ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 11, November 2013

Does Object-orientation really affect the Algorithm performance

Nilima Karankar[1], Lalit Gehlod[2], Sarita Rathore[3]

Assistant Professor, Computer Engineering Department, IET, DAVV, Indore, India[1]

Assistant Professor, Computer Engineering Department, IET, DAVV, Indore, India[2]

Student, Computer Engineering Department, IET, DAVV, Indore, India[3]

Abstract: The performance analysis and measurement of object-oriented algorithms is becoming a new research topic in software design. Different evaluation parameters and metrics for different performance aspects are introduced for algorithm optimization for faster code development. Achieving performance efficiency with respect to time and code adaptability and reusability are the main goals of various aspect of software design. Object orientation is really helpful in coding algorithms fast and easily. This dissertation work will allow you to try various what-if scenarios using different object-oriented concepts like aggregation, composition, inheritance etc. and thus provide algorithmic improvements to it. Various work had been done for algorithm optimization with respect to hardware domain like CPU speed, memory size and I/O specifications but how code efficiency can be improved in software will be done in this dissertation work. The paper evaluate the effectiveness of software with respect to time, several optimizations will be suggested to improve the software quality and programmer productivity.

Keyboard: Object orientation and concepts, memory

I. INTRODUCTION

The goal of this dissertation work is to present the metrics for improving code efficiency so that object-oriented software development can be better understood, controlled and measured for better management of system development and maintenance. The aim is to disclose how changes in selection of different object-oriented concepts for implementation will affect your execution time of the software. Those aspects could be analyzed and optimizations will be suggested. This aided to coders in building efficient software with respect to time. These features measures in particular way could become the set of criteria for recognition of performance measurement of software. Interpretations of those results do not lead to generalized conclusions but shed more light on the phenomena of software optimization and defines the way it could be diagnosed, interpreted and improved. This dissertation work will take input as implementations of application built with the application of different object-oriented concepts. Output of our work is the information and knowledge that enables to ground the decisions with respect to software optimization. Earlier research work on Software Complexity was done by measuring the Cyclomatic Number of the program. Here the focus has been on developing various object-oriented metrics to measuresoftware quality, complexity and productivity.

II. RELATED WORK

Software complexity has been defined differently by many researchers. IEEE defined software complexity as "the degree to which a system or component has a design or implementation that is difficult to understand and verify".

An important issue encountered in software complexity analysis is the consideration of software as a human creative artifact and the development of a suitable measure that recognizes this fundamental characteristic. The existing measures for software complexity can be classified into two categories: the macro and the micro measures of software complexity.

Major macro complexity measures of software have been already proposed in several papers. The former considered software complexity as "the resources expended". The latter viewed the complexity in terms of the degree of difficulty in programming. Zuse defines software complexity as the difficulty to maintain, change and understand software. It deals with the psychological complexity of programs. According to Henderson-Sellers the cognitive complexity of software refers to those characteristics of software that affect the level of resources used by a person performing a given task on it. Basili defines software complexity as the measure of the resources expended by a system while interacting with the piece of software to perform a given task. Here, interacting system may be a machine or human being. Complexity may be defined in terms of execution time and storage required to perform the computation when computer acts as an interacting system, In case of human being (programmer) as an interacting system, complexity is defined by the difficulty of performing tasks such as coding, testing, debugging and modifying the software.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 11, November 2013

The micro measures are based on program code, disregarding comments and stylistic attributes. This type of measure typically depends on program size, program flow graphs or module interfaces such as Halstead's software science metrics and the most widely known cyclomatic complexity measure developed by McCabe. However, Halstead's software metrics merely calculate the number of operators and operands they do not consider the internal structure of software components, while McCabe's cyclomatic measure does not consider me /Os of software systems. It is accepted by both software developers and researchers that complexity of software can be controlled more effectively through component-based and object-oriented approach than traditional function-oriented approach. It is because that object-oriented and component-based paradigms control complexity of a software system by supporting hierarchical decomposition through both data and procedural abstraction.This paper presents an object-oriented approach towards software complexity by focusing on the computational complexity that characterizes the runtime performance of an algorithm. The runtime measurement of various Java applications, built with the application of different object-oriented concepts, are measured and compared to determine which is faster with respect to time. Accordingly several metrics are suggested here to build time efficient Java object-oriented code. The complexity parameters are different object-oriented concepts like inheritance, interfaces, composition etc. Here we determine software complexity with respect to time in object-oriented software development. So, there is strong demand and need for designing of complexity metrics for software which may be used by the application developers to choose the best object-oriented concept in software development and then finally produce the better quality software

III. PROPOSED WORK

Through early background research, it has been decide to initially investigate the ways of implementing this benchmark.

