Software product metrics used to build defect prediction models

Marian Jureczko
Institute of Computer Engineering, Control and Robotics,
Wrocław University of Technology,
Wyb.Wyspianskiego 27, 50370 Wrocław, Poland

Lech Madeyski
Institute of Informatics,
Wrocław University of Technology,
Wyb.Wyspianskiego 27, 50370 Wrocław, Poland

M. Jureczko and L. Madeyski, "Software product metrics used to build defect prediction models," SPR 2/2014, Faculty of Computer Science and Management, Wroclaw University of Technology, April 2011.
http://madeyski.e-informatyka.pl/download/JureczkoMadeyskiSoftwareProductMetrics.pdf
Rev. 1: 2011, Rev. 2: 2014

Abstract
This document presents software product metrics we usually use to build software defect prediction models.

1 Software product metrics

This document presents software product metrics we usually use to build software defect prediction models, especially those built by Madeyski and Jureczko [6, 5, 4].

- The metrics suite suggested by Chidamber and Kemerer [2]:
  - Weighted Methods per Class (WMC). The value of the WMC is equal to the number of methods in the class (assuming unity weights for all methods).
  - Depth of Inheritance Tree (DIT). The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top.
  - Number of Children (NOC). The NOC metric simply measures the number of immediate descendants of the class.
  - Coupling between object classes (CBO). The CBO metric represents the number of classes coupled to a given class (efferent couplings and afferent couplings). These couplings can occur through method calls, field accesses, inheritance, method arguments, return types, and exceptions.

  - Response for a Class (RFC). The RFC metric measures the number of different methods that can be executed when an object of that class receives a message. Ideally, we would want to find for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the transitive closure of the method call graph. This process can however be both expensive and quite inaccurate. Ckjm calculates a rough approximation to the response set by simply inspecting method calls within the class method bodies. The value of RFC is the sum of number of methods called within the class method bodies and the number of class methods. This simplification was also used in the original description of the metric.
  - Lack of cohesion in methods (LCOM). The LCOM metric counts the sets of methods in a class that are not related through the sharing of some of the class fields. The original definition of this metric (which is the one used in Ckjm) considers all pairs of class methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods do not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that do not share a field access the number of method pairs that do.

- Lack of Cohesion in Methods (LCOM3) suggested by Henderson-Sellers [3]. LCOM3 is a normalized version of the Chidamber and Kemerer's LCOM metric and can be calculated using the following equation:

$$LCOM3 = \frac{(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)) - m}{1-m} \tag{1}$$

  where m is the number of methods in a class; a is the number of attributes in a class and μ(A) is the number of methods that access the attribute A.

- The metrics suite suggested by Bansiy and Davis [1]:
  - Number of Public Methods (NPM). The NPM metric simply counts all the methods in a class that are declared as public. The metric is known also as Class Interface Size (CIS)
  - Data Access Metric (DAM). This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
  - Measure of Aggregation (MOA). The metric measures the extent of the part-whole relationship, realized by using attributes. The metric is a count of the number of class fields whose types are user defined classes.
  - Measure of Functional Abstraction (MFA). This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class. The constructors and the java.lang.Object (as parent) are ignored.

- Cohesion Among Methods of Class (CAM). This metric computes the relatedness among methods of a class based upon the parameter list of the methods. The metric is computed using the summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.

• The quality oriented extension to Chidamber & Kemerer metrics suite suggested by Tang et al. [9].

  - Inheritance Coupling (IC). This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods functionally dependent on the new or redefined methods in the class. A class is coupled to its parent class if one of the following conditions is satisfied:
    - ∗ One of its inherited methods uses an attribute that is defined in a new/redefined method.
    - ∗ One of its inherited methods calls a redefined method.
    - ∗ One of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method.
  - Coupling Between Methods (CBM). The metric measures the total number of new/redefined methods to which all the inherited methods are coupled. There is a coupling when at least one of the given in the IC metric definition conditions is held.
  - Average Method Complexity (AMC). This metric measures the average method size for each class. Size of a method is equal to the number of Java binary codes in the method.

• Two metrics suggested by Martin [7]:

  - Afferent couplings (Ca). The Ca metric represents the number of classes that depend upon the measured class.
  - Efferent couplings (Ce). The Ca metric represents the number of classes that the measured class is depended upon.

• Class level metrics built on the basis of McCabe's complexity metric [8] — Max(CC) (the greatest value of CC among methods of the investigated class) and Avg(CC) (the arithmetic mean of the CC value in the investigated class). McCabe's cyclomatic complexity (CC) is equal to number of different paths in a method (function) plus one. The cyclomatic complexity is defined as: $CC = E - N + P$; where E - the number of edges of the graph, N - the number of nodes of the graph, P - the number of connected components. CC is the only method size metric. For the sake of prediction models class level metrics Max(CC) and Avg(CC) have been derived.

• Lines of Code (LOC).

References

[1] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4–17, 2002.

[2] Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[3] Brian Henderson-Sellers. *Object-oriented metrics: measures of complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[4] Marian Jureczko and Lech Madeyski. Predykcja defektów na podstawie metryk oprogramowania - identyfikacja klas projektów. In *Inżynieria Oprogramowania w procesach integracji systemów informatycznych*, pages 185–192. Wydawnictwo Komunikacji i Łączności, 2010. Draft: http://madeyski.e-informatyka.pl/download/JureczkoMadeyski10e.pdf.

[5] Marian Jureczko and Lech Madeyski. Towards identifying software project clusters with regard to defect prediction. In *PROMISE'2010: Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pages 9:1–9:10. ACM, 2010. Draft: http://madeyski.e-informatyka.pl/download/JureczkoMadeyski10f.pdf.

[6] Lech Madeyski and Marian Jureczko. Which Process Metrics Can Significantly Improve Defect Prediction Models? An Empirical Study. *Software Quality Journal*, 2014 (DOI: 10.1007/s11219-014-9241-7) (accepted).

[7] Robert C. Martin. OO Design Quality Metrics: An Analysis of Dependencies. In *OOPSLA'94: Proceedings of Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*, pages 1–8. Object Mentor, Inc., 1994.

[8] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[9] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An empirical study on object-oriented metrics. In *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, page 242, Washington, DC, USA, 1999. IEEE Computer Society.