Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.776490

ASSESSMENT OF JAVA PROGRAM USING METRICS FOR OBJECT ORIENTED DESIGN (MOOD)

Dr K D Gupta[1] Sangeeta Gupta[2]
[1]Head & Guide, Department of Computer Science, Apex University, Jaipur
[2]PhD Scholar, Department of Computer Science, Apex University, Jaipur
Guptasangi9@gmail.com, Kdevgupta@gmail.com

Abstract
Years of software testing research have provided us with new methods to reason about and test the behavior of large software systems with hundreds of thousands of lines of code. Many of these methods, including genetic algorithms, swarm intelligence, and ant colony optimization, were inspired by nature. They do, however, employ a one-way comparison, implying that they are taking from nature without giving back. In this work, MOOD metrics are extensively used in the Object-Oriented world to measure various characteristics of a programme. A Java programme assessment and grading system based on the MOOD was developed in this study. The MOOD metrics for Java programmes have been adjusted as a result of experimentation so that they may be examined. A weight factor has also been included to illustrate the significance of each characteristic's value. The system has been tested with a variety of applications, each with its own degree of intricacy and usefulness. Furthermore, the University of Jordan has put the metrics to the test in terms of evaluating and grading student programmes. The system routinely produces favourable results.

Introduction

Software metrics are measures used for software engineering items to be assessed & characterized. To improve software quality, software metrics are one of the essential techniques used for software engineering. Program metric is a basic measure of quantity derived from any feature in the life cycle of the software. Software metrics have the work to identify substantial software product evaluations and lead us to fascinating management and technical decisions. In every phase of the life cycle of development (Bieman & Kang, 1995), software metrics have developed into an essential component of software development. The software metric is associated with different measures & the development of computer software. In software metrics, research tends to focus mostly on static metrics that may be acquired through static analysis of a programming artifact. In addition to processing variables, the resulting software quality should also be affected by product aspects. The design (Chidamber & Kemerer, 1991) is one of them. The architecture transition analysis is an activity in which a skeleton is developed for computable employment that assists the stated needs of the system. This shift gives a lot of liberty degrees. Decisions on the greatest options are generally fuzzy & depend mostly on expert opinions. Cumulative information, in other terms, plays a crucial role in The design stage. This open issue is supposed to be alleviated by extensive use of trends, structures, and other reusable components. However, the current approach is still not widely adopted. A series of measures(Harrison et al., 1998) dubbed MOOD has been used to assess OO design features. The reasons for a definition of a MOOD were:

1. covering essential structural principles like encapsulation, inheritance, polymorphism & transmittal of messages of an object-oriented paradigm;
2. formal description to prevent measurement subjectivity & hence enable replicability,
3. size independence to enable inter-projected comparison, thus promoting cumulative information & language independence by enabling comparison of heterogeneous systems implementation to extend the apps of this set of criteria.

Each of these metrics corresponds to an essential structural mechanism of an OO paradigm like encapsulation (MHF & AHF) (MIF & AIF), polymorphism (POF) & message passing (COF). The descriptions of MOOD metrics do not refer to specific linguistic structures. However, because each language has its structure, which makes it possible to create more or less detailed OO-mechanisms, an abstract binding is supplied for 2 OO languages (C++ (Stroustrup & Stroustrup, 1995) & Eiffel(Meyer, 1994), is included ahead.

Related Work:

MOOD metrics have been used by many software developers to assess object-oriented programmes. Abreu et al. [3] pioneered the quantitative evaluation of object-oriented software system design attributes. An experiment for collecting and analysing MOOD measurements was defined, and several design hypotheses were examined. As a data source, a significant number of C++ class taxonomies were utilised. It was built and used to collect these metrics. The collected data was statistically analysed.

Abreu and Melo [4] conducted experiments on the impact of object-oriented design on software quality characteristics. MOOD measurements are used to assess OO design methods. The identical set of standards was used to construct all eight small-scale information management systems. OO design characteristics like as inheritance and polymorphism have been found to influence quality factors such as reliability and maintainability. Abreu et al. presented Moody to Eiffel. Code samples were supplied for illustration and understanding. As previously indicated, these metrics were taken from a sample of Eiffel libraries. Following statistical analysis of the sample, certain hypotheses were developed and addressed. Following that is an initial set of prescriptive principles for the design process. Harrison et al. [6] performed a study on MOOD measurements. They published their results. From the standpoint of measurement theory, the MOOD metrics were examined in terms of encapsulation, inheritance coupling, and polymorphism: the object-oriented traits intended to be quantified. MOOD measures were used to actual data acquired from three diverse application fields to support this theoretical assumption. The findings indicate that through the use of metrics, an overall assessment of a software system may be valuable in controlling software development activities.

There are also a variety of instruments for reviewing programmes available, such as: Redish and Smyth [7] developed AutoMARK to evaluate FORTRAN programmes produced by students in line with their particular style. AUTOMARK++, a technique developed by Al-Ja'afer and Sabri [8], is used to analyse object-oriented programmes based on style. Berry and Meekings' [9] approach can be used to evaluate students' C language applications. It was created by Jackson and Usher [10] to assess programmes based on their correctness, efficiency, complexity, and style using the ASSYST software tool. Jumaa [11] has developed a tool for assessing structural languages like as Pascal, FORTRAN, C, and Basic based on Halstead, McCabe, AUTOMARK, and Lipow and Thayler models.

MOOD Matrics

The MOOD set (Metrics for OO Design) contains the following metrics:

- Attribute Hiding Factor (AHF)
- Method Hiding Factor (MHF)
- Attribute Inheritance Factor (AIF)
- Coupling Factor (COF)
- Polymorphism Factor (POF)
- Method Inheritance Factor (MIF)

The complexity of software indicates the complications of software understanding, maintenance, modification, and reuse. According to IEEE definitions, the complexity of the system or component is the extent to which it is difficult to comprise and verify the design or implementation. The objective-oriented (OO) method is played a critical part in Software Developers for 25 years but research is currently on to modify and improve these techniques. Due to different software-design measurements, the popularity of OO programming is. These design measurements are used to detect the maintenance and reliability of the object-oriented design. The object-oriented paradigm's reusability characteristic is used to maximize the capability of the software, although its usage is complicated.

Therefore, a complete measure of complexity is needed, incorporating other software factors, to address this problem. A new technique is needed in the software business to quantify software complexity more precisely. Measuring the cognitive component of OO software helps to better understand complexities since it makes an understanding of the inner structure and input-output of the program difficult for designers and consumers. So the work needed to design, test, and maintain the software may be easily predicted. A new measure is proposed in this paper, which goes above AWCC limits. It helps to better understand the complexity of the class. For the development of object software, we offer a new metric termed Cognitive Weighted Inheritance Class Complexity (CWICC) (Maheswaran & Aloysius, 2018).

In the previous two decades, object-oriented (OO) technology-dominated software engineering. The maintenance of OO software is one of the reasons behind this predicament. The quality of its design must also be assessed using appropriate quantitative methods to assess the maintainability of OO software. Since it is difficult and costly to modify after the design has been implemented. This means that the design should be good from the start and software metrics are the tools to assess the design quality.

The OO software is popular because of its powerful features such as encapsulation, the composition of objects, heritage, interaction, polymorphism, dynamic binding, and reusability. In addition, the OO approach is characterized by its classes and objects, defined according to characteristics (data) and operations (methods). In class declarations, these components are defined. This includes the technique that works on data in response to a message. The method is an important one While method complexities directly impact software's comprehensibility, method-based complexity measures have not yet been thoroughly investigated. Very few metrics are available in the literature to evaluate the complexity of the process. The underlying architecture and special features of OO design are not taken into account in the majority of these measures(Misra et al., 2011).

Statement of Research Problem

The system has assessed the quality of a wide range of programmes, each with a unique level of complexity and design. In each case, the algorithm properly identified the program's faults. Appendix A has three different programme designs, each with a unique set of evaluation results.

Both applications provide for easy access to and storage of information about a company's personnel. The employee's initial and last name, as well as the employee ID number, are included in each programme. Temporary and long-term hourly employees are both included in this breakdown of earnings. All long-term employees are subject to deductions for benefits. Permanent piece-worked workers are well-versed in product quality and cost per unit. Also included among fixed-wage workers are commission-earning employees.

Methodology

The following problems in design result in a poor score for Design 1:

- When compared to the total number of methods in use, the number of inherited methods is quite low. Inheritance reduces the amount of errors in a software. Thus, the application is given a low rating since it is expected that the quality of its content would decline as more inherited methods are used.
- However, it is estimated that between 12.7 and 21.8 percent of the program's strategies are yet to be discovered. Gradually adding additional information to classes is the best way to go about the implementation process. The above-mentioned data was gathered by the use of obscure methodologies, hence supporting an increase in MHF. Since a low MHF indicates a lack of abstraction, this element gets a failing score.
- Polymorithism's complexity reduction is too low, resulting in a reduction in complexity.

For a better grade, use inheritance to raise the standard of the code, and simplify the class as much as possible. A considerable change in grade between designs 1 and 2 was made possible via inheritance, which allowed the classes to be more easily reused. However, there are still a few drawbacks, such as: despite the great rating Since polymorithism makes testing and maintaining the programme more complicated and harder to utilise inherited techniques as a result, it should be minimised or eliminated altogether. Though the ability to pass on genes is one of the benefits of doing so,

- It is imperative that the programme be redesigned in order to further enhance its quality.
- According to this design's highest score, only MHF deviates from the expected range. This isn't a big problem because the programme only has two capabilities (read and write). There's no need for any secret methods.
- Depending on the weight of a particular item, grades are altered accordingly. Overriding all other factors, COF is more important. It also overrides other factors like MIF, MHF, POF, and even other factors such as MHF or AIF.

The grade given to the evaluated program depends on Table 1. This table is produced from reported study [2,3,4,5,6] and the results of experiments.

Table 1: The range of MOOD factors

Factor	Minimum	Maximum	Minimum Tolerance	Maximum Tolerance
MHF	12.70%	21.80%	9.50%	36.90%
AHF	75.20%	100%	67.70%	100%
MIF	66.40%	78.50%	60.90%	84.40%
AIF	52.70%	66.30%	37.40%	75.70%
COF	0%	11.20%	0%	24.30%
POF	2.70%	9.60%	1.70%	15.10%

It should be noted that the minimum value of COF reported in the literature is 4.0%. This is due to the fact that coupling is required in every program to deliver some functionality.

The final grade is computed as follow:

```
Input: Indicator
Output: Grade
If indicator value in range
Than Grade=100
Elseif in tolerance range
Then 60 ≤Grade<100
Else  Grade < 60
Total Grade= $\sum_{i=1}^{n} grade[i] * w[i]$
Final Garade= Total Grade/2
```

Table 2: evaluation Result

Factor	Model Program		Evaluated Program		Factor Weight
	Lower Bound	Upper Bound	Score	Grade (%)	
Method Hiding Factor (MIF)	12.7	21.8	0	0	1
Attribute Hiding Factor (AHF)	76	100	100	100	2
Method Inheritance Factor (MIF)	66.4	78.5	0	0	2
Attribute Inheritance Factor (AIF)	52.7	66.3	0	0	1
Coupling Factor (COF)	0	11.2	0	100	3
Polymorithism Factor (POF)	2.7	9.6	0	0	2
Final Grade is 45 %					

When it comes to predicting system quality, software metrics are one of the most effective tools available, highlighting areas of concern that may be fixed before the system is released to the public. In addition to this, some of the advantages of software measurement are as follows:

- Measurements provide unbiased data on the present state of development of a software product, technique, or resource, and they are used to make decisions regarding future development.
- Software measurement supports management strategy to make better and timely decisions.
- Metrics aid software developers in finding and addressing design flaws early in the SDC, hence reducing the likelihood of spectacular failures later.
- It helps to assess the quality of design and identify complex modules for further division of such modules to achieve improved and simplified design.
- Measurement provides a way to discover and correct the management issues early that can be more difficult or costly to resolve later.
- It directly addresses and aids the assessment of impact due to the changes in the process.
- This indicates that the functions of a module must be goal specified and therefore should concentrate on a single well-defined purpose. Cohesion is a term used in OOP to refer to the degree of intra-relatedness of functions inside a component.
- A module is considered to be fair if all of its properties are shared by all of its functions. Software cohesive is a metric that assesses the complexity of a program module in such a manner that high cohesion indicates low difficulty, as well as low cohesion, which indicates computational dimensionality in the program module.
- In OOP, cognitive metrics provide numerical weights to OO features to emphasize their underlying definition or qualities, as opposed to traditional metrics.

Conclusion

The MOOD metrics are the most appropriate metrics for evaluating object-oriented programmes, and they have been successfully applied to Java projects. The System is useful not only for assessing programmes, but also for finding where problems exist within each programme under examination. Weights are quite useful in adapting the marking system of the tested programmes to the suitable level. The system is easy to use and may be used to assess applications at the process level. A disadvantage of the technique is that it can only be used to analyse large Java programmes.

References:

[1] R. Pressman, Software Engineering: a Practitioner's Approach: European Adaptation, 5th edition, McGraw-Hill, UK, 2000.

[2] F. Abreu and R. Carapuça, Object-Oriented Software Engineering: Measuring and Controlling the Development Process, Proceedings of the 4th International Conference on Software Quality, McLean, VA, USA, 1994.

[3] F. Abreu, M. Goulão and R. Esteves, Toward the Design Quality Evaluation of Object-Oriented Software Systems, Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, 1995.

[4] F. Abreu and W. Melo, Evaluating the Impact of Object-Oriented Design on Software Quality, Proceeding of the 3rd International Software Metrics Symposium (METRICS’96), IEEE, Berlin, Germany, pp. 90-99, 1996.

