Using Clause Slicing as Program Robustness Measurement Technique

Mohammad M. Abdallah and Ayman M. Abdalla

Faculty of Science and IT
Al-Zaytoonah University of Jordan
Amman, Jordan
m.abdallah@zuj.edu.jo, ayman@zuj.edu.jo

Abstract—The program slicing technique is an abstracting technique that focuses on the program code. Clause Slicing is the type of program slicing that focuses only on the code clauses, which allows the quality assurance to measure program robustness by measuring every code clause against the programing language standards. The proposed model gives a new way for measuring the robustness quality factor using program clause slicing.

Keywords—Program slicing, Clause slicing, Robustness, Quality, Measurement.

I. INTRODUCTION

Program slicing was introduced in 1979 by Weiser [1] as "a method used for abstracting from computer programs." Regarding the formula; in Program P, the program slice has criteria which are represented as <s, v>, where s is the statement number and v is the variable. With respect to slicing criteria, the slice includes only those statements of P needed to capture the behavior of v at s [2].

Clause slicing was introduced in 2012 by Abdallah [3] as a static slicing technique. Clause slicing is concentrating on the code syntax of the program. It is also a syntax preservation technique where the code does not change after slicing.

Clause slicing considers most of the code syntax words as potential slicing criteria, which makes it more useful in testing and measuring program quality.

This research paper is focused on the technique of how the clause slicing technique can be used as a quality measurement technique to measure the software robustness quality. Clause slicing as static technique focuses on each code sentence and sub-sentence. It can be very useful in testing the code ability to run robustly and to predict how the code could fail or crash by highlighting the weak points.

This paper will have the following structure: Section II will address the work related to this research. Section III will address the proposed robustness measurement model. Conclusion and future directions will be given in Section IV.

II. RELATED WORK

In this section, the main program slicing techniques are addressed and compared in terms of use to help determine the proper slicing technique for measuring program robustness, which is also defined and discussed in the following sections.

A. Program Slicing

Program slicing was first introduced in 1979 by Weiser as an abstraction for the program without introducing any changes to the code syntax [1]. Static slicing means that the code syntax will be reserved after slicing. In other words, all possible executions of the program are taken into account [4, 5]. The term "static slicing" has been used frequently in recent years.

Program static slicing has a slice criterion (C) which is the Variable of interest (V) and its code line number (N) and presented as C (V, N). In Program slicing in general, only the code that is directly or indirectly related to the slicing criteria is captured and abstracted as a Program Slice. In further researches, many types of static slicing and other slicing types have been developed and introduced such as backward and forward slicing [6], decomposition slicing [7], clause slicing [8, 9], conditional [10] and Decomposition slicing. One of the static slicing techniques is the analysis method based only on the name of the variable without considering the variable line number or variable input values. This is the same as merging both the forward and backward slicing techniques [7]. A few more static slicing techniques were described in [11]. There are other types of slicing, such as dynamic slicing [12], which introduce the variable value to the slicing criteria and can be presented as (V, N, Input). Other slicing techniques have been developed by [4, 13].

Program slicing has been applied to improve program debugging [14], Software Maintenance [7], Regression testing [15, 16], Software robustness [3, 8, 17-19] and Software quality [20]. It has been applied to different programming languages such as C, C++, and Java [9, 21, 22].

According to [11, 23], the slicing program divides the program into several slices. These slices are characterized based on their dependency, where each slice contains all the statements that affect or are affected by the variable in the slicing criterion. This criterion consists of the effective variable and the statement it contains. This is based on the slice containing all statements that affect the value of the variable.

Static slicing can be executable or non-executable [24]. An executable slice means the code produced after the slicing operation (the slice) can be compiled and run as a program.

Weiser [1, 6, 14, 25] introduced program slicing which became known later as Executable Backward Static Slicing. It

is Executable because the slice produced is an executable program which can be debugged and run as a usual program. Backward Slicing is computed by gathering statements and control predicts by way of a backward traversal of the program starting at the slicing criteria [24]. Backward slicing contains the statements of the program which affect the criteria slice, and it answers the question "what program components might affect a selected computation?" [5]

Another form of static slicing is Forward Slicing. Forward Slicing performs traversal of data and control dependence edges in the forward direction and answers the question "what program components might be affected by a selected computation?" [5].

A Forward slice captures the effect of its slicing criteria on the rest of the program, and it is considered a kind of flow effect analysis [4, 26]. It contains the set of statements and control predicts that were affected by the computation of the slicing criterion. The Slicing criteria are the same as in backward slicing (V, L) [21, 24, 27-30].

Forward Slicing usually does not produce an executable slice, unlike backward slicing, because the challenge addressed by Forward Slicing is defining the semantics captured by a forward slice [24, 31, 32].

Decomposition slicing is a slice used to decompose the program into different components. It is the union of certain slices taken at certain line numbers on a given variable [6, 7]. Decomposition slicing has two parts: The slice, which is the slice criteria, and the complement. The slice "captures all relevant computations involving a given variable" [7], where a decomposition slice depends on the variable name only and does not depend on statement number. The complement is the rest of the program code; it also can be considered as a slice that corresponds to the rest of the slicing criteria [7].

Clause slicing was introduced in [3, 33] as a slicing technique interested in a part of the statement that may affect the rest of the slice. The main purpose of Clause slicing is to enhance the software robustness measurement of C programs.

Clause slicing is a special type of static slicing techniques; it is a reserved syntax technique. Clause Slicing has the same types of static slicing; backward, forward and decomposition. In this paper, only the forward clause slice will be discussed.

A Clause is defined as the minimum piece of code that can be sliced [33, 34]. Some clauses are not sliceable, such as #*include* and *break*, so they are called the un-sliceable clauses. Clauses are different form the code statements and the code line. The Clauses can be a part of the code line or statement. For example, the *printf* statement can be divided into three clauses. There are some rules on how to slice the code into clauses, as what was introduced in [8].

Program slicing is widely used for many purposes: debugging [6], maintenance [7], testing [16, 35], detecting dead code [36], measuring program robustness [17, 18, 37] and quality [38-40] and many other applications [13, 38, 41, 42]. Therefore, researchers have tried applying different ideas of using program slicing and advanced tools. Tools of program slicing were developed to slice different programming languages [43]. This research paper is focused on C language slicing, so it only lists some slicing tools for C language: CSurf, frama-C, and Wisconsin Program-Slicing.

CodeSurfer [44] is a program-understanding tool that makes a manual review of code easier and faster. It also has CodeSonar, which finds bugs and generates reports automatically. Another code analysis tool is called Frama-C [45]. Frama-C is used only for programs written in the C programming language. It supports static slicing techniques; Forward and backward slicing. It also provides dependency analysis. There is also a program slicing tool called Wisconsin Program-Slicing tool [46]. It can do a Forward Slicing, backward slicing, and chopping. It consists of a package for building and manipulating control-flow graphs and program dependence graphs. The srcML [46, 47] program is a command line application for the conversion source code to srcML. It is an interface for the exploration, analysis, and manipulation of source code in this form, and the conversion of srcML back to source code. The current parsing technologies support C/C++, C#, and Java [21, 22, 41].

B. Program Robustness

Critical programs must be robust to avoid the problems that could be caused by failures [48]. The C language standards were introduced to avoid code misinterpretation, misuse, or misunderstanding. The IEEE presented the ISO/IEC 9899:1999 standard [49], which was later used by MISRA to produce MISRA C1 and C2. This, in turn, led to Jones producing "The New C Standard: An Economic and Cultural Commentary" [48]. The LDRA company uses MISRA C rules, in addition to 800 rules it created, to assess programs. Other C standards such as "C programming language coding guideline" [50] are less frequently used.

Measuring the application of a language standard to a program is one technique of program robustness measurement. Several techniques were used to measure program robustness. Software measurement could mean estimating the cost, determining the quality, or predicting the maintainability [51]. Arup and Daniel [52] presented features, such as portability, to evaluate some existing benchmarks of Unix systems. As a result, they built a hierarchy-structured benchmark to identify robustness issues that were not detected before. Behdis and Shokat [53] introduced a theoretical foundation for robust matrices that reduce the uncertainty in distributed system. Arne et al. [54] used some robustness criteria such as input date rate and CPU clock rate to create multi-dimensional robustness matrices, and then use them to measure the robustness of the system.

A robustness hierarchy [17] is a relative scale to find the robustness characteristics that need to be added to programs. It is a technique used to build a robust program. The hierarchy starts with a non-robust program as the first step and then adds robust features before reaching a robust program in the highest level of the hierarchy.

III. THE PROPOSED MODEL

This section addresses the Robustness Grid. The Robustness Grid measures software Robustness using program clause slicing in addition to the program language standards. The Robustness Grid needs to examine the features of programming languages in order to produce a relative scale for functions, methods, and the entire program. The Robustness Grid will show the Robustness Degree in details for a selected program. The Robustness Grid Measurement is the process by which relative numbers are assigned to the Robustness Degree of a C program in a way that describes them according to

Programming language standards and their language features weight, which is specified using the clause slicing. Figure 1 shows the Robustness Grid building process.

