Slicing for Java Program: A Preliminary Study

Siti Aminah Selamat and Amir Ngah
School of Informatics and Applied Mathematics,
Universiti Malaysia Terengganu, 21030 Kuala Nerus, Terengganu, Malaysia.
ami.selamat@gmail.com

***Abstract*—Program slicing is a technique that proposed to help in understanding the program code. After several decades, the technique has been derived into several other techniques and proposed to be applied in many fields such as debugging, program comprehension, software measurement, testing and maintenance. The application of program slicing sometimes specifies for certain programming language such as C and Java. This paper will discuss existing program slicing techniques that were proposed focusing on the Java programming language.**

***Index Terms*— Dependency Representation; Java; Program Analysis; Program Slicing.**

I. INTRODUCTION

Program slicing is a task of breaking down a large program into smaller components called Slice. The slice consists of lines of code that performs the behavior in the program. The first technique was proposed by Weiser [1], [2] known as the static backward slicing. After that, many papers were proposed to cater to the need of analyzing code in debugging, program comprehension, testing, and maintenance.

This paper focuses on the existing slicing technique proposed for Java program. Java is a programming language that has been widely used and has become a leading programming language [3]. Program slicing is a technique that can be used to aid debugging, program comprehension, testing, fault localization, maintenance, and security. Using program slicing, the debugging process can be simplified by reducing the program into related lines of codes only. Program slicing can also be used in the maintenance process to help the software evolve and up dated. In addition, program slicing can be applied in testing newly developed programs or modified programs to make sure there is no bug or any line of code that can trigger the production of bugs.

Section II briefly discusses the concept of program slicing along with the most popular techniques, which are Static slicing, Backward and Forward slicing, and Dynamic slicing. Section III explains the program slicing proposed for the Java program. Section IV discusses the program representation for the Java program and the application of program slicing is discussed in Section V.

II. PROGRAM SLICING

The program slicing algorithm was previously only proposed for a sequential program [5],[6]. However, the technique was modified in order to slice the object oriented program. The object oriented programs that often used in the research of program slicing techniques are C++ and Java programming language.

The most popular technique in program slicing is static slicing, which was first introduced by Weiser [2] and dynamic slicing, which was proposed by Korel and Laski [7]. The program is sliced based on data dependency and control dependency in the program. If one statement execution affects the data of the slicing criterion, then the statement is included in the slice, similar to control dependency, if the execution of one statement can affect the execution of the slicing criterion, that statement is included in the slice.

A. Static Slicing

Static slicing considers all possible executions of the slicing criterion [8], and includes all the statements in the program that might affect the value at some point of interest into the slice. Danicic, Harman, and Sivagurunathan [9] proposed a parallel algorithm for the static slicing technique. Parallel algorithm can be used in several ways, such as to construct slices with multiple slicing criteria and to perform simultaneous slicing previously proposed by using the Program Dependence Graph (PDG) approach [9]. Static slicing for concurrent programs using the new program representation approach was proposed by Zhao, Cheng, and Ushijima [10]. The new program representation is called System Dependence Net (SDN), which extends the previous program representation. The net consists of a group of procedure dependence nets with each representing the main procedure. There are few research papers that have proposed combination of two existing slicing techniques. Static slicing is one of the techniques that has quite a few combinations with other techniques, for example forward slicing, backward slicing, and dynamic slicing. These proposed combination techniques will be discussed in the following sub-sections.

B. Backward and Forward Slicing

Slicing can be traversed forward or backward starting from the statement where the slicing criterion is located. This traversal is also used in the intermediate representation graph. Forward slicing includes all the statements that will be affected by the slicing criterion and the backward slicing includes all the statements that will affect the slicing criterion. Binkley [11] states that the slicing criterion proposed by Weiser is sliced in a backward manner and therefore the Weiser technique is known as the backward static slicing [12]. The forward slicing technique was proposed by Bergeretti and Carre [13] that was later proposed to be combined with static slicing [14]–[16].

