Article 3

Review on Program Slicing Techniques towards Program Comprehension Application

Rozita Kadar, Naemah Abdul Wahab, Jamal Othman
Faculty of Computer and Mathematical Sciences,
Universiti Teknologi MARA Pulau Pinang Branch, Malaysia

Abstract
Presently, the software system has grown in size. One of the main challenges faced by programmers is to keep up with thousand or million lines of source code that needs to be read and understood. The source code is an essential resource for programmers to become familiar with the software system. Program comprehension is important in software engineering activities before performing maintenance tasks. One of the techniques that can assist the programmers in comprehending software system is known as program slicing. Program slicing is the process of extracting parts of source code programs by tracing the programs' control and data flow related to some data item. In this paper, we conduct the literature review on program slicing in order to explore the direction of this method by focusing on various slicing techniques to provide a novel idea in program comprehension application.

Keywords: *Program Slicing, Program Comprehension, Software Maintenance, Source Code*

Introduction

Nowadays, most of the software developers have improve their way in developing simple source code representation to reduce the complexity of the system. This paradigm becomes widely applied in the enhancement of large-scale application where, the size of software system become huge and often exceeds a hundred or million lines of code. The main idea behind the usage of source code fragment is due to the extensive reuse of the existing system into a new system. We look into the role of program slicing technique in this paper to support the program comprehension activities prior to performing the maintenance tasks.

Program comprehension is one of the major problem in software maintenance phase (Maletic & Kagdi, 2008; Sasirekha &Hemalatha, 2011). The study of program comprehension is important in order to understand the problem domain. In recent time, software system grows in size causing additional program comprehension activities, as programmers have to face the complexity during maintenance phase. As a result of this problem, cost and time become a major constraints of this activity (Koushik & Selvarani, 2012; Roongruangsuwan & Daengdej, 2010). Norman and Vasilecas in (Lahtinen, Järvinen, & Melakoski-Vistbacka, 2007; Normantas & Vasilecas, 2013) stated that 41.8% of the total effort spent on maintenance phase.

Although, much research has been done directed to the problem of program comprehension but the studying of program comprehension remains incomplete and it should be continued in order to produce the best strategies to improve it (Maletic & Kagdi, 2008). Most researchers still have yet to discuss in great length on methods to facilitate software engineers in understanding a program.

One way to assist programmers to improve their program comprehension activity is by applying program analysis using the slicing technique. Program slicing provides mechanism to analyze and understand the program behavior for further restructuring and refinement (Koushik & Selvarani, 2012). It plays an important role in program comprehension, since it allows

programmers to focus on the relevant portions of program (Barros, da Cruz, Henriques, & Pinto, 2011; Zhang, Zheng, Huang, & Qi, 2011). Moreover, previous works found that program slicing has unique importance in addressing the issues of cost and time, and can be helpful in producing effective cost and time (Koushik & Selvarani, 2012; Saleem, Hussain, Ismail, & Mohsin, 2009). Program slicing is one of the techniques in program analysis that evaluates the program by acquiring smaller fragments of code, therefore, this application will increases program understanding. The technique is to find all statements in a program that directly or indirectly influence the value of certain variable at some point in a program. The next section discusses in details on the process of program slicing technique.

The organizations of the paper are as follows: the next section introduces the concepts of program slicing followed by the third section where, we review the previous work by comparing the techniques in this area. Finally, the last section will be the conclusion of the paper.

The Concepts of Program Slicing

The goal of program slicing technique is to eliminate some part of program statements that are unimportant, leaving only the program codes that are significance for the programmer to evaluate. There are various aspects to be considered in slicing technique that are listed in (Sasirekha & Hemalatha, 2011). In order to identify the relevant parts of programs, user must specify the *slicing criterion*, which indicates the program characteristics concerned by the programmer.

The discovery of slicing technique came from the work of Weiser(Weiser, 1982), which proposes a program understanding aid. The paper (Weiser, 1982) defined this technique as a process of breaking up any subset of the program and at the same time maintains the original program. The slicing process required the slicing criterion, a pair $c = (s, V)$, where s denotes a statement at a certain point in the program while V is represents a subset of the program's variables. The slicing variable is dependent on the variables specified in the criteria or all variables. The slicing point is the point of interest that can be placed either before or after a particular statement.

