M., & Rana, A. (2016). Key Software Metrics and its Impact on each other for Software Development Projects. International Journal of Electrical & Computer Engineering (2088-8708), 6(1).
[33] Rana, A., & Sharma, S. (2016). Mechanism of sphingosine-1-phosphate induced cardioprotection against I/R injury in diabetic rat heart: Possible involvement of glycogen synthase kinase 3β and mitochondrial permeability transition pore. Clinical and Experimental Pharmacology and Physiology, 43(2), 166-173.
[34] G. Dubey, A. Rana and N. K. Shukla, "User reviews data analysis using opinion mining on web," 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Greater Noida, India, 2015, pp. 603-612, doi: 10.1109/ABLAZE.2015.7154934.
[35] Ghosh, S., Rana, A., Kansal, V. (2017). Predicting Defect of Software System. In: Satapathy, S., Bhateja, V., Udgata, S., Pattnaik, P. (eds) Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications . Advances in Intelligent Systems and Computing, vol 516. Springer, Singapore. https://doi.org/10.1007/978-981-10-3156-4_6
[36] Sanjay Kumar Dubey, Ajay Rana, and Yajnaseni Dash. 2012. Maintainability prediction of object-oriented software system by multilayer perceptron model. SIGSOFT Softw. Eng. Notes 37, 5 (September 2012), 1–4. https://doi.org/10.1145/2347696.2347703
[37] S. Chawla, G. Dubey and A. Rana, "Product opinion mining using sentiment analysis on smartphone reviews," 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2017, pp. 377-383, doi: 10.1109/ICRITO.2017.8342455.
[38] Dubey, S. K., Rana, A., & Sharma, A. (2012). Usability evaluation of object oriented software system using fuzzy logic approach. International Journal of Computer Applications, 43(19), 1-6.
[39] Saini, Rimmi, Sanjay Kumar Dubey, and Ajay Rana. "Analytical study of maintainability models for quality evaluation." Indian Journal of Computer Science and Engineering 2.3 (2011): 449-454.
[40] Ghosh, Soumi, Ajay Rana, and Vineet Kansal. "A statistical comparison for evaluating the effectiveness of linear and nonlinear manifold detection techniques for software defect prediction." International Journal of Advanced Intelligence Paradigms 12.3-4 (2019): 370-391.
[41] A. Singh, M. Chaudhary, A. Rana and G. Dubey, "Online Mining of data to generate association rule mining in large databases," 2011 International Conference on Recent Trends in Information Systems, Kolkata, India, 2011, pp. 126-131, doi: 10.1109/ReTIS.2011.6146853.
[42] N. Tyagi, A. Rana and V. Kansal, "Creating Elasticity with Enhanced Weighted Optimization Load Balancing Algorithm in Cloud Computing," 2019 Amity International Conference on Artificial Intelligence (AICAI), Dubai, United Arab Emirates, 2019, pp. 600-604, doi: 10.1109/AICAI.2019.8701375.
[43] Dubey, Sanjay Kumar, and Ajay Rana. "A fuzzy approach for evaluation of maintainability of object oriented software system." International Journal of Computer Applications 49.21 (2012).