# Calculating different software matrices for Java and Jimple

[1]Shammi Jain,[2]Ajay Kushwaha
[1]M.Tech. Scholar,[2]Associate Proff.
[1]Rungta College of Engineering & Technology, Bhilai, (CG) India

***Abstract*— Java is a very promising language, which makes use of JVM for platform independency. Java compiler converts program into byte code which is further executed. There is scope for the optimization in the byte code produced by the compiler. For the static optimization in java one can work on the byte code or different intermediate code of java. The intermediate code of java has immeasurable unwrapped code and sophisticated statement that directly result to reliability of any computer code. Java code has solely abstraction of this code that hides immeasurable quality kind user. The 3 address code that is one in every of the intermediate kind in compiler method is extremely on the subject of the assembly level code, therefore analysis on this level turn out additional correct result and generate nice impact on computer code. There square measure different styles of analysis potential on TAC (three address code). Jimple is one in every of the TAC code. Various tools have been for the conversion and SOOT is one of the most popular java code optimization analysis tool.**

**These intermediates code are also useful in the various are of Computer Science like Network Security, Software Engineering etc. Software engineering is associate degree engineering branch related to development of wares exploitation well-defined scientific principles, strategies and procedures. The end result of software system engineering is associate degree economical and reliable wares.**

**Formulas have been proposed to find the reliability of software. Reliability itself depends on the understandability of software system. Understandability depends on the some software metrics like number of lines, number of comments etc. Finding the accurate reliability is more difficult. In the project we have given experimental results of the analysis of the software metrics between java and jimple. We have found that variation in jimple file depends on the java code that we write. In our work we are comparing the deviation of different software metrics on java and jimple file. We have found that some of the metrics differ in a linear way where as some the metrics depends on the logic.**

***IndexTerms*— Soot-Byte code optimization framework, Software Engineering, Usability, Reliability**

---

## I. INTRODUCTION

To achieve platform independency in Java language code is converted into byte code and then it is converted in the native machine code. Since Java is an Open Source thus it opens the way to optimize the intermediate code for the better performance. SOOT is a framework which provides various intermediate code representations for performing optimization analysis.

### Intermediate Representations

The Soot framework [1] provides four intermediate representations code for:

- Baf
- Jimple
- Shimple
- Grimp

The representations provide different levels of abstraction on the represented code and are targeted at different uses e.g., baf is a byte code representation resembling the Java bytecode and Jimple is a stack less, typed 3-address code suitable for most analyses. In this section we will give a detailed description of the Jimple representation and a short description of the other representations.

In software engineering software measures may be understood as a method of quantifying and figuration numerous attributes and aspects of software package. Software package Metrics offer measures for numerous aspects of software package method and merchandise. Software package measures area unit elementary necessities of software package engineering. They not solely facilitate to regulate the software package development method however additionally aid to stay the standard of final product wonderful.

Various software metrics are used to find the various factors to calculate the reliability, reusability, maintainability etc. This is shown in the figure 1.

In figure 1, we can see that there are various factors on which reliability of software depends on various factor. One of the factor for the reliability is understandability and understandability depends on the following software metrics:

- Line of codes
- Number of comments
- Number of methods
- Number of classes.

In the project we are going to do a comparative study of java and jimple file for the above 4 software metrics.