The basic idea of slicing technique is derived from two approaches, which are static slicing and dynamic slicing. A static slicing may contain statements that have no influence on the value of the variables of the interest for the particular execution whilst a dynamic slicing takes the input supplied to the program during its execution. In other words, it preserves the effect of the program for a fixed input. For example, Figure 1(a) illustrates the original program while in Figure 1(b), it shows the example of static slicing with respect to slicing criterion, $c = (8, a)$.The slicing criterion, $c = (8, a)$ means the "8" is the number of the line statement and "a" is the set of variables to be observed at statement "8". As compared to Figure 1(c) which depicts the result of dynamic slicing with respect to slicing criterion $c = (a=-2, 8, a)$. Dynamic slicing criterion, $c = (a=-2, 8, a)$, emphasizes on the input value "-2" to the observed variable "a". The result in Figure 1(c) shows that the slicing program getting smaller compared to the result in Figure 1(b). The result proves that dynamic slicing can give more contribution as compared to static slicing in the aspect of program size.

```
1.  sum=0;                  1.  sum=0;              1.
2.  cin>>a;                 2.  cin>>a;             2.  cin>>a;
3.  cin>>b;                 3.                      3.
4.  if (a>0)                4.  if (a>0)            4.
5.  sum = sum+a;            5.  sum = sum+a;        5.
6.  if (b>0)                6.                      6.
7.  sum = sum+b;            7.                      7.
8.  cout<<a;                8.  cout<<a;            8.  cout<<a;
9.  cout<<b;                9.                      9.
10. cout<<sum;              10.                     10.

        (a)                         (b)                     (c)
```

Figure 1 : (a) Example of source code program (b)Static slicing with respect to slicing criterion, c=(8, a), (c) Dynamic slicing with regard to slicing criterion, c=(a= -2,8,a)

The slicing method is classified into two directions, which are forward and backward. The forward direction requires tracing of the data and control dependences in forward direction. Usually, the result of this direction is used for modification activity in maintenance tasks. Forward is suitable for improving program comprehension because it is considered as the top-down approach in program comprehension strategies. The backward slicing direction is suitable for locating and tracing bugs. The output of this slicing direction is in the form of slices statements of a program, which has some effect on the slicing criterion.

Instead of viewing program slices statements in textual-base, program-slicing technique can be improved through visualization approach. The technique of program slicing can be enhanced by transforming a program slices into graphical-based. Essentially, program slicing technique is extracting parts of computer program by analyzing data and control flow dependence related to some data. Program Dependence Graph (PDG) is one of the graphical representations that can be used to represent data and to symbolize the control dependence of a program. In PDG, the vertices represent program statements; and the edges correspond to data and control dependences between them. These indicate a partial ordering on the statements of the program. For example, Figure 2 illustrates the PDG of the example program from Figure 1. Furthermore, this figure shows the vertices that represent the slices of program during static program slicing execution process took place.

Another factor to be considered of slicing technique is the application of slicing. Although in this paper we focus on program comprehension but slicing technique is also applicable in various areas of software engineering activities such as debugging, re-engineering, testing, model checking, verification, program segmentation and many more. In the next section, we explore the techniques of program slicing and the application of the technique in the field of software engineering.

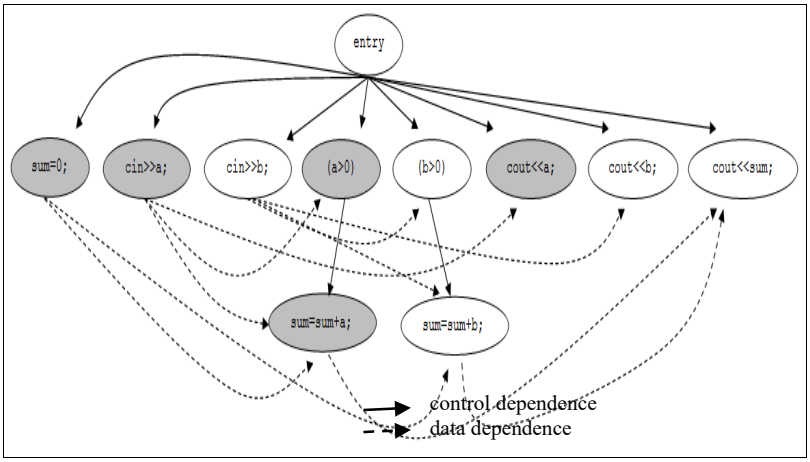

Figure 2:The PDG of the example program from
Figure 1(a) and the grey vertices represent the slices of statement from Figure 1(b)

Program Slicing Techniques and its Application

Understanding an application is important in a successful evolution of software system and program slicing is able to help programmers in comprehension phase for program maintenance purposes. This section will discuss various program slicing techniques that have been proposed in literature including static, dynamic, simultaneous, quasi and conditioned slicing as well as its application.

A static slicing is constructed by deleting those parts of the program that are irrelevant to the values stored in the chosen set of variables at the chosen point. The point of interest usually is identified by annotating the program with line numbers, which identify every primitive statement and each branch node. By exploring and understanding the slice of source code, it allows us to find the bugs faster than the original. The dynamic slicing takes the input supplied to the program during implementation and the slice contains only the statement that caused the failure during the specific execution of interest. Dynamic slicing uses dynamic analysis to identify all and only the statements that affect the variables of interest on the particular anomalous execution trace(Korel & Rilling, 1998).

The simultaneous technique combines the use of a set of test cases with program slicing. The method is called simultaneous dynamic program slicing because it extends and simultaneously applies to a set of test cases the dynamic slicing technique(Korel & Laski, 1990)which produces executable slices that are correct on a single input. Quasi-static slicing is a hybrid of static and dynamic slicing. Static slicing examines during compile time, using no information about the input variables of the program. On the other hand, dynamic slicing analyses the code by giving input to the program. It is constructed at run time with respect to a particular input. A conditioned slicing consists of a subset of program statements, which preserves the behavior of the original program relating to a slicing criterion for any set of program executions. The set of initial states of the program that characterize these executions is specified in terms of a first order logic formula on the input. Conditioned slicing allows a better decomposition of the program giving human readers the possibility to evaluate the code fragments concerning different perspectives.

The idea behind all approaches to program slicing is to produce the simplest program possible that maintains the meaning of the original program in connection with this slicing criterion. The conditioned criterion is the most general of these, subsuming both static and dynamic

criteria as the special cases. The conditioned criterion consists of a set of variables, a program point of interest and a condition. There are some drawbacks between static and dynamic slicing methods. Static slicing needs more space, more resources and will perform every possible execution of the program. Conversely, dynamic slicing needs less space and is specific to a program execution. Dynamic slices are smaller than static slice(Binkley et al., 2005). For complete program understanding, one execution of the program is not enough. Hence, the Quasi static slicing method was first introduced by Venkatesh(Roongruangsuwan & Daengdej, 2010). In Quasi slicing, the value of some variables are fixed and the program is analyzed while the value of other variables vary. The behaviour of the original program is not changed with respect to the slicing criterion. Slicing criteria includes the set of variables of interest and initial conditions, therefore, quasi slicing is called as Conditioned slicing(Binkley et al., 2005). This is an efficient method for program comprehension.

Existing Works on Program Slicing Technique

This section explains the existing works on program slicing in supporting program comprehension. The discussion is based on selected papers focusing on the strengths and weaknesses of the work as well as the approaches, slicing criterion and the output representation.

The first work proposed by Servant & Jones (2013) uses the history slicing method to discover the historical change events to the source code. The proposed work is useful for understanding source code growth, however, it consumes much time when applying the proposed work. The output representative is in a set of lines of interest contains all their snap-shots in all the past revisions in which they were modified. The work provides a tool namely Chronos. Santelices et al. (2013) uses the quantitative slicing approach focusing on novice users to quantify the relevance of each statement in a slice. This method help user focus their attention on the part of slices that matter the most. The study is able to assess the potential impact of changing location for a given score to quantifies the lines but the method is ambiguous and just by estimation. The output representation is in a form of slices and source code snippets. They also develop a tool called SensA.

Jain & Poonia (2013) combined the static and dynamic slicing method to reduce the time taken in slicing process. The output representation is in slices and source code snippets. The tool is not available in this work. Maruyama et al. (2012) applied the program slicing technique to discover the historical change events to the operation of source code by replaying recorded past code changes. The output is in a set of lines of interest contains all their snap-shots in all the past revisions in which they were modified. Their study are useful for understanding source code growth, nevertheless, the implementation of their slicing task is time-consuming. On the other hand, the study proposed by Zhang et al. (2011) perform the dynamic slicing according to the calling relationship of the program. The work proposed the method called Structured Dynamic Program Slicing (SPS) that use the register or memory address in certain instruction of the program as slicing criterion. The work extracts a part of code, which influenced by users, and organizes the result using the call graph of the program. The summary of the works are shown in Table 1.

