A Practical Model for Measuring Code Complexity of PHP

Cho Thet Mon, Khin Mar Myo
University of Computer Studies, Mandalay
chothetmonucsm@gmail.com,kmmyo.ag@gmail.com

Abstract

Software quality metric for PHP is scarce in literature. This paper presents a technical and research overview of software quality assessment especially for software complexity to establish a software quality observatory for PHP language since PHP is one of the most popular languages that millions of websites and web applications are developed every month using PHP. Complexity is one important quality for software source codes. In this paper, software metric related to code complexity is investigated and a suitable set of metric is identified for the given system. Here, we present motivating examples, tool and techniques that can be used to evaluate the quality of source code. We have carried out an empirical study and tried to find out the nature of relationship between the metric and other well-known metric. In other words, it has been investigated whether complexity features may have positively or negatively on the code maintainability effort.

1. Introduction

Computer software has become a driving force and is used in all sorts of everyday activities. At the same time, the demand for sophisticated and flexible software also increases. Currently, software market is driven by urgent market needs which drive software developers to produce software without delay in delivery. Such urgency causes a lot of problems in producing quality software. In addition, software maintenance becomes extremely difficult. In most cases, the delivered product is not reliable. Hence, quality assurance, customer satisfaction and reliable products are immediate needs of current software industries. The intent of software engineering is to provide a framework for building software with higher quality [8].

Software measurement gives an excellent opportunity for software developers to evaluate their own products, to convince themselves the outcome of the software development process and to estimate or predict the efforts required for a future product. Research on software metrics is an ongoing process for several decades. Software metrics are helpful in several activities of the software development life cycle, and therefore contribute to the overall objectives of software quality.

Controlling and minimizing software complexity is the most important objective of each software development paradigm because it affects all other software quality attributes including reusability, reliability, testability and maintainability. For this purpose, a number of software complexity measures have been reported to quantify different aspects of complexity.

Software complexity is traditionally a direct indicator of software quality and cost [1-5]. The greater the complexity (by some measure), the more fault prone the software resulting in higher cost. Logically, many of these measures have been shown to be correlated in some manner. Understanding these relationships is important to understand and evaluate the metrics themselves and ultimately in reducing software development and maintenance efforts.

If complexity could somehow be identified and measured, then software developers could adjust development, testing, maintenance procedures and effort accordingly. This concern has motivated several researchers to define and validate software complexity measures and establish relationship between software complexity and maintenance effort [23, 19, and 22].

Earlier complexity metrics are not sufficient to determine complexity of the class. One of the problems of Cyclomatic complexity is that it only considered control flow complexity and it ignored unstructured properties. Halstead complexity is based on the assumption that a program is made only of operators and operands and it ignored other properties. Moreover, weighted method per class (WMC) is only a partial view of complexity. In this respect, a more complete model of program complexity is introduced with the increased spread of Object-Oriented programming for the need of a metric suite that could take into consideration the complexity of Object-Oriented structure. The system contributes new set of metrics for PHP codes especially for object-oriented PHP to develop a tool that can automatically collect complexity attributes, to investigate the relation between the proposed software metric and other indicator of software quality and to assess and evaluate the effectiveness and usefulness of proposed metrics theoretically and empirically.

2. Related Works

Software complexity measures attempt to quantify intuitive notions of program complexity. Most software complexity measures are based on measuring a single program characteristic which is deemed the most significant factor contributing to the conceptual complexity of a program.

Various OO complexity and quality metrics have been proposed and their reviews are available in the literature. Rajnish et al [10] has studied the effect of class complexity (measured in terms of lines of codes, distinct variables names and function) on development time of various C++ classes. Kulkarni et al [11] presents a case study of applying design measures to assess software quality. Sanjay et al [17] applied their proposed metric on a real project for empirical validation and compared it with Chidamber and Kemerer metrics suites [14] and their theoretical, practical, empirical validations and the comparative study to prove the robustness of the measure. Alshayeb and Li have presented an empirical study of OO metrics in two processes [12]. They predict that OO metrics are effective in predicting design efforts and lines of source code added, changed and deleted in one case and ineffective in other. Emam, Benlarbi, Goel and Rai validate the various OO metrics for effects of class size [6]. Yacoub et al [20] defined two metrics for object coupling (Import Object Coupling and Export Object Coupling) and operational complexity based on state charts as dynamic complexity metrics. The metrics are applied to a case study and measurements are used to compare static and dynamic metrics. Jagdish et al [7] described an improved hierarchical model for the assessment of high-level design quality attributes in OO design. In their model, structural and behavioral design properties of classes, objects, and their relationships are evaluated using a suite of OO design metrics. Their model relates design properties such as encapsulation modularity, coupling and cohesion to high-level quality attributes such as reusability, flexibility, and complexity using empirical and anecdotal information. Munson et al [3] showed that relative complexity gives feedback on the same complexity domains that many other metrics do. Thus, developers can save time by choosing one metric to do the work of many.