C. Dynamic slicing

Dynamic slicing includes the slice with the statement in the program that is affected during the execution of the program using the input. This results in smaller slices. The dynamic slice can be divided into two categories: executable slice and non-executable slice [17]. The executable dynamic slice includes the statement needed for the execution, and the non-

executable slice only includes the statement that might affect the variable of interest and it cannot be executed.

III. Program Slicing for Java

There are many program slicing techniques that have been proposed for Java program. One of the most popular techniques is static slicing. In 1996, static slicing was proposed for Java program [18]. Since Java program is an object oriented program, the object oriented features such as inheritance, polymorphism, and dynamic binding need to be taken into account during the slicing process. Kovács, Magyar, and Gyimóthy [18] introduced the new representation for the polymorphic call that helps to reduce additional vertices during the polymorphic handling. The tool for slicing sequential Java program was proposed 11 years later to slice Java program in the Soot framework [5]. Java is also a concurrent programming that runs concurrently instead of sequentially, thus, the approach to slice a concurrent Java program using static slicing was proposed [19]. In order to slice the program, concurrent control flow graph (CCFG) and concurrent program dependence graph (CPDG) were presented to represent the concurrent program. Zhao and Li [20] also had the same basic idea to represent dependency in the concurrent Java program, but the approach proposed by Zhao and Li used the class dependence graph and method dependence graph in order to construct the concurrent program dependence graph (CPDG). Ranganath and Hatcliff [21] proposed the slicing concurrent Java program using the slicing tool Indus and Kaveri which is a program slicing plugin for eclipse.

Dynamic slicing is another technique that has also gained much attention in the slicing field. The slicing technique to slice a distributed Java program was proposed by [22]. The representation for the distributed program is constructed as follows. The edges of the dependency representation graph are marked when the dependency arises and unmarked when the dependency ceases. By using this approach, the dynamic slicing is performed on the program and this technique is called distributed dynamic slicing [22]. Bytecode is a compiled Java program. Wang and Roychoudhury [23] proposed dynamic slicing to slice the Java program. Instead of a program statement, the Java program is sliced using the compact bytecode traces that provide flexibility in tracing/not tracing certain bytecodes. The dynamic slicing is used to traverse the compacted bytecode traces to capture the control and data dependence in the Java program. Wang and Roychoudhury [23] also extended their research in dynamic slicing to perform relevant slicing. Another research on dynamic slicing is presented in [24], which proposed a slicing process that does not require accessing the source code during slicing. The idea is to produce an instrumented virtual machine for Java program. The technique is capable of handling an advanced aspect of the Java environment such as exception handling, multithreaded execution, and execution of native machine code linked with the Java classes. Raphnash and Bidyadhar [14] proposed forward static program slicing for a Java program that can be used to eliminate redundancy and repeated codes in a Java program. Szegedi, Gergely, and Berzedez [25] proposed the paper that verifies the concept of union slice on Java program, comparing the result with a corresponding static slice which shows that the union slice is precise enough.

Meanwhile in 2013, another slicing technique is proposed for slicing a Java program called hierarchical slicing. The technique decomposes a Java program into different components; packages, classes, methods, and statements that are affected when the program is modified. This technique is used to test the modified Java program by using these components to derive the new test suite for testing. The technique is used to reduce the test cases in regression testing [26] and to measure cohesion [27] in the Java program.

IV. Program Representation for Java Program

Many of the slicing techniques use the dependency graph to slice the program. The dependency graph consists of nodes or vertices and visualizes the dependency in the program using the edge. The most popular program intermediate representations are Program Dependence Graph (PDG) and System Dependence Graph (SDG). SDG for Java was proposed by [28] and known as the Java System Dependence Graph (JSysDG). The plus point for JSysDG is that it produces a more accurate graph by enabling static analysis to be performed on the graph. JSysDG is also able to represent classes, methods and packages, abstract methods/classes and interface, individual object and single inheritance from the class hierarchy.

Another SDG for Java intermediate representation ia the Java System Dependence Graph (JSDG) [29]. The paper proposed an intermediate representation of SDG for an object oriented program and an aspect oriented program. The SDG was then used as an input to compute the slice of the Java program with respect to the slicing criterion.