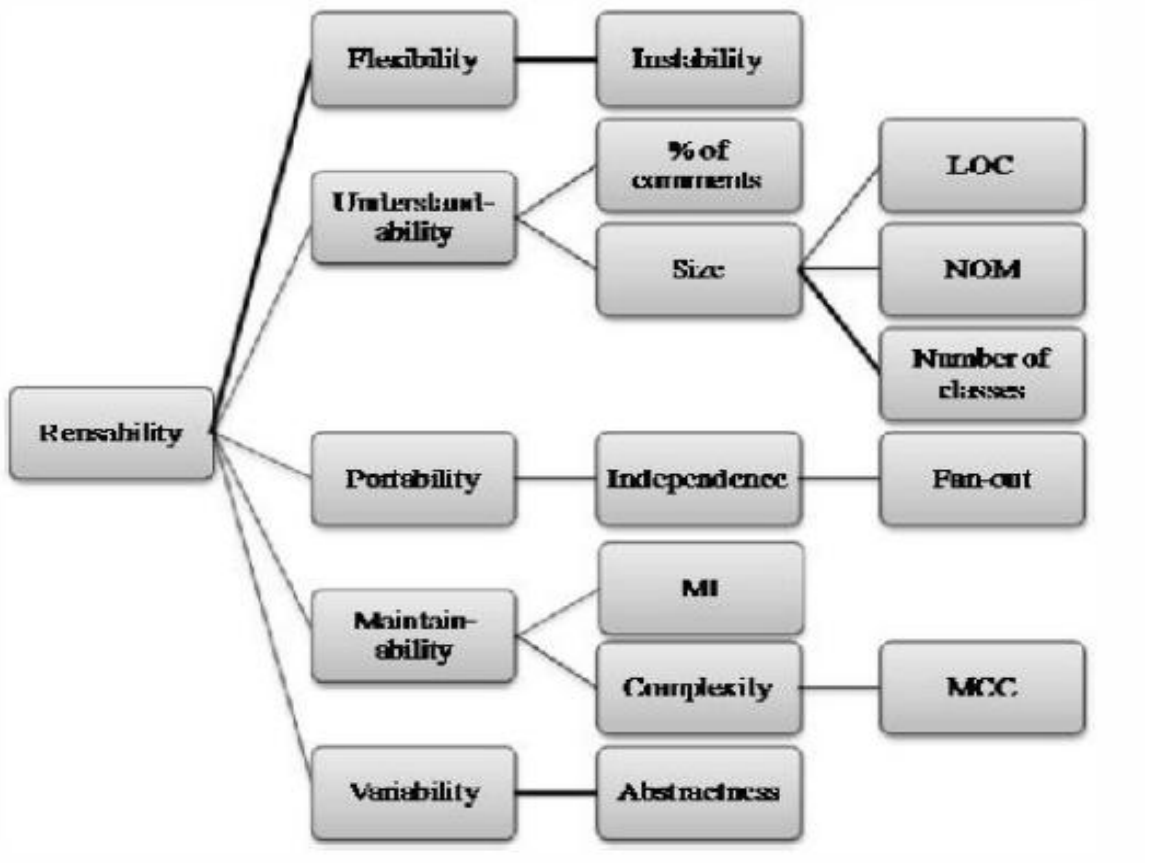


**Figure1 Figure Reusability Attribute Model**

Work has been done to perform software metrics analysis on java file. If these operations are performed on the intermediate code then we can get more accurate results because intermediate codes are closer to the executable code.

## II. Problem Identificatio

There are many approaches have been provided to find the reusability of software but it is very hard to find the accurate reliability based on the reusability [17]. Java code is not directly run on the machine first it is changed to some intermediate code and it is then changed to native machine code. We have observed that java compiler when converts java code to byte code then it performs several optimization operations on java code, due to this number of methods calling is increased, number of lines are increased and many more changes occurs in intermediate code than the actual java source code.

In the area of Software Engineering reusability is difficult to find the reusability of software and whatever experiment have been done those are based on Java code. So here in our experiment we are performing operations to finding understandability, which is a factor to find reusability, on Jimple file and comparing with result of java file.

## III. Methodology

To perform the comparative study we have done the following steps:

1. Used SOOT framework API to generate Jimple file of java.
2. Selected 11 java programs to perform our analysis
3. Developed a java program to count software metrics in java as well as Jimple file.
4. Calculated software metrics using our developed code.