3. Software Complexity

Software complexity is the most important attribute of a software product. It influences in various degrees, all software quality characteristics, starting from maintainability, reliability, reusability, testability. Developing software products to meet current business needs already means developing complex systems, because of the complexity of the business that has to model.

Code complexity metrics play an important role in software development process. But despite of numerous studies many questions are still open partially due to new programming languages appearance, design paradigm evolutions and lack of factual material. The main objectives of this work are to obtain quality metrics for a number of small and medium real world PHP languages, to assess the complexity quality of the language and to identify the relations between complexity and maintenance effort.

3.1. Software Metric Tool Selection

The importance of measuring and controlling the quality of the source code has determined the development of tools which can measure software metrics automatically based on the source code. These tools, either integrated in each developer's IDE or used separately, are applied on a regular basis to identify the deviations of the metrics from the ranges initially defined.

The accepted values for the metrics are defined based on the specific project requirements, company quality criteria or industry best practices. Depending on the metrics required for a project, one or more tools can be used.

Several such open source tools for measuring software metrics are available on sourceforge. Net portal [24]. There are small projects, standalone applications or plug-ins for various IDE products like Eclipse, Net Beans, IDEA or Visual Studio; they support one or more programming languages, like C, C++, Java, C#. For finding a set of suitable software metric tools, we conducted a free search on the internet.

In the proposed system, PHPMD tool will be used to obtain required attributes for code complexity. We chose to analyze PHP class files and the detail of PHPMD can be viewed in [25].

3.2. Attribute Selection and Proposed Metric

The original usage of PHPMD tool is to scan PHP source code and to find potential problems such as possible bugs, dead code, suboptimal code and overcomplicated expressions by using the rules.

Although there are 31 rules that can measure from PHPMD, we only emphasized on which factors are responsible for the complexity of the code. This tool is run from command line window to test PHP cod files and xml files are exported.

These files are parsed and extracted using DOM parser and we select some rules that can affect code complexity according to several research papers [13, 2, 21 and 9] and a new complexity metric is introduced based on the following factors:

Weighted Method per Class (WMC) - number of class methods' complexity

Nested block depth (NBD) - number of class methods' NBD

Number of parameters (PAR) - number of class methods' parameters

Depth of inheritance tree (DIT) - number of ancestor classes measured from the hierarchy root (class object for PHP)

Number of children (NC) - number of direct subclasses of a class

Attribute Complexity (AC) - number of attribute used in class file

$$CM_{php} = \frac{\sum_{i=1}^{j} WMC + \sum_{i=1}^{k} NBD + \sum_{i=1}^{l} PAR + \sum_{i=1}^{m} DIT + \sum_{i=1}^{n} NC + \sum_{i=1}^{o} AC}{ELOC} \quad (1)$$

Where CM_{php} = Complexity metric of PHP

j= number of WMC

k= number of NBD

l= number of PAR

m= number of DIT

n= number of NC

o= number of AC

The attributes were summed up to obtain the proposed complexity metric, based on the assumption that "measures must be additive (i.e., if two independent structures are put into sequence, then the total complexity of the combined structure is simply the sum of the complexities of the independent structures)" [16].

In Figure 1, a sample PassResult.php code is shown and the required quality attributes will be extracted. PHPMD tool is run to test this code and the following xml file is reported.

```php
<?php
class PassResult {
                var $passResult;
                var $failResult;
                private static function showResult() {
                                echo "Call pass::showResult()\n";
                }
                public static function do_showResult() {
                                 PassResult::showResult();
                }
}
PassResult::do_showResult();

class FailResult {
                public static function showResult() {
                                echo "Call fail::show()\n";
                                PassResult::showResult();
                }
}
failResult::showResult();
echo "Done\n";
?>
```

