QUALITY OF ENGINEERING COMPUTING SOFTWARE SYSTEMS: AN EMPIRICAL CASE-STUDY OF OPENFOAM (2011-2018)

Jens K. Carter*, Eric S. McDaniel, Warren S. Vaz, Saleh M. Alnaeli

Department of CSEPA

University of Wisconsin-Fox Valley

Menasha, Wisconsin 54952

Corresponding Author (cartj522@students.uwc.edu)

Abstract

An empirical study examines some of the source code quality aspects (e.g., usage of recursive function calls, jump statements, and parallelizability inhibitors) from software engineering perspective in a well-known engineering software system, OpenFOAM. OpenFOAM is commonly used by experts across areas of engineering and science within both industrial and academic organizations. It is written in the C/C++ and is comprised of over a million lines of code. The system was statically analyzed to detect and tabulate all recursive invocations, used jumping statements (e.g., goto and breaks), and parallelization inhibitors. The results show that number of the recursive calls, jumping statements, and parallelization inhibitors is significant and thus poses roadblocks to analyze, parallelize, and maintain the analyzed system and suggests to re-engineer it to better utilize new technologies such as multicore architecture. The study shows an increase of issues in source code from the lack of coding standards.

Keywords: engineering software, software vulnerabilities, parallelization, open source, OpenFOAM, recursion.

1. Introduction

Engineering software allows engineers to rapidly and cost-effectively model and solve complex problems that would be analytically impossible or would require laborious experimentation. As such, there exist hundreds of different software packages that continue to undergo expansion. Although most of the software used in high-level engineering is proprietary [1], there are communities that use open source software systems to analyze projects and solve problems [9-10]. These open source systems may not always take advantage of modern computational architecture. While developers of engineering systems have a deeper understanding of the domain in which the system will be used, they may lack the programming background knowledge and experience that would allow them to better utilize new hardware [2].

Here, issues are examined in a popular finite element analysis tool used by engineers and scientists, namely OpenFOAM [7]. OpenFOAM is a tool to develop numerical solvers for computational fluid dynamics (CFD) and other multi-physics engineering problems. The reason it is a focal point in this study is because it has been updated each year. This allows for a historical data that can be retrieved and analyzed with the tools developed and used in similar studies of programming efficiency and software vulnerability [2,4].

1.1.Multicore Technology and Parallelizability

One new feature of modern hardware is the use of multicore technology, which provides computers the ability to run functions in parallel. Parallelizability is a computational technique through which executions of processes are run concurrently. This technique is not inherent in the C++ language and most C++ compilers are limited in their ability to parallelize data [4]. When programmed properly, a system can utilize multicore technology to complete processes more efficiently by making full use of the system's resources. Effective use of parallelization is inhibited when source code that contains for-loops are not parallelizable. Many functions commonly used by engineers to generate their own software are not able to be parallelized include go-to and break statements. In this study, for-loops that do not contain any inhibitors to parallelizability are considered free loops [4].

1.2.Direct and Indirect Recursive Calls

A common technique found in software development is the use of direct and indirect recursion. Statements within a function that calls itself during execution time is known as direct recursion [12]. In addition, statements within a function that calls a second function only to be called back to the original function are known as indirect recursion. Software systems that invoke the use of recursion often inhibit the utilization of newer multicore architecture, which can reduce the efficiency of the software. Recursive calls may also make software systems more difficult to analyze and maintain [12]. Functions that are recursively called require the previously called function to be suspended, which allocates a frame in stack memory. As this process repeats, a substantial amount of memory could be allocated that does not solve the function's problem until its return to the nested invocation. In this study, the use of recursive techniques are measured and analyzed throughout the evolution of the systems.

1.3.Go-To and Break Statements (Jumps)

Go-to and jump statements are common amongst programs written for scientific and engineering programs [4]. These statements are involved in one-way transfers of control to separate lines of code, termination, and switch statements. These types of statements are not parallelizable in modern multicore technology and the programmers and developers using them are not utilizing their resources as efficiently as possible. In this study, jump statements are analyzed both within for-loops and throughout the entirety of each system.

1.4.Data Dependency

Data dependency is yet another problem that programmers should consider when developing engineering software. The order of statement execution within the body of a for-loop must be consistent to have the same expected outcome. Future iterations rely on the outcome of previous iterations, so running for-loops that contain data dependency concurrently will result in an incorrect outcome. Because these loops often prevent parallelization, all loop iterations need to be independent of each other [4]. In this study, a dependence analysis is done on the OpenFOAM systems to determine the distribution of data-dependent code.

1.5.Objectives

The research objectives of this study are to determine the distribution of recursive calls and jump statements throughout OpenFOAM. The first research goal is to quantify the trend of these inhibitors from 2011 to 2018. The second research goal is to compare the analysis of jump statements throughout the entire software with the distribution of jump statements that are only found within for-loops. The overarching goal of this case study is to determine if there needs to be new standards put in place to help engineers better develop the tools they rely upon in their field.

2. Related Work

This study extends work previously done towards determining which type of system is more susceptible to side effects from not optimizing function paths [6]. Side effects are found when a function either modifies a global or static variable, modifies a parameter that was passed by reference, when a function passes input/output operations, or when a function calls another function that also has side effects [13]. Among fifteen different scientific systems, the percentage of functions with side effects averaged over 50%. This study is different from the previous study in that it focuses on a single system from the engineering domain. Other previous work looked at general scientific systems rather than focusing on engineering ones [1,6]. The methodology for that study was similar in that it looked at a five-year history of eight open source scientific systems. What the previous study showed was that, in general, scientific systems are not utilizing new technologies and techniques such as parallelization as they evolve over time.

3. Data Collection

The source code of OpenFOAM was gathered online through a website that promotes the usage of open source software by being a medium in which software is displayed, downloaded, and

OpenFOAM	**Lines of code**	**Number of files**	**Number of functions**
2011	1410669	5554	16531
2012	1177835	6552	19282
2013	1466667	7751	23791
2014	1464745	7751	23735
2015	1451021	7690	23685
2016	1001355	7558	22789
2017	1420310	7625	23036
2018	1456678	7878	23708
Total	10849280	58359	176557

Table 1: The original data analysis of OpenFOAM from 2011 to 2018.

transferred from programmer to programmer. These types of websites give programmers the opportunity to debug, rewrite, and modify compilers, modules, and entire software packages written in every coding language and for any operating system. The online repositories used for this study are GitHub and Source Forge, which allowed for the download of over forty thousand files and nearly ten million lines of code [8-9].

The files containing the source code of the OpenFOAM systems were then unzipped and transformed into XML using a srcML toolkit [2-4]. The code was then analyzed by a srcQuality tool that parsed the code and recorded the number of direct and indirect recursive calls within each system. At the same time, the srcQuality tool records the number of jump statements (Goto and break) found within each system. The second analysis utilized a ParaSTAT tool that recorded the number of for-loops that contained jumps and the other inhibitors to parallelizability mentioned in section 1. These tools were the same ones used in previously mentioned studies and developed by one of the authors. This was repeated uniformly for each year in the range of this study, namely 2011-2018. Over ten million lines of code were examined containing more than one hundred and seventy thousand functions and methods. Table 1 shows the data from the analysis through time.

4. Results and Discussion

The results of the software analysis tools described in the previous section were recorded in tables. This was done for each version of OpenFOAM. The following describes the trends that have been observed.

4.1 Recursive Calls

Figure 1 represents the total number of recursive calls found within the source code of each system from 2011 to 2018. The total number of recursive calls has increased at an average rate of 3% per year since 2011.

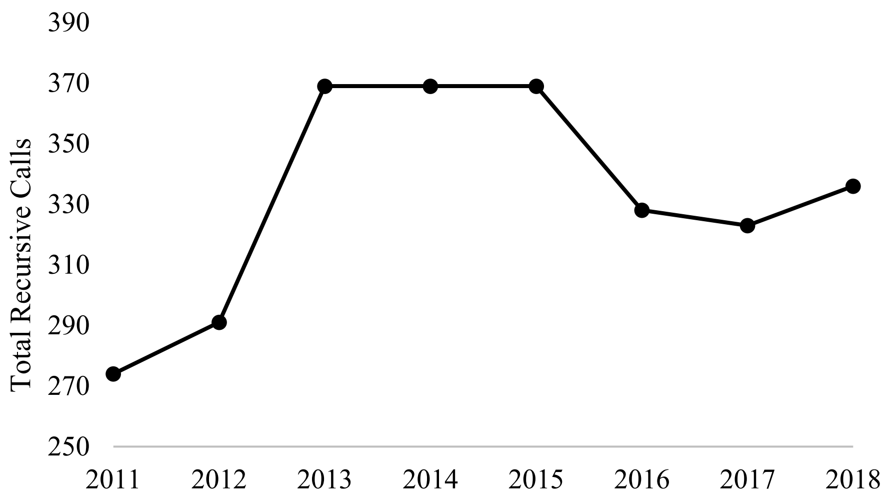

Figure 1: Represents the total number of recursive calls found in OpenFOAM from 2011 to 2018.