## IV. Result And Discussion

Beforeyoubegintoformatyourpaper,firstwriteandsavethecontentasaseparatetextfile.Keepyourtextandgraphicfilesseparateuntilafterthetexthasbeenformattedandstyled.Donotusehardtabs,andlimituseofhardreturnstoonlyonereturnattheendofaparagraph.Donotaddanykindofpaginationanywhereinthepaper.Donotnumbertextheads—thetemplatewilldothatforyou.

Result of total number of lines, comments and methods in Java and Jimple file Difference in the number of lines and number of methods in Jimple and java file is our motivation to work in this area. Calculation of various metrics in java and Jimple file is given in table below:

**List of program and result of different metrics**

| S No | Program Name | No of lines In Java File | No of Lines in Jimple file |
|---|---|---|---|
| 1 | HelloWorld | 14 | 23 |
| 2 | SwapCalulation | 31 | 65 |
| 3 | ArraySort | 49 | 141 |
| 4 | GenericMethod | 61 | 117 |
| 5 | Factorial | 67 | 108 |
| 6 | StringPermutaion | 68 | 151 |
| 7 | MatrixExample | 71 | 224 |
| 8 | CreateThreadRunnable | 74 | 146 |
| 9 | Dog | 75 | 116 |
| 10 | MainClass | 84 | 230 |
| 11 | StackExample | 162 | 266 |

Above is the result of our experiment in which different java files and jimple files are given to the program written in step2 for finding different software metrics.

## V. Conclusion And Future Work

Comparing different java and Jimple file gives variations for different software metrics as shown table. Two java file which has small difference in number of line but respective Jimple file has big difference. From result table we can observe that in program number 9 and 10, number of lines in java file is 75 and 84 respectively whereas number of lines in corresponding Jimple file is 115 and 230 in respectively. Various factors affect the variation in number of lines in the creation in Jimple file like the calculation in the program; number of method calling, reusability of the components etc. Calculations of the statements in java file increases number of lines, because one expression is broken down in many intermediate statements. We have also found that comments in the Jimple file is completely removed and number of methods in java file and Jimple file is in a proportion.

In our work we have seen that understandability of any program depends on the reusability, reusability depends on the percentage of comments and size of the program, and If we go in more depth, Size of a program depends on line of code, number of methods, number of classes. We have performed comparison between java file as well as Jimple file. Jimple file is 3 address code, an intermediate code of Java language, generated by SOOT. Thus if we perform analysis of understandability to find reliability on Jimple file instead of simple java file, we get more accurate result.

## References

[1] Arni Einarsson, Janus Dam NielsenA, "Survivor's Guide to Java Program Analysis with Soot", BRICS, Department of Computer Science University of Aarhus, Denmark, 07/17/2008

[2] Raja Vall´ee-Rai Phong CoEtienne Gagnon, Laurie Hendren Patrick Lam Vijay Sundaresan, "Soot - a Java Bytecode Optimization Framework",Sable Research Group School of Computer Science McGill University

[3] Raja Vallee-Rai, "The Jimple Framework" , Sable.mcgill.ca, 11 feb 1998.

[4] Aiwu Shi and Gleb aumovich "Field Escape Analysis for Data Confidentiality in Java Components" Department of Computer and Information Science Polytechnic University 6 MetroTech Center, Brooklyn, NY 11201, USA, 2007

[5] Kyungwoo Lee, Samuel.P. Midkiff "A Two-Phase Escape Analysis for Parallel Java Programs",PACT'06, September 16–20, 2006

[6] Binxian Tao, Ju Qian, Xiaoyu Zhou, "Side-Effect Analysis with Fast Escape Filter", SOAP'12, June 14, 2012, Beijing, China.

[7] Lei Wang, Xikun Sun, "Escape Analysis for Synchronization Removal", SAC'O6 April 23-27,2006, Dijon, France

[8] Ada Diaconescu, Adrian Mos, John Murphy, "Automatic Performance Management in Component Based Software Systems", Enterprise Ireland Informatics Research Initiative 2001

[9] Raja Vallee-Rai, Laurie J. Hendren, "Jimple: Simplifying Java Bytecode for Analysis and Transformation", Sable Research Group, McGill University.