Figure 1. A sample PassResult.php code

```xml
<? xml version="1.0" encoding="UTF-8" ?>
<pmd>  <filename="D:\workspase\PHPQualityMetrics\PassReslt.php>
<violationbeginline="3"endline="13"rule="TooManyFields"
ruleset="CodeSizeRules" class="PassResult" priority="3">
Attribute Complexity 2
</violation>
<violationbeginline="3"endline="13"rule="ClassComplexity"
ruleset="CodeSizeRules" class="PassResult" priority="3">
Method Complexity 2
</violation>
<violationbeginline="6"endline="8"rule="CyclomaticComplexity"
rueset="CodeSizeRules" class="PassResult"
method="showResult" priority="3">
Cyclomatic Complexity  1
</violation>
<violationbeginline="6"endline="8"rule="NPathComplexity"
ruleset="CodeSizeRules" class="PassResult"
method="showResult" priority="3">
NPath Complexity 1
</violation>
<violationbeginline="6"endline="8"rule="ParameterList"
ruleset="CodeSizeRules" class="PassResult"
method="showResult" priority="3>
No of Parameters 0
</violation>
<violationbeginline="10"endline="12"rule="CyclomaticComplexity"rule
set="CodeSizeRules" class="PassResult"
method="do_showResult" priority="3">
Cyclomatic Complexity 1
</violation>
<violationbeginline="10"endline="12"rule="NPathComplexity"
ruleset="CodeSizeRules" class="PassResult"
method="do_showResult" priority="3">
NPath Complexity 1
</violation>
<violationbeginline="16"endline="21"rule="ClassComplexity"
ruleset="CodeSizeRules" class="FailResult"priority="3">
Method Complexity 1
</violation>
<violationbeginline="16"endline="21"rule="CouplingBetweenObject"ru
leset="DesignRules"  class="FailResult" priority="2">
Coupling between Objects Value 1
</violation>
<violationbeginline="17"endline="20"rule="CyclomaticComplexity"rule
set="CodeSizeRules" class="FailResult"
method="showResult" priority="3">
Cyclomatic Complexity 1
</violation>
<violationbeginline="17"endline="20"rule="NPathComplexity"
rulest="CodeSizeRules" class="FailResult"
method="showResult" priority="3">
NPath Complexity 1
</violation>
</file> </pmd>
```

Figure 2. XML file report

DOM parser is used to parse the required attributes from Figure 2 and complexity is computed according to (1).

3.3. Analysis

Many research papers used MI as maintainability indicator to validate and predict the maintainability of their proposed metrics [15].

Maintainability Index is a software metric which measures how maintainable (easy to support and change) the source code is. It is calculated as a factored formula consisting of Lines of Code, Cyclomatic Complexity and Halstead volume that is shown in (2).

$$MI = 171 - 5.2 * \ln(V) - 0.23 * CC(g) - 16.2 * \ln(LOC) \quad (2)$$

Where V= Halstead Volume

CC (g) = Cyclomatic Complexity per module

LOC = Lines of Code per module

If maintainability of system is better, its maintainability index should be higher and vice versa.

In [12], there are a lot of studies done on measuring software metrics and analyzing the correlation between them to determine the way software characteristics are influencing each other and are influenced by software complexity. According to [21], more complexity implies more possibility of faults and hence less quality.

Complexity can lead to subtle vulnerabilities that are difficult to test and diagnose [4], providing more chances for attackers to exploit. Complex code is difficult to understand, maintain, and test [20]. Therefore, complex code would be more difficult to maintain than simple code.

In software engineering, empirical study involves introducing assumptions or hypotheses about observed phenomenon, investigating of the correctness of these assumptions and evolving it into body knowledge. In order to validate any metric as an indicator of software quality, experimental hypotheses are tested to confirm or refute relationship between two or more variables.

From this reasoning, the following hypothesis was proposed to identify whether there is a consistent relation between complex code and maintainability index or not:

H1: The higher the software complexity, the more difficult it is to understand its source code for maintenance. This leads to a decrease in the maintenance effort [16 and 18].

The hypothesis will be evaluated by a set of statistical analysis techniques to clearly understand the relationship between maintainability and the defined metric.

For hypothesis H1, if the correlation is significant at the p-value 0.05 level or $p<0.05$, we will we accept the hypothesis H1 otherwise we will reject H1.

To verify this, each of the characteristics studied and put in relation with software complexity and then evaluated through a set of software metrics. In the system, 120 PHP programs are used to measure complexity and maintainability index for these program are also calculated. Then, the relation between them is studied by applying correlation indicator, PEARSON coefficient being one of them.

The correlation coefficient is a numerical value between -1 and 1 that expresses the strength of the linear relationship between two variables. When r is closer to 1, it indicates a strong positive relationship. A value of 0 indicates that there is no relationship. The value close to -1 signals a strong negative relationship between the two variables.

Putting in relation indicator CM_{php} and MI, PEARSON coefficient has the following value:

$$r(CM_{php}, MI) = \frac{\sum CM_{php} MI - \frac{\left(\sum CM_{php}\right)\left(\sum MI\right)}{n}}{\sqrt{\left(CM_{php}{}^{2} - \frac{\left(\sum CM_{php}\right)^{2}}{n}\right)\left(\sum MI^{2} - \frac{\left(\sum MI\right)^{2}}{n}\right)}} = -0.29266 \quad (3)$$

Where, CM_{php}= Complexity Metric of PHP Files

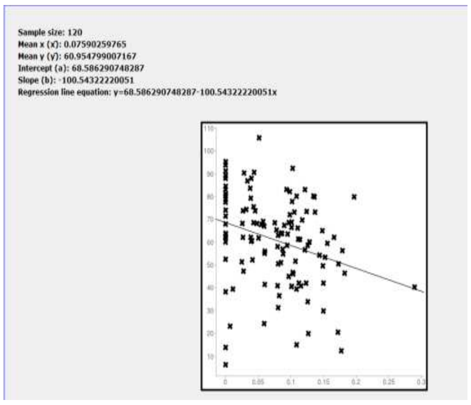
MI= Maintainability Index

n= Number of Pairs

r= Correlation Coefficient

The value in (3) will indicate that there is a negative correlation between code complexity and MI from source code defined through PHPMD. Moreover, we need to use critical value table to determine weak or strong correlation between variables. To do so, degree of freedom is computed. For a correlation study, the degree of freedom is equal to 2 less than the number of subjects we have. And then, critical value table is used to find intersection of alpha and degree of freedom. In [26], it shows critical values and how to use critical value table. According to result, r (120) =-.029266, $p<0.05$ means that for 120 PHP files, there is significant relationship between complexity metric and MI. That is because p value less than 0.05 means that our correlation coefficient is less than critical value on the table and we can be 95% confident that the relationship exits. This statement supports our hypothesis H1. The scatter plot of their relation is shown in Figure 1.

In this respect, the hypothesis H1 accepts: there is a statically significant relationship between complexity and MI at r=-0.29266.



```
Sample size: 120
Mean x (x̄): 0.07590259765
Mean y (ȳ): 60.954799007167
Intercept (a): 68.586290748287
Slope (b): -100.54322220051
Regression line equation: y=68.586290748287-100.54322220051x
```

Figure 3. Scatter plot for correlation between complexity and MI

The more complex a piece of software, the more effort is required to maintain it. The higher the software complexity, the more difficult it is to understand its source code for maintenance and evolution purposes. Hypothetically, complexity metric has been shown to have a strong negative correlation with MI.

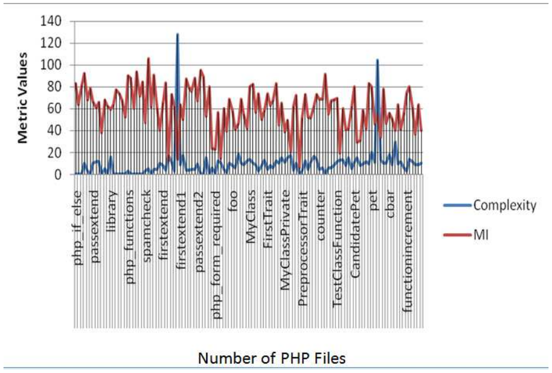


Figure 4. Comparison results of complexity and MI

A graph which covers comparison between complexity and MI values is also plotted in Figure 4 to observe the relation between them. As shown the graph in Figure 4, it is evident that the proposed complexity metric gives result which is opposite trends to the results given by MI. In other words, when code complexity is increase, MI value is decrease.

4. Conclusion and Future Works

In this paper, code complexity metric for PHP is introduced using PHPMD tool. By using our system, developer can easily assess the quality of PHP code and the features of PHP code. This may help to developer as an automated quality assessment tool to measure complexity of PHP code.

We have carried out an empirical study using statistical method and tried to find out the nature of relationship between the metric and code maintainability. In other words, it has been investigated whether the complexity metric is significantly associated with easy to maintain or not. Sample 120 PHP class files have been taken from the web and used for this purpose. More similar type of studies must be carried out with large data sets to get an accurate measure. We plan to replicate our study on large data set for different types of open source software system.

References

[1] A. Ganpati, "*Maintainability Index over Multiple Releases: A Case Study PHP Open Source Software*", International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 6, August – 2012

[2] C. Adrian, PhD, University Lecturer, "*On Measuring Software Complexity*", Academy of Economic Studies, Bucharest, Romania

[3] C.J. MunsonX and M.T. Khoshgoftaar, "*Measuring Dynamic program Complexity*", IEEE Software, Vol. 9, No. 6, 1992, pp. 48-55

[4] D. Coleman, D. Ash and H. Packard, "*Using Metrics to Evaluate Software System Maintainability"*

[5] D. Stavrinoudis, "Relation between Software Metrics and Maintainability"

[6] EL.K. Emam, S. Benlarbi, N. Goel and N.S. Rai, "*The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics*", IEEE Trans. Software Eng., Vol. 27, No.7, 2001, pp. 630 – 650.

[7] J. Bansiya and C.G. Davis, "*A Hierarchical Model for Object-Oriented Design Quality Assessment*", IEEE Transaction on Software Engineering, Vol.28, No. 1, 2002, pp. 4-17

[8] Kaiserslautern, H. Dieter Rombach, "*The impact of design complexity on software cost and quality*", Germany, 2010

[9] K.D. Mrinal, D. Swapan, D.Nikhil, C. Kunal and J. Anupam, "*A Review and Analysis of Software Complexity Metrics in Structural Testing*"

[10] K. Rajnish and V. Bhattacherjee, "*Complexity of Class and Development Time*": Journal of theoretical and Applied Information Technology (JATIT), Asian Research Publication Network (ARPN), Scopus (Elsevier) Index, Vol. 3, No. 1, 2006, pp. 63-70.

[11] L. Kulkarni, R.Y. Kalshetty and V.G. Ard , "*Validation of CK metrics for Object-Oriented Design Measurement*", Proceedings of third international conference on Emerging Trends in Engineering and Technology, IEEE Computer Society, 2010, pp. 646-651

[12] M. Alshayeb and W. Li, "*An Empirical Validation of Objec-Oriented Metrics in Two Different Iterative*

Software Processes", IEEE Trans. on Software Engineering, Vol. 29, No. 11, 2003,pp.1043 – 1049.

[13] P. Abdul Jabbar and S. Sarala ,"*Advanced Program Complexity Metrics and Measurement*"

[14] R.S. Chidamber and F.C. Kemerer, "*A Metric Suite for Object-Oriented Design*", IEEE Transaction on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493

[15] R. Victor Basili, L. Briand, et al., "*A Validation of Object-Oriented Design Metrics as Quality Indicators*"

[16] S. Henderson, "*Object Oriented Metrics. Measures of Complexity*", Prentice Hall PTR, Upper Saddle River, New Jersey, 1996.

[17] S. Misra , I. Akman and M. Koyuncu, "*An inheritance complexity metric for object-oriented code :A cognitive approach*", Indian Academy of Sciences, Vol. 36, Part 3, 2011, pp. 317–337.

[18] S. Nasib Gill and S. Sunil, "*Correlating Dimensions of Inheritance Hierarchy with Complexity & Reuse*"

[19] S. R. Chidambera and C.F. Kemerer, "*A Metrics Suite for Object Oriented Design*", IEEE Transactions on Software Engineering, 1994, pp. 476-492.

[20] S. Yacoub, T. Robinson and H.H. Ammar, "*Dynamic Metrics for Object-Oriented Design*", Proceedings of 6th International Conf. on Software Metrics Symposium, 1999, pp. 50-61.

[21] T. McCabe, "*A Software Complexity Measure*", IEEE Transactions on Software Engineering SE-2(4): 308-320, 1976

[22] T.J. McCabe, "*A Complexity Measure*," IEEE Trans. Software Eng., vol. 2, no. 4, pp. 308-320

[23] V.R. Basili, L. Biand and W.L. Melo, "*A validation of object-oriented design metrics as quality indicators*", Technical report, University of Maryland, USA., 1995

[24] http://pmd.sourceforge.net/

[25] http://phpmd.org

[26] http://faculty.fortlewis.edu/CHEW_B/Documents/Table of critical values for Pearson correlation.htm