The total number of recursive calls remained constant from 2013 to 2015 at 369 direct and indirect recursive calls, which was also the largest amount of recursive calls found within any single year.

4.2 Jump Statements

Figure 2 represents the total number of jump statements found within the source code of each system from 2011 to 2018. The total number of jump statements has increased at an average rate of 3% per year since 2011. The total number of jump statements remained constant from 2013 to 2015 at 1376 total jump statements. The largest amount of jump statements used was found to be 1454 in 2018

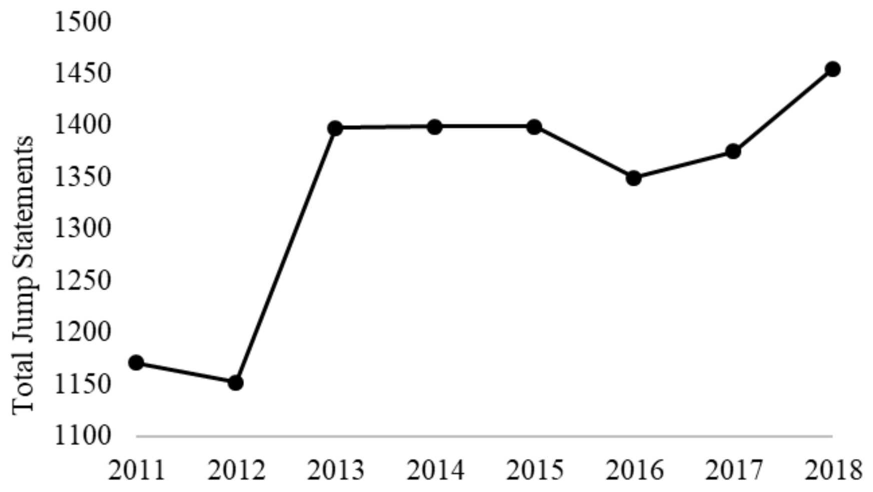

Figure 2: Represents the total number of jump statements found in OpenFOAM from 2011 to 2018.

The distribution of jump statements between Goto and Break statements is shown in Table 2. The number of Goto statements found is significantly lower and consistent from year to year, with the exception from 2013 to 2015 where 23 where found. The number of Break statements found was `significant and growing through time, following the trend of the overall jump statements found.

OpenFOAM	Goto	Break	Total Jumop Statements	Percent Change
2011	1	1170	1171	-
2012	23	1129	1152	-3.50%
2013	23	1375	1398	21.8%
2014	23	1376	1399	0.07%
2015	23	1376	1399	0.00%
2016	1	1349	1350	-1.96%
2017	1	1374	1375	1.85%
2018	1	1454	1455	5.82%

Table 2: Shows the distribution of jump statements in OpenFOAM from 2011 to 2018.