Table 1 Summary of Existing Works on Program Slicing Technique

Authors	Method of Slicing	Direction	Technique of Slicing	Tool	Slicing Criterion
(Servant & Jones, 2013)	History Slicing	Backward	Static	Chronos	Set of lines of interest
(Santelices et al., 2013)	Quantitative Slicing	Hybrid	Hybrid	SensA	The score or probabilities of the set of interested lines
(Jain & Poonia, 2013)	Mixed S-D Slicing	Hybrid	Hybrid	N/A	Variables
(Maruyama et al., 2012)	History Slicing	Hybrid	Static	Operation Slice Replayer	Operations change of interest
(Zhang et al., 2011)	SPS - Structured Dynamic Program Slicing	Forward	Dynamic	SPS	The register or memory address in certain instruction of the program

Conclusion

Program slicing has been applied to a range of maintenance tasks. This paper attempts to provide the insight behind computing a program slice and reflects on several types of program slicing techniques that have been written. This study also aims to recommend excellent sources for further information on program slicing. Finally, we offers a direction for future work on program slicing research. We plan to apply this technique taking into consideration the use of domain knowledge in object-oriented programs. Program slicing of such programs is more complicated due to global and local variables, reference parameters, procedure call/return, and recursion. By extracting, the elements in OOP source code and represent it into a form of knowledge representation, this method will be able to enhance a successful evolution of program slicing.

References

Barros, J. B., da Cruz, D., Henriques, P. R., & Pinto, J. S. (2011). *Assertion-based slicing and slice graphs*. *Formal Aspects of Computing* (Vol. 24). http://doi.org/10.1007/s00165-011-0196-1

Binkley, D., Danicic, S., Gyimóthy, T., Harman, M., Kiss, A., & Korel, B. (2005). Minimal slicing and the relationships between forms of slicing. In *Source Code Analysis and Manipulation, 2005. Fifth IEEE International Workshop on* (pp. 45–54). IEEE.

Jain, M. S., & Poonia, M. S. (2013). A New approach of program slicing: Mixed SD (static & dynamic) slicing. *International Journal of Advanced Research in Computer and Communication Engineering Vol*, *2*.

Korel, B., & Laski, J. (1990). Dynamic slicing of computer programs. *Journal of Systems and Software*, *13*(3), 187–195.

Korel, B., & Rilling, J. (1998). Dynamic program slicing methods. *Information and Software Technology*, *40*(11), 647–659.

Koushik, S., & Selvarani, R. (2012). Review on Cost Effective Software Engineering Using Program Slicing Techniques. In *Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012* (pp. 631–637). Springer.

Lahtinen, E., Järvinen, H.-M., & Melakoski-Vistbacka, S. (2007). Targeting program

visualizations. *ACM SIGCSE Bulletin*, *39*(3), 256. http://doi.org/10.1145/1269900.1268858

Maletic, J. I., & Kagdi, H. (2008). Expressiveness and effectiveness of program comprehension: Thoughts on future research directions. *2008 Frontiers of Software Maintenance*, 31–37. http://doi.org/10.1109/FOSM.2008.4659246

Maruyama, K., Kitsu, E., Omori, T., & Hayashi, S. (2012). Slicing and replaying code change history. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012*, 246. http://doi.org/10.1145/2351676.2351713

Normantas, K., & Vasilecas, O. (2013). A Systematic Review of Methods for Business Knowledge Extraction from Existing Software Systems, *1*(1), 29–51.

Roongruangsuwan, S., & Daengdej, J. (2010). A test case prioritization method with practical weight factors. *J. Software Eng*, *4*, 193–214.

Saleem, M., Hussain, R., Ismail, V., & Mohsin, S. (2009). Cost effective software engineering using program slicing techniques. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (pp. 768–772). ACM.

Santelices, R., Zhang, Y., Jiang, S., Cai, H., & Zhang, Y. (2013). Quantitative program slicing: Separating statements by relevance. *2013 35th International Conference on Software Engineering (ICSE)*, 1269–1272. http://doi.org/10.1109/ICSE.2013.6606695

Sasirekha, N., & Hemalatha, M. (2011). Program Slicing Techniques and its Applications. *International Journal of Software Engineering & Applications*, *2*(3), 50–64.

Servant, F., & Jones, J. a. (2013). Chronos: Visualizing slices of source-code history. *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, 1–4. http://doi.org/10.1109/VISSOFT.2013.6650547

Weiser, M. (1982). Programmers use slices when debugging. *Communications of the ACM*, *25*(7), 446–452.

Zhang, R., Zheng, Y., Huang, S., & Qi, Z. (2011). Structured Dynamic Program Slicing. *2011 International Conference on Computer and Management (CAMAN)*, 1–4. http://doi.org/10.1109/CAMAN.2011.5778759