[10] BRUNO BLANCHET, "Escape Analysis for JavaTM: Theory and Practice" in ACM Transactions on Programming Languages and Systems, Vol. 25, No. 6, November 2003, Pages 713–775.

[11] Sun Microsystems, Inc. The Java HotSpot Virtual Machine, v1.4.1, Sept. 2002.http://java.sun.com/products/hotspot/.

[12] R. Griesemer and S. Mitrovic "A compiler for the Java HotSpotTM Virtual Machine". In L. B¨osz¨orm´enyi, J. Gutknecht, and G. Pomberger, editors, The School of Niklaus Wirth: The Art of Simplicity, pages133‑152. dpunkt.verlag, Heidelberg, 2000.

[13] J.-D. Choi et al. "Stack allocation and synchronization optimizations for Java using escape analysis". ACM Transactions on Programming Languages and Systems, 25(6):876‑910, Nov. 2003.

[14] R. Cytron et al. "Efficiently computing static single assignment form and the control dependence graph". ACM Transactions on Programming Languages and Systems, 13(4):451‑490, Oct. 1991.

[15] C. Wimmer and H. M¨ossenb¨ock. "Optimized interval splitting in a linear scan register allocator". In Proceedings of the Conference on Virtual Execution Environments, June 2005.

[16] Jong-Deok Choi, Mannish Gupta, Mauricio Serrano, Vugranam C. Sreedhar, Sam Midkiff, "Escape Analysis for Java". IBM T. J. Watson Research Center.

[17] Kirti Tyagi, Arun Sharma, "An adaptive neuro fuzzy model for estimating the reliability of component-based software systems", 6 May 2014, King Saud University.

[18] Khyati M. Mewada, Amit Sinhal, Bhupendra Verma, "Adaptive Neuro-Fuzzy Inference System (ANFIS) Based Software Evaluation", IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 5, No 1, September 2013 ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784

[19] Arun Sharma, P. S. Grover, Rajesh Kumar, "Dependency Analysis for Component-Based Software Systems", SIGSOFT Software Engineering Notes, July 2009 Volume 34 Number 4.

[20] John Whaley and Martin Rinard, ―Compositional Pointer and Escape Analysis for Java Programs‖. OOPSLA _99 11199 Denver, CO, USA 1999 © ACM l-58113-238s7/99/0010.

[21] J. Bogda and U. Holzle. ―Removing unnecessary synchronization in java‖. OOPSLA _99 11199 Denver, CO, USA © 1999 ACM l-58113-238-7/99/0010.

[22] Alexandru Salcianu and Martin Rinard ―Pointer and Escape Analysis for Multithreaded Programs‖. PPoPP'01, June 18-20, 2001, Snowbird, Utah, USA. Copyright 2001 ACM 1-58113-346-401/0006.

[23] C Andreas Krall and Mark Probst. Monitors and Exceptions: How to implement Java efficiently. In S. Hassanzadeh and K. Schauser, editors, ACM 1998 Workshop on Java for High-Perfonnance

[24] Tim Lindholm and Frank Yellin. ―The Java Virtual Machine Specification‖. Copyright © 1997 Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A..

[25] J. Aldrich, C. Chambers, E.G. Sirer, and S. Eggers, "Static analyses for eliminating unnecessary synchronization from Java programs," in Proceedings of the 6th International Symposium on Static Analysis. London: Springer-Verlag, 1999, pp. 19-38.

[26] Jong-Deok Choi, Mannish Gupta, Mauricio Serrano, Vugranam C. Sreedhar and Sam Midkiff ―Escape Analysis for Java‖ OOPSLA _99 1 l/99 Denver, CO, USA © 1999 ACM l-581 13-238.7/99/0010.