[5] F. Abreu, S. Esteves and M. Goulao, The Design of Eiffel Programs: Quantitative Evaluation Using the MOOD Metrics, Proceedings of TOOLS'96, Santa Barbara, CA, USA, 1996.

[6] R. Harrison, S. Counsell and R. Nithi, An evaluation of the MOOD set of object-oriented software metrics, IEEE Transaction on Software Engineering, 24(6), 1998, pp. 491-496.

[7] K. Redish, and W. Smyth, Program Style Analysis: A Natural By-Product of Program Compilation, Communications of the ACM, 29(2), 1986, pp. 126-133.

[8] J. Al-Ja'afer, and K. Sabri, AUTOMARK++ an CASE tool to automatically mark student Java Programs., International Arab Journal of Information Technology, to appear.

[9] R. Berry, and B. Meekings (1985), A Style Analysis of C Programs, Communications of the ACM, 28(1), 1985, pp. 80-88.

[10] D. Jackson and M. Usher, Grading Student Programs Using ASSYST. Proceeding 28 the ACM SIGCSE Tech. Symposium on Computer Science Education, San Jose, California, USA, pp. 335-339, 1997.

[11] D. Jumaa, A Computer Model for Evaluation of Programs, Master Thesis, University of Engineering and Science, Iraq, 1992.

[12] A. Baroni, Formal Definition of Object-Oriented Design Metrics, Master Thesis, Universidade Nova de Lisboa, Portugal, 2002.

[13] Alqadi, B. S., & Maletic, J. I. (2020). Slice-Based Cognitive Complexity Metrics for Defect Prediction. *SANER 2020 - Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering*. https://doi.org/10.1109/SANER48275.2020.9054836

[14] Beniwal, R. (2015). Analysis of testing metrics for object oriented applications. *Proceedings - 2015 IEEE International Conference on Computational Intelligence and Communication Technology, CICT 2015*. https://doi.org/10.1109/CICT.2015.34

[15] Campbell, G. A. (2018). Cognitive complexity: An overview and evaluation. *Proceedings - International Conference on Software Engineering*. https://doi.org/10.1145/3194164.3194186

[16] Crasso, M., Mateos, C., Zunino, A., Misra, S., & Polvorín, P. (2016). Assessing cognitive complexity in Java-based Object-Oriented systems: Metrics and tool support. *Computing and Informatics*.

[17] De Silva, D. I., Kodagoda, N., Kodituwakku, S. R., & Pinidiyaarachchi, A. J. (2017). Analysis and enhancements of a cognitive based complexity measure. *IEEE International Symposium on Information Theory - Proceedings*. https://doi.org/10.1109/ISIT.2017.8006526

[18] De Silva, D. I., Weerawarna, N., Kuruppu, K., Ellepola, N., & Kodagoda, N. (2013). Applicability of three cognitive complexity metrics. *Proceedings of the 8th International Conference on Computer Science and Education, ICCSE 2013*.

https://doi.org/10.1109/ICCSE.2013.6553975

[19] Francis Thamburaj, T., & Aloysius, A. (2017). Models for Maintenance Effort Prediction with Object-Oriented Cognitive Complexity Metrics. *Proceedings - 2nd World Congress on Computing and Communication Technologies, WCCCT 2017*. https://doi.org/10.1109/WCCCT.2016.54

[20] Husein, S., & Oxley, A. (2009). A coupling and cohesion metrics suite for object-oriented software. *ICCTD 2009 - 2009 International Conference on Computer Technology and Development*. https://doi.org/10.1109/ICCTD.2009.209

[21] Ibrahim, S. M., Salem, S. A., Ismail, M. A., & Eladawy, M. (2012). Novel sensitive object-oriented cohesion metric. *2012 22nd International Conference on Computer Theory and Applications, ICCTA 2012*. https://doi.org/10.1109/ICCTA.2012.6523562

[22] Jakhar, A. K., & Rajnish, K. (2014). A new cognitive approach to measure the complexity of software's. *International Journal of Software Engineering and Its Applications*. https://doi.org/10.14257/ijseia.2014.8.7,15

[23] Jayalath, T., & Thelijjagoda, S. (2020). A modified cognitive complexity metric to improve the readability of object-oriented software. *Proceedings - International Research Conference on Smart Computing and Systems Engineering, SCSE 2020*. https://doi.org/10.1109/SCSE49731.2020.9313049

[24] Jha, S., & Ratha, B. K. (2018). OOMT-Object oriented metric technique towards predictive & qualitative software. *2017 International Conference on Infocom Technologies and Unmanned Systems: Trends and Future Directions, ICTUS 2017*. https://doi.org/10.1109/ICTUS.2017.8286069