On the Relationship between Cyclomatic Complexity and the Degree of Object Orientation

Grégory Seront[1], Miguel Lopez[1], Valérie Paulus[1], Naji Habra[2]

[1] CETIC asbl
Rue Clément Ader, 8
B-6041 Gosselies, Belgium
{gs, malm, vp}@cetic.be
[2] Faculty of Computer Science
University of Namur – FUNDP
Namur, Belgium
nha@info.fundp.ac.be

Abstract. In this paper we explore the relationship between the degree of object orientation of a software entity and the cyclomatic complexity. This work is part of a broader research aiming at validating the usage of this popular complexity measure in object oriented contexts. In a previous work we showed that the shape of the distribution of the cyclomatic complexity differs according to the programming paradigm in which it is used. We hypothesized that this was due to the fact that a part of the complexity was hidden within object oriented mechanisms. This paper aims at validating this hypothesis.

1 Introduction

Despite its old age, the McCabe's cyclomatic complexity number (CCN) (McCabe, 1976) is still widely used in industrial contexts. It is notably referred to in many quality guides such as the ones from the US Department of Defense, the NASA and the European Space Agency (DOD, 1996), (ESA, 1995), (NASA, 1994).

CCN has been defined in the seventies in order to measure complexity in a procedural paradigm context. Today it is used to measure complexity in object oriented (OO) contexts.

In (Lopez ,2005), we showed that the distribution of the CCN is different in OO and procedural contexts. We showed that the methods in software based on OO paradigm tends to have a lower CCN that the procedure of procedural software. This led us to question the pertinence of CCN in OO contexts and to formulate hypotheses to explain this phenomenon.

The work we present here aims at investigating the hypothesis that this decrease in complexity is due to the fact that a part of it is deferred to OO mechanism such as inheritance and polymorphism. If this hypothesis is true, the less the OO mechanisms are used, the more complex the methods should be (all other things being equals). In

other words, the less "Object Oriented" a program is the greater the CCN of its methods should be.

In section 2, we define the cyclomatic complexity and explain how we think the OO mechanisms decrease it. In section 3, we precise what we mean by degree of object orientation and define how we will measure it.
Section 4 describes the experimental settings we used and in section 5 we explain the results of our measures.

2 Cyclomatic Complexity

McCabe's cyclomatic complexity number (CCN) defined in (McCabe, 1976), aims at measuring the complexity of the instruction control flow inside a method or module. It is computed from the control flow graph of the module which represents the topology of the control flow. The following formula defines this complexity:

$$CCN = E - N + p\,, \tag{1}$$

where E is the number of edges in the graph, N the number of nodes and p is the number of connected components.

What is captured by CCN is the number of linearly independent circuits within the method. The higher this number, the more complex it is to understand, test and modify the measured object.

This measure was defined in the seventies with structured procedural languages in mind and has been very popular ever since.

In (Lopez, 2005), the pertinence of this measure for object Oriented (OO) languages is questioned. We showed that the distribution for CCN exhibits significant differences between procedural and OO languages. In OO contexts, the distribution is much more concentrated towards low complexity values with more than 80 percents of the occurrences having complexity of 1 or 2.

We hypothesized that the shape of the distribution was due to the fact that a large portion of the complexity is hidden in object oriented mechanisms such as polymorphism and overloading. These mechanisms are in fact hidden decision nodes.

Let us suppose for instance that we have the two following method definitions within the same class

```
ClassA::Process( int a){…};

ClassA::Process(String a)…};
```

Then if we look at the following call:

```
ClassA A;

A.Process(b);
```

We could explicitly represent the hidden decision node the following way:

```
Class A::ProcessInt(int a);

Class A::ProcessString(String a);

If TYPEOF(b) = int then

          ProcessInt(b);

Elsif TYPEOF(b)=String then

          ProcessString(b);

Endif;
```

From this hypothesis, we defined in (TR-SC-2,2005) a new measure of complexity taking these OO constructions into account.

These findings also drove us to the following hypothesis:

***Hypothesis**: McCabe's cyclomatic complexity is inversely correlated to the degree of object orientation*

The reminder of this paper clarifies what we mean by "degree of OO" and describes the experiments we conducted to validate or invalidate our hypothesis.

3 Measuring the Degree of Object Orientation

The degree of object orientation of a software code can be informally defined as the degree with which the constructions introduced by the OO paradigm are used. These concepts are (Meyer, 1997).

- Data Abstraction
- Encapsulation
- Polymorphism
- Inheritance

From these concepts, several OO measures and suites of OO measures (Chidamber, 1991), (Brito, 1995) have been defined. These measures aim at capturing how and how much these constructs are used inside a given software code. They all measure the degree of object orientation from a different angle. So far there is no such thing in the literature as a metric representing the global "degree of object orientation" of a software entity. Defining one in this paper would be surely interesting but out of scope. This is why we restrict here our investigations to the most important concept introduced by Object Orientation: the inheritance. Data abstraction and encapsulation are important structuring concepts, but they are not as likely to reduce the degree of

complexity as inheritance.

The measure we use to capture inheritance is Chidamber and Kemerer Depth of Inheritance Tree (DIT) for a class (Chidamber, 1991). This measure is defined as the longest path from the class to the root of the inheritance hierarchy tree. This is a good indicator of object orientedness. If no inheritance is used, all classes will have a DIT of 0. It can also be argued that the deeper we are in the inheritance tree, the more complexity can potentially be deferred to classes higher in the hierarchy.

4 Experimental Settings

The measures computed in this study are the McCabe cyclomatic complexity and the Depth of Inheritance Tree (DIT).

Cyclomatic complexity (McCabe, 1976) is used to evaluate the complexity of an algorithm in a method. It is a count of the number of test cases that are needed to test the method comprehensively.

A method with a low cyclomatic complexity is generally better. This may imply decreased testing and increased understandability (Gill, 1991), (Heimann , 1994), (Kafura , 1987).

The inheritance is a programming mechanism introduced by the emergence of the object-oriented paradigm. Inheritance can be defined as follows *a relationship among classes, wherein an object in a class acquires characteristics from one or more other classes*.

The depth of inheritance aims at measuring "how much" the inheritance is used. So, the deeper a class is within the hierarchy, the greater the number methods it is likely to inherit making it more complex to predict its behavior.

Both measures are related to classes. However, the McCabe cyclomatic measure has been defined at the method level. So, the weighted method per class (WMC) has been used, which computes the McCabe cyclomatic measure of each method of a given class by summing them.

In this work, we build a sample of 694 software Java projects. Most of the products (694) are coming from the Open Source community, i.e. Sourceforge (Sourceforge, 2005) and a very low number of products (3) are coming from the industry. So, the projects that populate this sample are real software systems and in that sense, it can be argued that we have a representative sample.

But, which population does this sample represent exactly? Before answering this question, it is important to solve the three following problems.

Firstly, which is the entity related to cyclomatic complexity? The McCabe cyclomatic complexity related entity (i.e. the entity being measured) for Java programs is the "method within a class". Concretely (at the measurement tool level), the Cyclomatic Complexity number is related to a method. However, the inheritance depth is related to the class entity. So, the measure used for the complexity of a class

is actually the weighted method per class. So, the population studied here is the Java classes.

Secondly, most of the products of the sample are labeled in the Sourceforge website as production software. That is, these products are mature enough to be deployed in a production environment. Actually, this quality of the software products is not considered as relevant, since the own members of the project freely assign the production label to their product. In that context, the verification of such quality remains difficult.

Thirdly, the open source characteristic can be considered as a weakness of the sample. Indeed, it is often accepted that open source and closed source (industrial) software are two different types of software. Both projects are so different in terms of process that the resulting products cannot be regarded as similar software. This implies that mixing both "types" can lead us to draw incorrect conclusions. Basically, in the scope of current paper, it is assumed that this distinction (close source vs open source) is not relevant. In other words, close and open source programs belong to the same category in terms of cyclomatic complexity. It is accepted that the independent variables close source and open source do not have any impact on the dependent variable cyclomatic complexity number.

So, our sample represents a population of Java classes. The "maturity level" of the product and the "open source label" are not seen as qualities of the entity class that can have a serious influence on the cyclomatic complexity number. Both working assumptions are set up in order to facilitate the current work.

Table 1 shows a summary of the Java classes sample. The methods considered in this sample are the method with a non-empty body. A method with a non-empty body is a method whose Halstead program length is greater than 0. The Halstead program length takes into account the number of operators and operands

Only concrete methods with a non-empty body have been considered in this study. Indeed, empty methods would seriously enhance the proportion of the method with a cyclomatic complexity number of 1, whereas the real cyclomatic number value of such methods would have to be non-applicable.

Table 1. Java Sample Summary

Number of Products	694
Number of Classes	80136
Number of non-empty Methods	695741

The current empirical study is organized in 5 steps:

- Automatic download of the 694 software products from SourceForge done by a python script: each project is copied into a directory whose name is the name of the product.
- Computing of the WMC and inheritance depth: a python script launches the tool such that, for each product, the cyclomatic complexity number is computed. The measurement tool JStyle can be launched with a command line interface.

- JStyle generates a text file, which contains the measurement result. So, each file corresponds to one software product.
- Another python script merges all the text files into one single file.

The merged file is loaded in R software wherein the frequency distribution of the cyclomatic complexity number is computed.

The tested hypothesis of the current study can be stated as follows:

Table 2. Tested Hypotheses

H0: rho(WMC, ID) = 0
H1: rho(WMC, ID) < 0

So, it is expected that the correlation between both measures is negative. It would mean that the more a class is complex (in the sense of cyclomatic complexity), the more the inheritance depth is low. This assumption means that the complexity is partially distributed in the inheritance tree.

5 Results

The correlation coefficients computed are the Pearson ,Spearman and Kendall coefficients. The three correlations coefficients have been computed since the scale of both used measures is not clearly identified Indeed, Pearson and Kendall are applicable with measures with a ratio scale, whereas Spearman is used when the measure has an ordinal scale.

So, in order to be as careful as possible, the three coefficients have been computed. Since the scale has at least to be ordinal, at worst, only the Spearman coefficient will be valid from a theoretical standpoint. Since all three coefficients give the same indication, we can draw valid conclusions concerning the correlation.

The software package used to compute the correlation coefficients is the R Stat software package (R, 2005).

The procedure of the computing is three-fold:

- Load the data from a csv file
- Generate a sample with 5000 observations without replacement
- Compute the three correlation coefficients

Table 3 shows the results of the correlation tests.

Table 3. Correlations Coefficients

	Score	p-value
Kendall	-0.008390372	0.1868
Pearson	-0.0073102	0.3027
Spearman	-0.01101954	0.218

Each statistical test has an associated null hypothesis, the p-value is the probability that your sample could have been drawn from the population(s) being tested (or that a more improbable sample could be drawn) given the assumption that the null hypothesis is true. A p-value of .05, for example, indicates that you would have only a 5% chance of drawing the sample being tested if the null hypothesis was actually true.

Null Hypothesizes are typically statements of no difference or effect. A p-value close to zero signals that your null hypothesis is false and typically that a difference is very likely to exist. Large p-values closer to 1 imply that there is no detectable difference for the sample size used. A p-value of 0.05 is a typical threshold used in industry to evaluate the null hypothesis. In more critical industries (healthcare, etc.) a more stringent, lower p-value may be applied.

So, it seems that the null hypothesis is satisfied for the three tests, that is, Kendall, Pearson, and Spearman. Furthermore, the p-value for each test is greater than 0.05. And in that sense, the null hypothesis can be considered as true.

6 Conclusion

From the results of the experiment we conducted, we observed no significant correlation between the depth of inheritance of a class and its weighted method complexity. We designed this experiment to validate the hypothesis that a part of the complexity of OO programs is hidden inside OO constructs. Before jumping to the conclusion that our hypothesis is false, several remarks have to be made:

Firstly, there might be two opposite effects canceling each other out linked to the depth of inheritance:

a) The deeper a class is within the inheritance tree it belongs to, the more specialized and thus complex (Benlarbi , 2004) it tends to be.

b) The deeper a class, the more chance we have to defer its complexity to other classes higher in the hierarchy.

Secondly, regarding the results of this empirical study, the inheritance factor cannot be considered as an important factor that reduces the cyclomatic complexity. Indeed, other object-oriented features are better candidate for reducing the cyclomatic complexity. For instance, the polymorphism can be considered as an object-oriented

mechanism which allows deferring decision node by overriding the methods. And, at run-time, the decision is made concerning which is the suited implementation of the methods which is selected.

At the very beginning of this work, the classical object-oriented features were identified as good candidate to redistribute the cyclomatic complexity in Java programs. Now, according to the current study, it is obvious that the inheritance is not the mechanism which can affect the cyclomatic complexity.

We accept that polymorphism factor would be a better factor that can explain the low scores of the cyclomatic complexity in a Java context.

Thirdly, the cyclomatic complexity number is related to a specific kind of a complexity, that is, the complexity related to the control flow structure. Since the inheritance mechanism can increase the complexity of a class – considering the all the parents of a given class, it seems that the McCabe number does not capture the inheritance-related complexity. Therefore, the correlation coefficient is closer to zero.

So, it seems that object-oriented programs can hold several types of software complexity. And, it could be interesting to construct a taxonomy of the software complexity with rigorous definitions in order to refine the software complexity measurement.

7 Acknowledgement

This research project is supported by the European Union (ERFD) and the Walloon Region (DGTRE) under the terms defined in the Convention n° EP1A12030000072-130008.

8 References

1. (Benlarbi , 2004) Benlarbi, S., El-Emam, K., Goel, N., and Rai, S., “Thresholds for Object-Oriented Measures”, National Research Council Canada Institute for Information Technology, 2004
2. (Brito, 1995) F.Brito e Abreu, "The MOOD Metrics Set," Proc. ECOOP'95 Workshop on Metrics, 1995
3. (Chidamber, 1991) Chidamber S. and Kemerer C., "Towards a Metrics Suite for Object Oriented Design", Proceedings of OOPSLA'91, pp.197-211, 1991.
4. (DOD, 1996) Dept of Defense (DOD), Dept of the Army, Practical Software Measurement, Version 2.1, 3/96
5. (ESA, 1995) Guide to the software operations and maintenance phase, ESA PSS-05-07 Issue 1 Revision 1 March 1995
6. (Gill, 1991) Gill, G., and Kemerer, C., “Cyclomatic Complexity Density and

Software Maintenance Productivity," IEEE Transactions on Software Engineering, December 1991.

7. (Heimann , 1994) Heimann, D., "Complexity and Defects in Software—A CASE Study, "Proceedings of the 1994 McCabe Users Group Conference, May 1994.
8. (Kafura , 1987) Kafura, D., and Reddy, G., "The Use of Software Complexity Metrics in Software Maintenance," IEEE Transactions on Software Engineering, March 1987.
9. (Lopez, 2005) M. Lopez, N. Habra, "Relevance of the Cyclomatic Complexity Threshold for the Java Programming Language", 2nd SOFTWARE MEASUREMENT EUROPEAN FORUM (SMEF2005) Roma, Italy, 2005.
10. (McCabe, 1976) Thomas J. McCabe, A measure of complexity, IEEE transaction on software engineering, Vol SE-2, No 4, December 1976
11. (Meyer, 1997) Meyer, B., "Object-Oriented Software Construction", Prentice Hall, ISE Inc., Santa Barbara, USA
12. (NASA, 1994) Software Engineering Laboratory, Software Measurement Guidebook, SEL-94-002, NASA.
13. (R, 2005), http://www.r-project.org/
14. (Sourceforge, 2005) http://www.sourceforge.net
15. (TR-SC-2, 2005) Lopez, M, Poels, G., Habra, N., Seront, G., "Measuring Software Complexity in an Object-Oriented Context", Technical Report, http://www.cetic.be , 2005