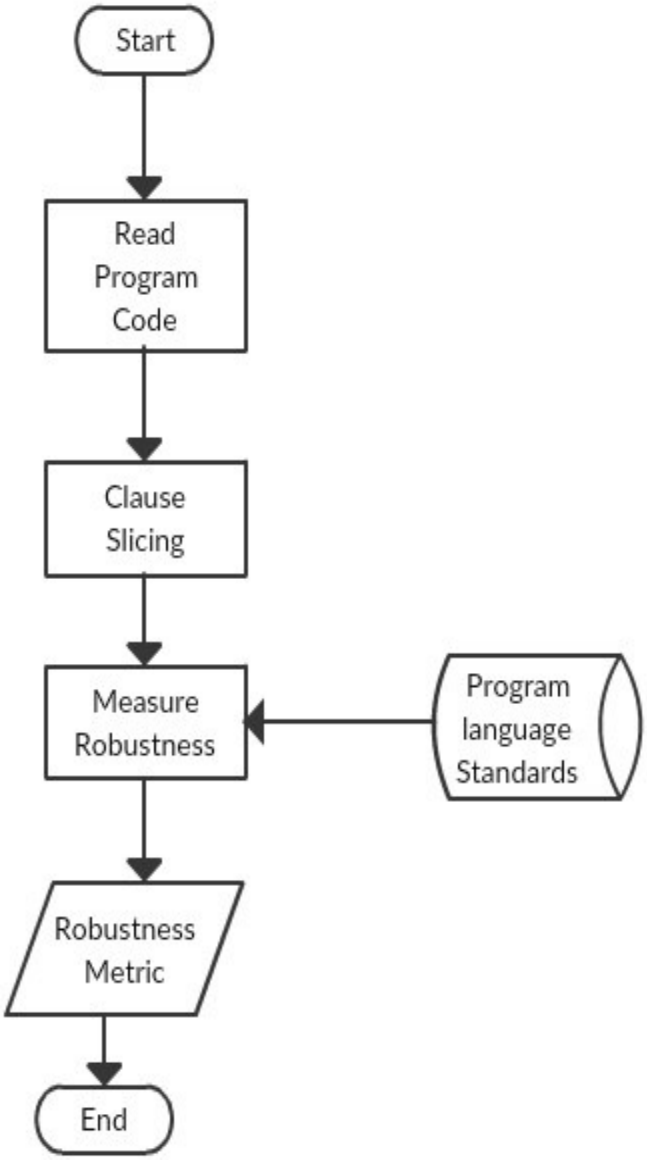

Figure 1. Robustness Grid Process

As seen in Figure 1, the Robustness Grid starts with reading and running the code to make sure that it runs correctly and has no compiling problems. The Robustness Grid is a quality measurement, which means that the code must compile and run correctly before being measured.

In the second step, the code is sliced using Clause Slicing techniques. A tool that can be used is called Clauser [33, 55]. The results of this process will be a group of clauses. These clauses will be measured using the programming language standards in step three. In addition, the size of each slice will determine the weight of it. The result will be how much the clauses are following the language standards and how much effect they have on the program robustness factor.

IV. CONCLUSION AND FUTURE WORK

The proposed model is quite useful and can be handy to measure the Robustness factor degree of any program written in any programming language. However, the proposed model is still immature and needs more work to convert it into an automated tool that can be relied on. In addition, considerable work is needed to determine how to choose the programming language standards, and which rules of these standards must be chosen.

The proposed model can be upgraded and extended to add more features such as: choosing which slicing techniques to be applied, which programming language to be chosen and which standards of this language to be selected.

REFERENCES

[1] M. Weiser, "Program slices: formal, psychological, and practical investigations of an automatic program abstraction method," PhD, The University of Michigan, Michigan, 1979.

[2] D. Binkley and K. Gallagher, "Program Slicing," in Advances in Computers. vol. Volume 43, V. Z. Marvin, Ed., ed: Elsevier, 1996, pp. 1-50.

[3] M. Abdallah, "A Weighted Grid for Measuring Program Robustness," PhD, Computer Science, Durham University, 2012.

[4] X. Baowen, Q. Ju, Z. Xiaofang, W. Zhongqiang, and C. Lin, "A brief survey of program slicing," vol. 30, pp. 1-36, 2005.

[5] K. Gallagher and D. Binkley, "Program slicing," in Frontiers of Software Maintenance, 2008. FoSM 2008., 2008, pp. 58-67.

[6] M. Weiser, "Program Slicing," IEEE Transactions on Software Engineering, vol. 10, pp. 352-357, 1984.

[7] K. Gallagher and J. R. Lyle, "Using program slicing in software maintenance," Software Engineering, IEEE Transactions on, vol. 17, pp. 751-761, 1991.

[8] M. Abdallah, "A Weighted Grid for Measuring Program Robustness," Doctor of Philosophy, Computer Science Durham University, http://etheses.dur.ac.uk, 2012.

[9] M. Abdallah and H. Tamimi, "Clauser : Clause Slicing Tool for C Programs," International Journal of Software Engineering and Its Applications, vol. 10, pp. 49-56, 03/31 2016.

[10] G. Canfora, A. Cimitile, and A. De Lucia, "Conditioned program slicing," Information and Software Technology, vol. 40, pp. 595-607, 1998.

[11] A. Ngah and S. A. Selamat, "A Brief Survey of Program Slicing," in International Symposium on Research in Innovation and Sustainability 2014 (ISoRIS '14), Malacca, Malaysia, 2014, pp. 1467-1470.

[12] B. Korel and J. Laski, "Dynamic program slicing," Information Processing Letters, vol. 29, pp. 155-163, 1988.

[13] A. Ngah and S. A. Selamat, "A BRIEF SURVEY OF PROGRAM SLICING," Science International, vol. 26, 2014.

[14] M. Weiser, "Programmers use slices when debugging," Communications of the ACM, vol. 25, pp. 446-452, 1982.

[15] N. Amir and G. Keith, "Regression test selection by exclusion using decomposition slicing," presented at the Proceedings of the doctoral symposium for ESEC/FSE on Doctoral symposium, Amsterdam, The Netherlands, 2009.

[16] A. Ngah, M. Munro, and M. Abdallah, "An Overview of Regression Testing," Journal of Telecommunication, Electronic and Computer Engineering (JTEC), vol. 9, pp. 45-49, 2017.

[17] M. Abdallah, M. Munro, and K. Gallagher, "Certifying software robustness using program slicing," in 2010 IEEE International Conference on Software Maintenance, 2010, pp. 1-2.

[18] A. Abdalla, M. Abdallah, and M. Salah, "ABrief PROGRAM ROBUSTNESS SURVEY," International Journal of Software Engineering & Applications, vol. 8, pp. 1-10, 2017.

[19] K. Gallagher and N. Fulton, "Using Program Slicing to Estimate Software Robustness," in Proceedings of the International Systems Software Assurance Conference, 1999.

[20] M. Abdallah and M. Alrifaee, "Towards a new framework of program quality measurement based on programming language standards," 2018, vol. 7, p. 3, 2018-03-08 2018.

[21] B. Alokush, M. Abdallah, M. Alrifaee, and M. Salah, "A Proposed Java Static Slicing Approach," Indonesian Journal of Electrical Engineering and Computer Science, vol. 11, pp. 308-317, 2018.

[22] M. Abdallah, B. Alokush, M. Alrefaee, M. Salah, R. Bader, and K. Awad, "JavaBST: Java backward slicing tool," in 2017 8th International Conference on Information Technology (ICIT), 2017, pp. 614-618.

[23] J. T. Lalchandani and R. Mall, "Regression testing based-on slicing of component-based software architectures," presented at the Proceedings of the 1st India software engineering conference, Hyderabad, India, 2008.

[24] F. Tip, "A survay of Program Slicing Techniques," JOurnal of Programming Languages, vol. 3, pp. 121-189, 1995.

[25] M. Weiser, "Program slicing," presented at the Proceedings of the 5th international conference on Software engineering, San Diego, California, United States, 1981.

[26] B. Sue, "Computing ripple effect for software maintenance," Journal of Software Maintenance, vol. 13, p. 263, 2001.

[27] A. d. Lucia, "Program Slicing: Methods and Applications," presented at the IEEE International Workshop on Source Code Analysis and Manipulation, 2001.

[28] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," ACM Transaction of Program Language Systms, vol. 12, pp. 26–60, 1990.

[29] M. Harman and R. Hierons, "An Overview of program slicing," software focus, vol. 2, pp. 85-92, 2001.

[30] H. Mumtaz, M. Alshayeb, S. Mahmood, and M. Niazi, "An empirical study to improve software security through the application of code refactoring," Information and Software Technology, vol. 96, pp. 112-125, 2018/04/01/ 2018.