Researcher [30] proposed a static analyzer for Java bytecode called JavaPDG. The static analyzer JavaPDG can be used to produce various types of dependence representation such as a system dependence graph, procedure dependence graph, control flow dependence graph, and call graph. JavaPDG is also capable of performing both intra- and inter-procedural analysis. The analyzer has a graphical viewer to browse and analyze the various graphs and a convenient JSON based serialization format.

V. Program Slicing Application

A. Debugging

Finding and removing bugs that cause the program to produce incorrect and unexpected results is called debugging. Bugs in software programs refer to errors. Some bugs can be easily found, but there are also bugs that act dormant, are difficult to be found, and only arise in the near future when the system hits the limits. In more serious cases, the bug can cause the system to freeze or crash. This problem might lead to scrambled or loss data. Software program consists of hundreds or thousands of lines of codes. Hence, performing debugging manually on thousands of line of codes will consume a lot of time, money, and human resources. This might increase the cost of maintaining or developing the software. To prevent the problem from happening, researchers have proposed a number of solutions. One of the proposed solutions is by performing slicing on the large software program to break the program into smaller components, so that the debugging will take a shorter time. Weiser in [31] stated that a programmer mentally uses slices during debugging and has debugging as the main application of program slicing [32].

Eranki and Moudgalya [33] proposed program slicing not

only to help students to understand the Java program, but also to elevate the debugging skill in a programming course. From the testing run, the analysis shows that program slicing indeed helps to increase the understanding and debugging skill in programming.

B. Program Comprehension

Software program needs to be understood before the program code can be modified or manipulated. The person in charge of code manipulation needs to understand how the original program works and the existing constraints in the program. After that, the desired modifications are identified and applied in the program. Failure to understand the code or program behavior can lead to data loss during the manipulation process.

Lillack, Johannes, and Eisenecker [34] proposed program slicing for understanding a software generator. Software generator is used to deploy software systems. Since the deployed software has to be maintained, the software generator used to deploy the system also needs to be maintained to follow the current technology [34]. However, the code used in a software generator is difficult to understand for it to be maintained and improved in order to replace the old technology with a new one, therefore, program slicing is proposed as a technique to understand the software generator.

One of the latest approaches to help program comprehension using program slicing was proposed by [35] who used a slicing tree to slice the program. Another research [33] used program slicing to help novice learners to have a better understanding while learning the programming course. A total of 160 non-computer science students who have a basic computer literacy is selected as samples to test the effectiveness of program slicing in helping students to understand the program code. The students are divided into two workshop groups, A and B. Each workshop consists of post-test and individual assignment for each tutorial with the duration of three hours. Both workshop groups watched Java oral tutorial, but only the experimental group used the slicing technique to solve the assignment. Another group with a total of 80 students were picked randomly to form two classroom groups with 40 students each. The classroom groups were divided into a control group and an experimental group. The control group had a one-hour Java lecture followed by a one-hour Java practice lab session, while the experimental group had a one-hour Java lecture followed by a one-hour Java practice lab using program slicing. The results of the tests show that the performance of the students in the workshop's experimental group is 85% compared to the control group with 63%; as for the classroom experimental group, the result is 75% compared to 60% for the classroom control group. This shows that program slicing can assist the students in understanding and learning the program codes.

C. Testing

Before software can be deployed, it needs to be tested to check if the software meets the requirement and is bug free. Software maintenance activity such as adding new functionality, fixing software defects or adapting the system to changes in its environment might cause a bug that can make the system behave in an undesirable way [36]. Thus, the program is tested to eliminate any possible threats in the system. Testing is an important activity in software engineering,

Chebaro, Kosmatov, Giorgetti, and Julliand [37] proposed the application of program slicing to enhance the verification technique that combines the static and dynamic analyses. The static analysis is used to report a possible runtime error in which some of the reports might be a false alarm and the dynamic analysis is used to accept or reject the alarm using test generation. A previous work by [38] used value analysis to report alarm of possible runtime error and structural test generation to confirm or reject the alarm. This method, however, has a drawback, which can time out before all the reported alarms are confirmed (accepted or rejected). In order to overcome the drawback, [39] improved the technique by applying program slicing to reduce the source code before test generation and further improved the technique by developing a theory on alarm dependencies and used it to determine a better synergy of the techniques [37].