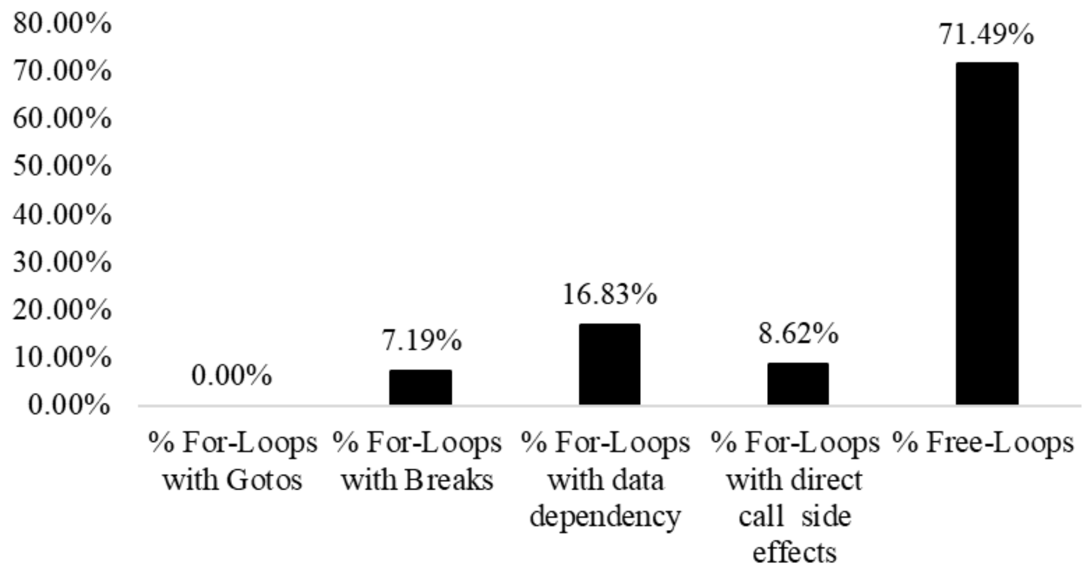

Figure 3: Shows the distribution of Goto, Break, Data-dependency, Direct recursive calls, and Free-Loops in OpenFOAM from 2011 to 2018.

4.3 Jump statements, Direct Recursive Calls and Data Dependency Within For-Loops

There different inhibitors to parallelization described were counted and tallied for each version of OpenFOAM. The average distribution was computer over 2011-2018. Figure 3 shows the average percentage of inhibitors to parallelization within for-loops since 2011. Data dependency is the most prevalent inhibitor to parallelizability at 16.83%. The percentage of Goto statements within for-loops were zero. However, the percentage of break statements averaged to 7.19% per year. The standard deviation of the percentage of break statements is only 0.17%, showing that the number of break statements varied minimally since 2011. Also shown in the same figure, is the average percentage of side effects within for-loops from direct recursive calls, which was found to be 8.62%. Lastly, the average percentage of free loops was found to be 71.49%. These percentages do not add up to 100% because the results stem from separate methods of observation.

4.4 For-Loops Containing Inhibitors

The distribution of the percentage of for-loops with at least one inhibitor compared to the total number of for-loops is shown in Figure 4. Also shown in the same figure, is the percentage of free loops. Expectedly, for any given year, the percentage of free-loops and for-loops with at least one inhibitor would add up to 100%. Between 2011 and 2018, an average of 28.40% of the for-loops had at least one inhibitor. While the overall number of functions with side effects has increased over time as shown in Figure 4, the distribution of for-loops with at least one inhibitor has decreased by an average of 1.16% annually since 2011. Meanwhile, the percent of free loops in the system increased by an average of 0.53% annually since 2011.

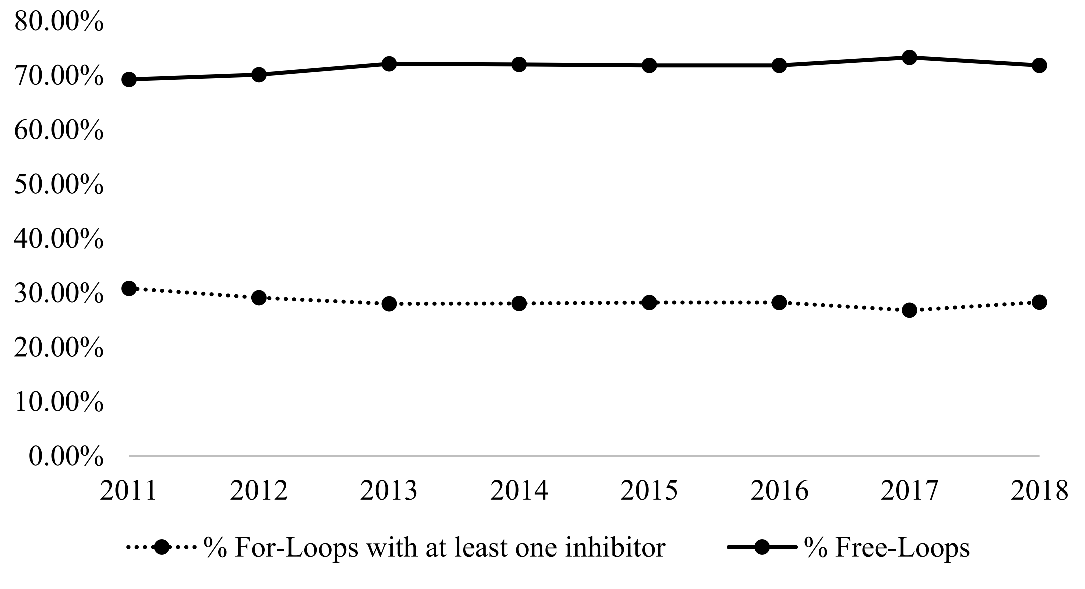

Figure 4: Shows the distribution of For-Loops with at least one inhibitor along with the distribution of Free-Loops found in OpenFOAM from 2011 to 2018.

4.5 Functions Containing For-Loops with Side Effects

In Figure 5, the number of functions with side effects is represented through time. This number has also increased over the six-year period. The average number of functions with side effects in OpenFOAM over eight years is 8,273 affected functions. The number of functions affected increased by an average of 405 functions per year, a 4.9% increase annually. When analyzing the total number of functions with side effects, 37.25% of the total number of functions are affected.

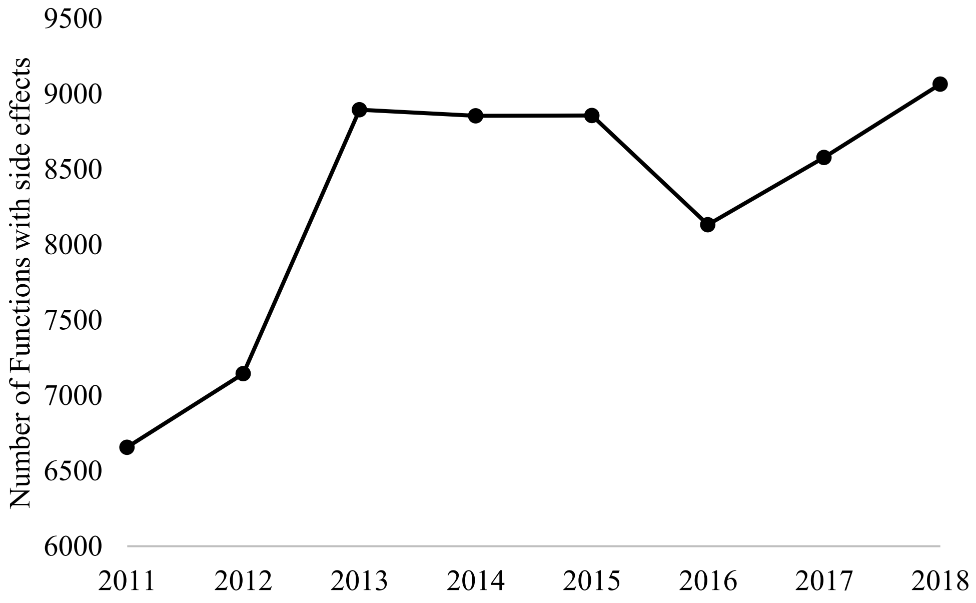

Figure 5: Represents the number of functions containing For-Loops with side effects found in OpenFOAM from 2011 to 2018.

It is clearly seen in Figures 1, 2 and 5 that the usage of jump statements and recursive calls has increased within the for-loops as well as the overall system. The distribution of for-loops and free loops in Figure 4 shows that there is room for improvement in the source code of OpenFOAM. Although more than half of the for-loops are parallelizable (free loops), there is still more than a quarter of the system not being utilized. It may be concluded that the OpenFOAM source code could be improved to better utilize multicore technology and increase overall performance by addressing the software quality issued described in this study [2]. It would be worth the time for engineers to work more closely with software engineers in the development of their software packages to improve how efficiently resources are utilized. The engineering community may suffer negative impacts when they rely on inefficient and poorly designed software [7]. Despite changes with the intent to expand and improve OpenFOAM, this open source system may be discredited as a reliable alternative when compared to proprietary software [1].

5. Conclusion

This study examined the typical problems engineering software encounters when attempting to utilize multicore technology. The open-source multi-physics engineering tool OpenFOAM was studied over an eight-year period using tools developed by one of the authors. Over ten million lines of code were examined contained more than 170,000 functions and methods.

In this study it was found that the number of recursive calls has increased in the overall system by an average rate of 3% per year since 2011. Additionally, the study found that direct recursive calls are prevalent within for-loops at 8.62%. It was also found that the number of jump statements has increased at an average rate of 3% per year since 2011. Within for-loops, the average percentage of jump statements was found to be 7.19%. It was also found that the number of functions with side effects has increased by 4.9% annually since 2011. Between 2011 and 2018, an average of 28.40% of the for-loops had at least one inhibitor. The study found that data dependency is the largest inhibitor to parallelization at 16.83%. This study shows that attention should be addressed systematic overall inhibitors if parallelization is to occur efficiently, which leads to better use of modern computational architecture. Moreover, this study shows that the breadth of knowledge engineers have when developing their software is limited. The results show that these issues are trending upward in OpenFOAM, which may disadvantage users. OpenFOAM developers are not considering well-known techniques in computer science to utilize multicore technology.

One goals of this study is to present an abstract overview of issues in the engineering systems domain from a software engineering standpoint. Engineers developing their software need to be aware of potential pitfalls with unsafe commands and parallelizability inhibitors to better utilize their software. The coding styles of engineers writing their systems must be improved, otherwise they will continue to generate more challenges for their software by using recursive calls and jump statements. A set of standards could be put in place to assist engineers when developing engineering software packages. This would make engineering programs current with modern hardware. Less time and resources would then be spent reengineering the software, while more time would be invested in developing new systems and solving problems in the field.

Acknowledgement

The authors are grateful to the University of Wisconsin-Fox Valley for supporting this study. The authors are also grateful to Melissa Sarnowski, a former UW-Fox Valley student, for assistance with the study and insightful comments pertaining to this paper.

References

[1] J.E. Hannay, C. MacLeod, J. Singer, H.P. Langtangen, D. Pfahl and G.Wilson, "How do scientists develop and use scientific software?", *Software Engineering for Computational Science and Engineering*, 2009, pp 1-8.

[2] S.M. Alnaeli, J.I. Maletic, and M.L. Collard, "An empirical examination of the prevalence of inhibitors to the parallelizability of open source software systems", *Empirical software engineering,* 2016, vol 21, pp 1272-1301.

[3] M. Young, "The technical writer's handbook", Mill Valley, CA: University Science, 1989.

[4] S.M. Alnaeli, M. Sarnowski, C. Meier, M. Hall, “Empirically Identifying the Challenges in Parallelizing Scientific Software Systems”, *25th International Conference on Software Engineering and Data Engineering,* 2016*,* pp 1-6.

[5] S.M. Alnaeli, M. Sarnowski, “Examining the Prevalence and the Historical Trends of Indirect Function Calls in Open Source Systems: A Case Study”. *The Midwest Instruction and Computing Symposium,* 2016, pp 1-15.

[6] S.M. Alnaeli, M. Sarnowski, S. Aman, K. Yelamarthi, A. Abdelgawad, H. Jiang, “On the Evolution of Mobile Computing Software Systems and C/C++ Vulnerable Code.” *Ubiquitous Computing, Electronics & Mobile Communication Conference,* 2016, pp 1-7.

[7] S.M. Alnaeli, M. Sarnowski, “Historical Trends of the Multicore Adaptability and Parallelizability of Scientific Software Systems”

[8] V. Starkiovicius, R. Ciegis, A.Bugajev, “On efficiency analysis of the OpenFOAM-Based Parallel Solver for Simulation of Heat Transfer in and Around the Electric Power Cables” *Informatica.* 2016, vol. 27, issue 1, p161-178. 18p.

[9] Official OpenFOAM Repository. 2017, GitHub repository, https://github.com/OpenFOAM.

[10] OpenFOAM repository, 2017, Source Forge repository,

https://sourceforge.net/projects/openfoamplus/?source=directory.

[11] W.A. Bhat, S.M.K. Quadri, “Open Source Code Doesn’t Always Help: Case of File System Development”, *Trends in Information Management,* 2011, vol. 7, issue 2, pp 135-144.

[12] S.M. Alnaeli, M. Sarnowski, Z. Blasczyk, “On the Usage of Recursive Function Calls In C/C++ General Purpose Software Systems.” *Journal of Computing Sciences in Colleges,* 2017, Vol. 33, Issue 1, pp. 51-59.

[13] S. M. Alnaeli, A. D. A. Taha, and T. Timm, "On the Prevalence of Function Side Effects in General Purpose Open Source Software Systems." 2016. IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), 2016, pp. 141-148.