[31] D. Binkley, S. Danicic, T. Gyimóthy, and M. Harman, "Theoretical foundations of dynamic program slicing," Theoretical Computer Science, vol. 360, pp. 23 – 41, 2006.

[32] S. Mitra, D. Kim, and M. Fong, "Program Slicing," 2007.

[33] M. Abdallah and H. A. Tamimi, "Clauser: Clause Slicing Tool for C Programs," International Journal of Software Engineering and Its Applications, vol. 10, pp. 49-56, 2016.

[34] M. Abdallah, "A Weighted Grid for Measuring Program Robustness," Durham University, 2012.

[35] A. Ngah, M. Munro, Z. Abdullah, M. A. Jalil, and M. Abdallah, "Regression Test Selection Model: A Comparison between ReTSE and Pythia," TELKOMNIKA (Telecommunication Computing Electronics and Control), vol. 17, 2018.

[36] N. AlAbwaini, A. Aldaàje, T. Jaber, M. Abdallah, and A. Tamimi, "Using Program Slicing to Detect the Dead Code," in 2018 8th International Conference on Computer Science and Information Technology (CSIT), 2018, pp. 230-233.

[37] M. Abdallah, M. Munro, and K. Gallagher, "A Static Robustness Grid Using MISRA C2 Language Rules," presented at the The Sixth International Conference on Software Engineering Advances, Barcelona, Spain, 2011.

[38] M. Abdallah and M. M. Al-rifaee, "Towards a new framework of program quality measurement based on programming language standards," International Journal of Engineering & Technology, vol. 7, pp. 1-3, 2018.

[39] K. S. Patnaik and P. Jha, "Proposed Metrics for Process Capability Analysis in Improving Software Quality: An Empirical Study," International Journal of Software Engineering and Technology (IJSET), vol. 1, pp. 152-164, 2016.

[40] M. Abdallah and M. Al-rifaee, "Java Standards: A Comparative Study," International Journal of Computer Science and Software Engineering, vol. 6, p. 146, 2017.

[41] A. Ngah and S. A. Selamat, "Using Object to Slice Java Program," Journal of Engineering and Applied Sciences, vol. 13, pp. 1320-1325, 2018.

[42] S. A. Selamat and A. Ngah, "Slicing for Java Program: A Preliminary Study," Journal of Telecommunication, Electronic and Computer Engineering (JTEC), vol. 9, pp. 147-151, 2017.

[43] T. Hoffner, "Evaluation and comparison of program slicing tools," Department of Computer and Information Science, Linkping University, Sweden1995.

[44] GrammaTech. (2009, 12/1/2018). CodeSurfer. Available: http://www.grammatech.com/products/codesurfer/overview.html

[45] P. Baudin, F. Bobot, R. Bonichon, L. Correnson, P. Cuoq, Z. Dargaye, et al., "Frama-C," Frama-C 16 - Sulfur ed: Informations légales et droit de diffusion, 2007.

[46] M.-C. Lee and T. Chang, "Software Measurement and Software Metrics in Software Quality," International Journal of Software Engineering and Its Applications, vol. 7, pp. 15-34, 2013.

[47] C. D. Newman, T. Sage, M. L. Collard, H. W. Alomari, and J. I. Maletic, "srcSlice: A Tool for Efficient Static Forward Slicing," in 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 2016, pp. 621-624.

[48] D. M. Jones, The New C Standard: A Cultural and Economic Commentary, 1st edition ed.: Addison-Wesley Professional, 2003.

[49] International Standard ISO/IEC 9899, 1999.

[50] E. Laroche. (1998, 22/12). C programming language coding guidelines.

[51] N. E. Fenton and S. L. Pfleeger, Software Metrics, A Rigorous and Practical Approach, 2 ed.: PWS Publishing Company, 1997.

[52] A. Mukherjee and D. P. Siewiorek, "Measuring Software Dependability by Robustness Benchmarking," IEEE Transactions of Software Engineering, vol. 23, pp. 94-148, 1994.

[53] B. Eslamnour and S. Ali, "Measuring robustness of computing systems," Simulation Modelling Practice and Theory, vol. 17, pp. 1457-1467, 2009.

[54] H. Arne, R. Razvan, and E. Rolf, "Methods for multi-dimensional robustness optimization in complex embedded systems," presented at the Proceedings of the 7th ACM & IEEE international conference on Embedded software, Salzburg, Austria, 2007.

[55] K. Awad, M. Abdallah, A. Tamimi, A. Ngah, and H. Tamimi, "A Proposed Forward Clause Slicing Application," Indonesian Journal of Electrical Engineering and Computer Science, vol. 13, 2019.