Regression testing is necessary when a new component is added to the system or when a modification done to the existing component affects another part of the system [26]. Since regression testing is an expensive activity, Panda and Mohapatra [26] proposed the application of hierarchical slicing to reduce the test while at the same time reducing the time and cost of retesting.

D. Fault Localization

Spectrum-based fault localization technique mainly utilizes testing coverage information to calculate the suspiciousness of each program element to find the faulty element. However, this technique does not fully consider the dependencies between program elements. Thus, the capacity for efficient fault localization is limited. Wen [40] proposed the implementation of program slicing into a fault localization technique called program slicing spectrum-based software fault localization (PSS-SFL). The technique consists of two steps; the first is to analyze the dependencies between program elements and delete the elements that have no dependencies with faulty elements to improve the precision of locating the fault, and the second is to build the program slice spectrum model to define the suspiciousness metric results. This technique is also more efficiently than the previous technique [41] whereby the latter can locate the fault in a multi-faults program efficiently.

Another technique proposed for fault localization is the forward slicing spectrum. The approach was proposed by Surendran and Samuel [42]. The proposed approach is expected to resolve some issues in standalone fault localization techniques such as program slicing and program spectrum based method. Examples of the problem solved include the issue related to the size of the program code, difficulties faced during the retrieval of the system feature and function, etc. [42]. In a program which contains several modules, forward slicing spectrum provides a way to determine the relevant information and an overview of the dependency between the program modules. Thus, any part of the program that is affected by the modification and integration of the new component is easily traceable.

E. Maintenance

One of the applications of program slicing is on maintenance activity. Maintenance is an expensive process in terms of cost, time, and human resources. This might be because of the many processes involved during maintenance such as program understanding, re-engineering, and testing. Firstly, the software maintainer needs to understand the code before the change can be made to the system. Re-engineering

activity then follows to make changes to the system, and lastly the system needs to be analyzed if the changes made to the system will affect any other part of the system. Gallagher and Lyle [4] proposed the decomposition program slicing to aid the maintenance process. The program is decomposed into components of the behavior of the program. Thus, maintenance can be performed on the smaller components of the program.

After the program has been maintained and changed, the program needs to be analyzed in order to understand the potential risk that might arise from the changes made to the system. This activity is called change impact analysis. Software change impact analysis is defined as a process to discover any possible effect to a particular system from a software change [43]. Acharya and Robinson [36] proposed an implementation of static slicing for assisting the change impact analysis for an industrial software system. They found that when implementing static slicing to the small program, the technique is able to produce an impact analysis result quickly and efficiently. However, when the technique is applied on a large system, it suffers from performance and accuracy issues in producing the impact analysis result. To overcome the problem, Acharya and Robinson proposed *Imp*, which is a static change analysis framework for a large evolving software system that contains over a million lines of codes. They also claimed to be the first to identify and address the challenges faced in designing the static impact analyzer which is the time and accuracy tradeoff.

Another technique for change impact analysis was proposed by [44] called the HSMImpact. HSMImpact implements the hierarchical slicing technique originally proposed by Li et al. [45]. Sun et al. [44] found that the previous change impact analysis technique for Java program focuses more on the method level without considering other granularity levels. Thus, Sun et al. proposed a new change impact analysis technique that has different granularity levels starting from the package level to the statement level. HSMImpact consists of three (3) steps which are the definition of hierarchical change sets at different granularity levels, promotion of change impact analysis based on the hierarchical slicing model, and computation of hierarchical impact set (HIS) from the package level to the statement level. Sun et al. also performed preliminary studies to demonstrate the effectiveness of HSMImpact.

Software evolves after changes have been made to the system and the evolution of the software can be tracked back by mining the software history. However, previous mining processes were manual and time consuming. Therefore, [46] proposed history slicing to aid the tracking process. The software program will be sliced based on the slicing criterion's set of line of codes and the slice will consist of all equivalent lines of codes in all the past revisions of the software project in which the line of code of interest was modified. The technique proposed also automated which reduce the amount of relevant information of the slicing criterion and the time it would take for computation.

Program slicing is also applied to estimate the maintenance effort before the maintenance is started. This process is important since the maintenance process has typically been a complex and costly process. By using forward decomposition static slice as proposed by [47], the variable in all the files in the system is recorded and each version of the system has its own dictionary. The system dictionaries are compared between the two versions and the changes recorded are modeled at the behavioral level. The change in the system is recorded and used to predict the next maintenance effort. Alomari et al. [47] uses the GNU Linux Kernel with over nine hundred (900) versions with 17 years of history as a case study. Since Linux is an open-source system, the maintenance effort estimation used for closed-source system cannot be applied directly because the maintenance effort data are not present which prevents the validation process. The model proposed for open-source maintenance estimation effort by Alomari et al. consists of five (5) phases. Firstly, the measures that are theoretically related to and can indirectly represent the maintenance effort are identified; then, the maintenance data is extracted. The third phase is to validate the correlation between the dependent and independent variables. The fourth phase is building the effort-prediction approach by using a multiple linear regression analysis and the last phase is to predict the maintenance effort based on the model built in phase four (4).

F. Security

The advanced technology allows the application and website to be accessed using the mobile application. Mobile devices such as smartphones and tablets are more convenient to be carried around instead of laptops and this indeed helps to ease everyday tasks such as making a transaction. However, this technology has a drawback in terms of security. The application that is downloaded and installed in the mobile device is used by attackers to spread malicious software (malware) that has the potential of damaging a mobile device ecosystem [48]. Thus, [48] proposed the Static Android Analysis Framework (SAAF) which is a technique to detect malicious apps in an automated way. The SAAF analyzes Smali code which is a disassembled version of DEX (Dalvik Executable). Dalvik is an Android's Java Virtual Machine Implementation that has been discontinued and replaced by Android Runtime (ART) that also used the .dex format file. The technique is used to analyze more than 140,000 applications and only has one failure during the evaluation phase.

VI. CONCLUSION

Program slicing is one of the analysis techniques that has currently received much attention. Many slicing techniques have been proposed to simplify software activity such as debugging, program comprehension, testing, maintenance, and software measurement. Program slicing has also been proposed for Java programming language and used widely in many current applications. This paper has listed a few existing slicing techniques and some of the proposed techniques are for slicing a Java program. Even though many program slicing has been proposed for a Java program, the implementation of slicing in the real world can still be numbered. The tools for slicing real world programs are limited and constrained. Thus, the tools that can be used in a real world program need to be developed to implement the slicing technique to assist software process activities.

ACKNOWLEDGMENT

This research is sponsored by the Ministry of Education, Malaysia Government under Research Acculturation Collaborative Grant (RACE), Vot No. 56032.

REFERENCES

[1] M. Weiser, "Program slicing," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 4, pp. 352–357, 1984.

[2] M. Weiser, "Program slicing," in *Proc. 5th Int. Conf. Softw. Eng.*, 1981, pp. 439–449.

[3] G. L. Taboada, S. Ramos, R. R. Expósito, J. Touriño, and R. Doallo, "Java in the high performance computing arena: research, practice and experience," *Sci. Comput. Program.*, vol. 78, no. 5, pp. 425–444, 2013.

[4] K. B. Gallagher and J. R. Lyle, "Using program slicing in software maintenance," *IEEE Trans. Softw. Eng. (TSE '91)*, vol. 17, no. 8, pp. 751–761, 1991.

[5] Devaraj, "A static slicing tool for sequential java programs", M.S. thesis, Fac. of Eng., Indian Inst. of Sci., Bangalore, 2007.

[6] J. Krinke, "Advanced slicing of sequential and concurrent programs," in *IEEE Int. Conf. Softw. Maintenance, ICSM*, 2004, pp. 464–468.

[7] Korel and J. Laski, "Dynamic program slicing," *Inf. Process. Lett.*, vol. 29, no. 3, pp. 155–163, 1988.

[8] Korel and J. Rilling, "Dynamic program slicing methods," *Inf. Softw. Technol.*, vol. 40, no. 11–12, pp. 647–659, 1998.

[9] S. Danicic, M. Harman, and Y. Sivagurunathan, "A parallel algorithm for static program slicing," *Inf. Process. Lett.*, vol. 56, no. 6, pp. 307–313, 1995.

[10] J. Zhao, J. Cheng, and K. Ushijima, "Static slicing of concurrent object-oriented programs," in *Proceedings of 20th International Computer Software and Applications Conference*, 1996, pp. 312–320.

[11] Binkley and K. B. Gallagher, "Program slicing," *Adv. in Computes*, vol. 43, pp. 1-52, 1996.

[12] Ngah and Selamat, S. A. "A brief survey of program slicing," *Sci. Int.*, vol. 26, no. 4, pp. 1467-1470, 2014.

[13] J.-F. Bergeretti and B. A. Carré, "Information-flow and data-flow analysis of while-programs," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 37–61, 1985.

[14] R. Raphnash and E. Bidyadhar, "Forward static program slicing", B. of Tech. thesis, Comp. Sci. and Eng. Dept., National Inst. of Technology Rourkela, India, 2010.

[15] H. W. Alomari, M. L. Collard, and J. I. Maletic, "A very efficient and scalable forward static slicing approach," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2012, pp. 425–434.

[16] H. W. Alomari, M. L. Collard, J. I. Maletic, N. Alhindawi, and O. Meqdadi, "srcSlice: very efficient and scalable forward static slicing," *J. Softw. Evol. Process*, vol. 26, no. 11, pp. 931–961, 2014.

[17] B. Korel and J. Rilling, "Dynamic program slicing methods," *Inf. Softw. Technol.*, vol. 40, no. 11, pp. 647–659, 1998.

[18] G. Kovács, F. Magyar, and T. Gyimóthy, "Static slicing of java programs," Technical Report TR-96-108, József Attila University, Hungary, December, 1996.

[19] Z. Q. Chen and B. W. Xu, "Slicing concurrent Java programs," *Acm Sigplan Not.*, vol. 36, no. 4, pp. 41–47, 2001.

[20] J. Zhao and B. Li, "Dependence-based representation for concurrent Java programs and its application to slicing," in *Proc. Int. Symp. Futur. Softw. Technol.*, 2004, pp. 105- 112.

[21] V. P. Ranganath and J. Hatcliff, "Slicing concurrent java programs using Indus and Kaveri," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, no. 5–6, pp. 489–504, 2007.

[22] P. Mohapatra, R. Kumar, R. Mall, D. S. Kumar, and M. Bhasin, "Distributed dynamic slicing of Java programs," *J. Syst. Softw.*, vol. 79, no. 12, pp. 1661–1678, 2006.

[23] T. Wang and A. Roychoudhury, "Dynamic slicing on Java bytecode traces," *ACM Trans. Program. Lang. Syst.*, vol. 30, no. 2, pp. 1–49, 2008.

[24] Szegedi and T. Gyimóthy, "Dynamic slicing of java bytecode programs," in *Proc. - Fifth IEEE Int. Work. Source Code Anal. Manip. SCAM 2005*, 2005, pp. 35–44.

[25] Szegedi, T. Gergely, Á. Beszédes, T. Gyimóthy, and G. Tóth, "Verifying the concept of union slices on java programs," in *Proc. Eur. Conf. Softw. Maint. Reengineering, CSMR*, 2007, pp. 233–242.

[26] S. Panda and D. P. Mohapatra, "Application of hierarchical slicing to regression test selection of Java programs," in *Workshop on Advanced Model Based Software Engineering (WAMBSE) of 6th India Software Engineering Conference (ISEC) 2013*, vol. 11, no. 2, pp. 3–20, 2013.

[27] S. Panda and D. P. Mohapatra, "ACCo: a novel approach to measure cohesion using hierarchical slicing of Java programs," *Innov. Syst. Softw. Eng.*, vol. 11, no. 4, pp. 243–260, 2015.

[28] N. Walkinshaw, M. Roper, and M. Wood, "The Java system dependence graph," in *Proc. - 3rd IEEE Int. Work. Source Code Anal. Manip. SCAM 2003*, vol. 0, pp. 55–64, 2003.

[29] S. K. Behera, "Slicing of object-oriented and aspect-oriented programs," M.S thesis, Comp. Sci. and Eng. Dept., National Inst. of Technology Rourkela, India, 2014.

[30] Shu, B. Sun, T. A. D. Henderson, and A. Podgurski, "JavaPDG: a new platform for program dependence analysis," in *Proc. - IEEE 6th Int. Conf. Softw. Testing, Verif. Validation, ICST 2013*, 2013, pp. 408–415.

[31] M. Weiser, "Programmers use slices when debugging," *Commun. ACM*, vol. 25, no. 7, pp. 446–452, 1982.

[32] Tip, "A survey of program slicing techniques," *J. Program. Lang.*, vol. 5399, no. 3, pp. 1–65, 1995.

[33] K. L. N. Eranki and K. M. Moudgalya, "Program slicing technique: a novel approach to improve programming skills in novice learners," in *Proceedings of the 17th Annual Conference on Information Technology Education*, 2016, pp. 160–165.

[34] M. Lillack, M. Johannes, and U. W. Eisenecker, "Program slicing to understand software generators," in *Proceedings of the 5th International Workshop on Feature-Oriented Software Development*, 2013, pp. 41–48.

[35] E. Hosnieh and H. Haga, "A novel approach to program comprehension process using slicing techniques," *J. Comput.*, vol. 11, no. 5, pp. 353–365, 2016.

[36] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, 2011, pp. 746-755.

[37] O. Chebaro and N. Kosmatov, "Program slicing enhances a verification technique combining static and dynamic analysis," in *Proceedings of the 27th Anuual ACM Symposium on Applied Computing (SAC '12)*, 2012, pp. 1284–1291.

[38] O. Chebaro, N. Kosmatov, A. Giorgetti, and J. Julliand, "Combining static analysis and test generation for c program debugging," in *International Conference on Tests and Proofs*, 2010, pp. 94–100.

[39] O. Chebaro, N. Kosmatov, A. Giorgetti, and J. Julliand, "The SANTE tool: value analysis, program slicing and test generation for c program debugging," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6706 LNCS, pp. 78–83, 2011.

[40] W. Wen, "Software fault localization based on program slicing spectrum," in *34th Int. Conf. Softw. Eng.(ICSE '12)*, 2012, pp. 1511–1514.

[41] W. Wen, B. Li, X. Sun, and J. Li, "Program slicing spectrum-based software fault localization*," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE '11)*, 2011, pp. 213–218.

[42] Surendran and P. Samuel, "Fault localization using forward slicing spectrum," in *Proceedings of the 2013 Research in Adaptive and Convergent Systems*. ACM, 2013, pp. 397–398.

[43] S. A. Bohner, "Extending software change impact analysis into COTS components," in *Proceedings. 27th Annual NASA Goddard/ IEEE Software Engineering Workshop (SEW'02)*, 2002, pp. 175–182.

[44] X. Sun, B. Li, C. Tao, and S. Zhang, "HSM-based change impact analysis of object-oriented java programs," *Chinese J. Electron.*, vol. 20, no. 2, pp. 247–251, 2011.

[45] Li, X. Fan, J. Pang, and J. Zhao, "Model for slicing java programs hierarchically," *J. Comput. Sci. Tech.(Jcst)*, vol. 19, no. 6, pp. 848–858, 2004.

[46] F. Servant and J. A. Jones, "History slicing," in *26th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE 2011)*, pp. 452–455, 2011.

[47] W. Alomari, M. L. Collard, and J. I. Maletic, "A slice-based estimation approach for maintenance effort," in *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, 2014, pp. 81–90.

[48] Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth, "Slicing droids: program slicing for smali code," in *28th Annu. ACM Symp. Appl. Comput.(SAC '13)*, 2013, pp. 1844–1851.