Timing analysis of Java source code:

To implement this benchmark, the following steps are taken:

1. Finding the running time of the Java source code
2. Interpretation of measured timings of the source codes.

The initial step towards implementation is the creation of the Java source codes with application of different Object-oriented concepts.

Figure shows the template that List the various steps:

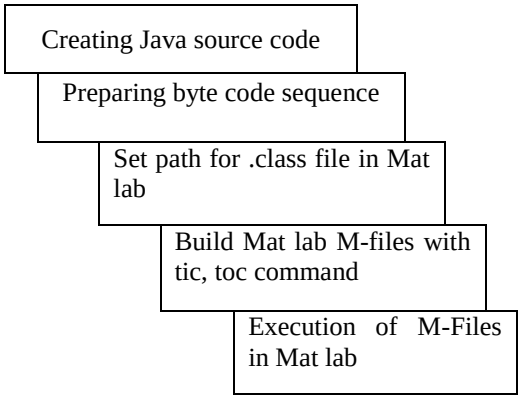

Figure 1.1 Source code prediction template using Mat lab

After creating Java source code, it is compiled to produce the byte code or .class file. The running time measurement is done in matlab after the successful connectivity of Java with matlab. Then matlab m-files are prepared with the embedded Java code and matlab commands to measure the execution time. After executing m-files in matlab we get the running time in seconds along with the output of the program. The scenario is shown in following figure:

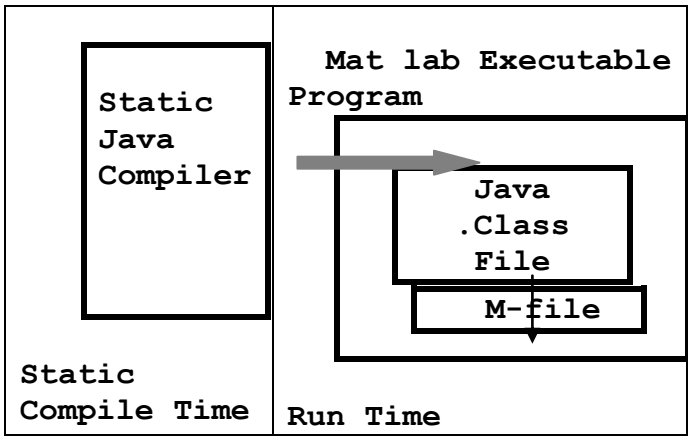

Figure 1.2 Execution of Java code in Matlab

There is a clear point where the execution of the source Code block is significantly different than the other Java Application built with other concept. For example, a Program is built with the inheritance and the same program Is built with the composition concept are both producing The same output but their running time will differ, so we Suggest that coding with composition is faster. Such Metrics are provided here so that application developers Can build an efficient code with respect to time. One domain of the characterization is the predicted application'sexecution time and as Java has become a popular programming medium our majority of work is done in javaand Matlab simulation software .

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 11, November 2013

IV. RESULTS AND DISCUSSIONS

The objective to implement this benchmark is to establish a framework for an efficient and accurate Java source code development. The graphical representations of the results are shown with the following bar charts:

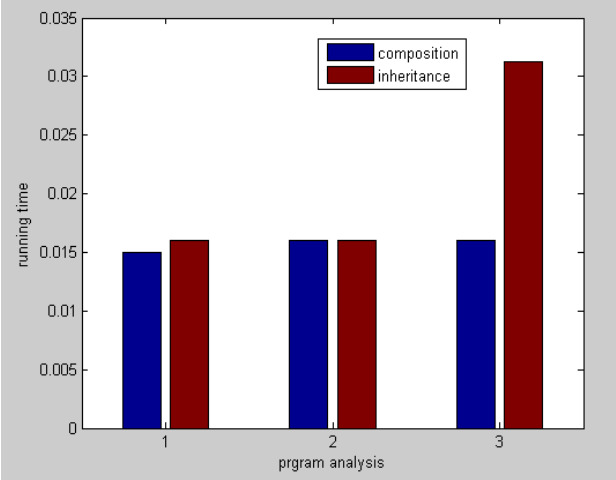

Figure 1.3 Compositions versus Inheritance

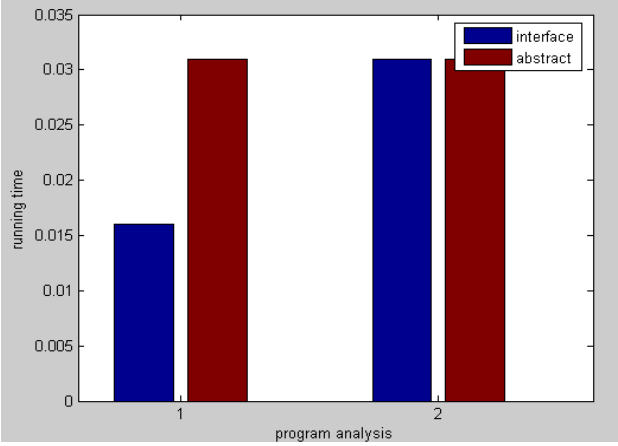

Figure 1.4 Interface versus abstract class

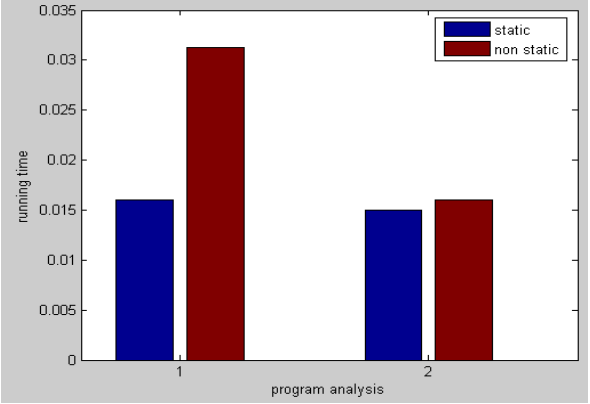

Figure 1.5 Static versus non-static members

The above charts represent the results tabulated as shown in the figure:

Benchmarks	# of runs	Minimum execution time in seconds
Inheritance	6	0.016
Composition	6	0.015
Abstract class	7	0.03
Interface	7	0.015
Static member	5	0.015
Non-static member	5	0.016

Table 1.1 Execution Time

By observing the tabular representation of the results, it could be seen that the relationship between the measured execution time and java applications built with the different object-oriented concepts is consistent with the theoretical concepts in the design patterns. To ensure the observations made are fair and accurate, several applications within the Java benchmark suites have also been taken into consideration in measuring the execution time. Experimental results presented here for benchmark suite shows that there is a marked reduction in execution time as we move from inheritance to composition, abstract class to interface and non-static member to static members.

V CONCLUSIONS

The presented paper provides certain metrics that are to be followed to do software optimizations. It should help in improving the programmer's productivity and software development. The software optimizations suggested here do need to be carefully employed to yield good results.

There are a number of directions in which this research can continue. One metric which needs to be explored is the storage analysis of Java Source code. It would be interesting to test this by designing a set of Java applications using the object-oriented technology and then measuring the complexity in storage space and determining the applications that need least space so as to save time and cost of software development.

REFERENCES

[1] Lars O Anderson. Program Analysis and Specialization C Programming language Ph.D.dissertation, University of Copenhagen, DIKU, May 1994.

[2] V.R Bacilli, Qualitative Software Complexity Models: A Summary in Tutorials on Models and Methods for Software Management and Engineering, Los Alamitos, Caliph: IEEE Computer Society Press, 1980.

[3] J.K Kearney, R.L. Sedlmeyer, W.B. Thompson, M.A Gary, and M.A Adler, Software Complexity Measurement,
vol. 28, New York: ACM Press, 1987.

[4] T.H McCabe, "A Complexity Measure", IEEE Trans. Software Eng., vol. 2,1976.