[27] Bruno BLANCHET ―Escape Analysis for Object Oriented Languages. Application to JavaTM ― OOPSLA _99 t l/99 ,Denver, CO, USA © 1999 ACM l-581 13-238.7/99/0010.

[28] Lian Li, Cristina Cifuentes, Nathan eynes ―Boosting the Performance of Flow-sensitive Points-to Analysis using Value Flow‖. ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary. Copyright 2011 ACM 978-1-4503-0443-6/11/09

[29] Bjarne Steensgaard ―Points-to Analysis in Almost Linear Time‖. Copyright c 1995 by the Association for Computing Machinery, Inc.

[30] Gokhale, Swapna S., and Suhrid A. Wadekar. "Reliability Maximization of Component-Based Software Systems."

[31] Lionel Briand, "Introduction to Software Reliability Estimation" Simula Research Laboratory, 2010

[32] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Pasareanu, Robby, Hongjun Zheng, "Bandera: Extracting Finite-State Models from Java Source Code".

[33] Dolbec, Jean, and Terry Shepard. "A component based software reliability model." Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research. IBM Press, 1995.

[34] EMAMI, M., GHIYA, R., AND HENDREN, L. 1994. Context- sensitive interprocedural points-to analysis in the presence of function pointers. In Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation.242-256

[35] J. Gosling, B. Joy, G. Steele, and G. Bracha, The JavaTM Language Specification, 3rd ed.. Boston, MA: Addison-Wesley, 2005. T. Lindholm, and F. Yellin, The Java Virtual Machine Specification, 2nd ed. Boston, MA: Addison-Wesley, 1999.

[36] LANDI, W. AND RYDER, B. 1992. A safe approximate algorithm for interprocedural pointer aliasing. In Proceedings of the SIGPLAN '92 Conference on Programming SIGPLAN Not. 27, 6, 235–248.

[37] WILSON, R. P. AND LAM,M. S. 1995. Efficient context-sensitive pointer analysis for C programs. In Proceedings of the SIGPLAN'95 Conference on Programming Language Design and Implementation. SIGPLAN Not. 30, 6, 1–12.

[38] D. Mandelin, L. Xu, R. Bodik, and D. Kimelman. Jungloid mining: "helping to navigate the api jungle". In Proc. of the ACM SIGPLAN Conf. on Prog. Lang. Design and Impl., pages 48–61, 2005.

[39] Bruno Dufour, Barbara G. Ryder, Gary Sevitsky. "Blended Analysis for Performance Understanding of Framework-based Applications". In ISSTA'07, July 9–12, 2007, London, England, United Kingdom.

[40] D. Gay and B. Steensgaard. "Fast escape analysis and stack allocation for object-based programs". In Proc. of the Int'l Conf. on Compiler Const. (CC), pages 82–93. Springer-Verlag, 2000.

[41] J. Bogda and U. H¨olzle. "Removing unnecessary synchronization in Java". In Proc. of the ACM SIGPLAN Conf. on Object-Oriented Prog. Sys., Lang. and Appl. (OOPSLA), pages 35–46. ACM Press, 1999.

[42] Fazal-e-Amin, Ahmad Kamil Mahmood, Alan Oxley , "An Evolutionary Study of Reusability in Open Source Software", Computer and Information Sciences Department, Universiti Teknologi PETR ONAS, Bandar Seri Iskandar, 31750 Tronoh, Perak, Malaysia 2012.

Websites
[43] http://www.sable.mcgill.ca /soot/doc/
[44]http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.htm
[45] http://www.tutorialspoint.com/software_engineering/
[46] http://java.sun.com/
[47] https://www.wikipedia.org/

**Books:**

Roger S. Pressman, "Software Engineering: A Practitioner's Approach" McGraw-Hill Higher Education 2001 ISBN:0072496681

Compilers Principles, Techniques and Tools| (Second Edition): Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman.