www.ijecs.in

International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 3 March 2017, Page No. 20710-20715
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i3.56

Cognitive Weighted Polymorphism Factor: Empirical and Theoretical Validations

T. Francis Thamburaj[1], *A. Aloysius*[2]
[1]Computer Science Department, Bharathidasan University, St. Joseph's College,
Tiruchiraappalli, Tamil Nadu 620002, India.
francisthamburaj@gmail.com
[2] Computer Science Department, Bharathidasan University, St. Joseph's College,
Tiruchiraappalli, Tamil Nadu 620002, India.
aloysius1972@gmail.com

Abstract: ***Software complexity metrics are valuable and widely accepted tools to produce high quality software. The quality of the software metrics depends on various validations to prove it as valid, robust, realistic, accurate, and comprehensive metric. So, validations are important while proposing a new complexity metric. The Cognitive Weighted Polymorphism Factor complexity metric, already proposed by the author, is validated empirically as well as theoretically to prove its worth. Case studies are conducted to prove the applicability of the metric in all situations. To show the theoretical soundness of the metric, validations are done against Weyuker's nine properties and Abreu's seven criteria. The empirical validation is done to corroborate the theoretical validations. To show the better accuracy of the metric, the comparative study is done. Finally, the statistical validity is displayed with the performance of correlation analysis. All the validations proved that the Cognitive Weighted Polymorphism Factor complexity metric is truly a valid, more robust, more realistic, more accurate and more comprehensive in nature.***

Keywords: Cognitive Weighted Polymorphism Factor, Software Metric Validation, Polymorphic Metrics, Cognitive Complexity Metrics, Object-Oriented Software Complexity Metrics, Software Metrics.

1. Introduction

The validation of software metrics, although an arduous task in the field of software engineering, is important and critical to the success of the software measurement [1]. The objective of software complexity metric validation is to assess whether the complexity metric is possessing the basic necessary properties of valid metric. The problem is that there is a lack of agreed upon validation procedures to analyze the quality of these metrics [2].

There are many ways to do the validation of the complexity metric. It can be experimented with case studies in which the metric is applied to pieces of programs [3]. It can be checked against the nine abstract measurement properties of Weyuker for a valid complexity metric [4]. Also, the metric can be validated by the seven criteria of Abreu for robust object-oriented complexity metric [5]. These theoretical validations permit to formally compare similar complexity metrics. Another way is to collect the actual data and empirically analyze to find out the veracity of the metric [6]. To find out the accuracy of the proposed metric, the comparative study can be done with similar metrics [7]. The statistical validation is done to inspect the quality of the complexity metric.

The following section explains the mathematical formula for the cognitive weighted polymorphism factor complexity metric. Section 3 portrays the experimentation with the case studies. Section 4 validates the complexity metric against Weyuker's nine properties. Section 5 verifies the validity with Abreu's seven criteria for the object-oriented complexity metric. Section 6 deals with empirical and comparative study and Section 7 does the correlation analysis for statistical validation. The final section draws the conclusion and possible future works.

2. The Complexity Metric for Validation

The Polymorphism Factor (PF) complexity metric is defined by Abreu et al. as the ratio of the actual polymorphism present in the software system with the maximum possible polymorphism potential, if all the methods, except the base ones, are overridden in all classes [8]. The mathematical formula for PF is

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) * DC(C_i)]} \tag{1}$$

where,
$M_o(C_i)$ = number of overriding methods in class C_i,
$M_n(C_i)$ = number of new methods defined in class C_i,
$DC(C_i)$ = number of children for class C_i,
TC = Total number of Classes in the whole software system.

Francis Thamburaj and Aloysius [9], have already defined the mathematical equation for Cognitive Weighted Polymorphism Factor (CWPF). The formal mathematical definition of CWPF, is given in the Eq. 2.

$$CWPF = \frac{\sum_{i=1}^{TC} CWM_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i)*DC(C_i)]*MCW} \tag{2}$$

where,
$CWM_o(C_i) = N_{PP} * CW_{PP} + N_{SP} * CW_{SP} + N_{DP} * CW_{DP}$,
$M_n(C_i)$ = number of overriding methods in class C_i,
$DC(C_i)$ = number of children for class C_i,
TC = Total number of Classes.
N_{PP} = number of pure polymorphism,
N_{SP} = number of static polymorphism,
N_{DP} = number of dynamic polymorphism,
CW_{PP} = cognitive weight of pure polymorphism,

CW_{SP} = cognitive weight of static polymorphism,
CW_{DP} = cognitive weight of dynamic polymorphism.
MCW = Max (CW_{PP}, CW_{SP}, CW_{DP})

In the numerator of Eq.2, each type of polymorphism is multiplied with corresponding cognitive weight. The denominator is multiplied by MCW the maximum of the three cognitive weights. Thamburaj et al, already calibrated the cognitive weights for different types of polymorphisms such as pure, static, and dynamic with series of empirical experiments. The unique cognitive values are: CWpp = 3, CWsp = 5, and CWdp = 7 [9].

3. Experimentation and Case Studies

The proposed CWPF metric given by Eq. 2 is evaluated with the following case study programs. In the first case study, the program has two classes. The parent class C1 has two methods m1 and m2. The child class C2 has two statistically overridden methods. The UML diagram is given in Figure 1.

```
/*** First Case Study Program ***/
class C1 {
    float v1 = 3;
    void m1(int i){ }
    void m2(char ch){ }
}
class C2 extends C1 {
    void m1(float f) { }
    void m2(String s) { }
}
```

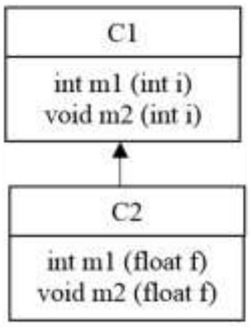

Figure 1: UML Diagram of First Case Study

Applying CWPF metric as given in Eq. 2,

$$CWPF = \frac{M_o(C1) + M_o(C2) * CW_{sp}}{(M_n(C1) * 1 + M_n(C2) * 0) * MCW}$$
= (0+2*5) / ((2*1+2*0)*5) = 1

```
/*** Second Case Study Program ***/
class C3 {
    float v1 = 3;
    void m1(int i){ }
    void m2(char ch){ }
    float m3(){
        return 4 * v1;}
    float m4(){
        return v1 * v1;}
}
class C4 extends C3 {
    void m1(float f) { }
    void m2(String s) { }
}
class C5 extends C3 {
    float m3(){
        return 2 * 3.14 * v1;}
    float m4(){
        return v1 * v1 * v1;}
    int m5(int a, int b) {
        return a+b;}
    int m6(int a, int b) {
        return a * b;}
}
class C6 extends C5 {
}
class C7 extends C5 {
}
```

In the second case study, the program has five classes. The root class C3 has two methods m3 and m4. The C4 class has two statically overridden methods. The C5 class has two dynamically overridden methods and two new methods. The C4 and C5 classes has no method. The UML diagram of the program is given in Figure 2. Applying CWPF metric as given in

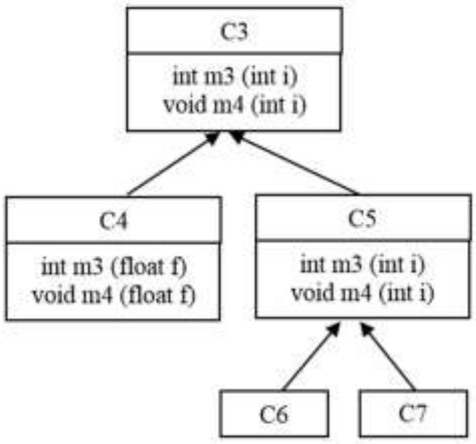

Figure 2: The UML of Second Case Study

Eq. 2,

$$CWPF = \frac{M_o(C3)+M_o(C4)*CW_{sp}+M_o(C5)*CW_{dp}+M_o(C6)+M_o(C7)}{\begin{pmatrix} M_n(C3)*4+M_n(C4)*0+M_n(C5)*2 \\ +M_n(C6)*0+M_n(C7)*0 \end{pmatrix}*MCW}$$
= (0+2*5+2*7+0+0) / (2*4+0*0+0*2+0+0)*7
= 24/56 = 0.48

In the third case study, the program has three classes. The root class C8 has two methods m1 and m2. The C9 class has two statically overridden methods. The C10 class has two statically overridden methods and one new method. The UML diagram is given in Figure 3. The Java program is given below.

```
/*** Third Case Study Program ***/
class C8 {
    float v1 = 3;
    void m1(int i){ }
    void m2(char ch){ }
}
class C9 extends C8 {
    void m1(float f) { }
    void m2(String s) { }
}
class C10 extends C8 {
    void m1(float f) { }
    void m2(String s) { }
    void m3(double d) {}
15:  }
16:
```

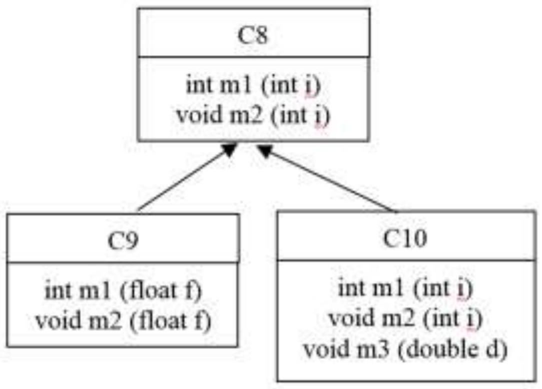

Figure 3: UML Diagram of Third Case Study

Applying CWPF metric as given in Eq. 2,

$$CWPF = \frac{M_o(C8) + M_o(C9) * CW_{sp} + M_o(C10) * CW_{sp}}{(M_n(C8) * 1 + M_n(C9) * 0 + M_n(C10) * 0) * MCW}$$

= (0+2*5+2*5) / ((2*2+0*0+1*0)*5) = 1

4. Validation by Weyuker's Properties

Weyuker has proposed nine properties to evaluate any software complexity measure [10]. A good software complexity metric should follow all or at least the majority of these properties in order to be a good complexity metric, since they evaluate the weaknesses of a measure in a concrete way. Chidamber and Kemerer refer Weyuker's properties to validate their metric suite [11]. This section validates CWPF against these criteria to establish the robustness of the complexity metrics.

Property 1: $(\exists P)(\exists Q) \mid (|P| \neq |Q|)$ *where P and Q are program bodies.*
The first property states that a complexity metric should not rank all program bodies as equally complex. The CWPF complexity metric has different complexity values in the case study, namely, 1 and 0.48. Hence it satisfies the first property

Property 2: Let c be a non-negative number. Then there are only finitely many programs of complexity c.
This property states that there should be only a finite number of classes with same complexity metric values. That is, a complexity measure should not be too "coarse" to rate too many programs as being of equal complexity. Any software system will have only finite number of classes and polymorphism can happen only within these classes. Further, the number of pure, static and dynamic types may not be the same in all polymorphic tree. So, they will have different CWPF complexity metric values as shown in case studies. Hence the property is satisfied.

Property 3: There are distinct program P and Q such that $|P| = |Q|$
This property states that a measure should not be too "fine" to assign to every program a unique complexity. In the case studies there are two different polymorphic trees, namely case study 1 and 3, have the same CWPF complexity value of 1. Hence, it is not too fine to assign different complexity values. So, this property is satisfied.

Property 4: $(\exists P)(\exists Q) \mid (P \equiv Q \;\&\; |P| \neq |Q|)$ *where P and Q are program bodies.*
This property is about the syntactic complexity of the classes. That is, the implementation of two identical classes may differ and hence the complexity metric value will differ. In the case studies, the identical classes C1 and C3 have different complexity values 1 and 0.48 due to the variation in the number of children. Hence this property is satisfied.

Property 5: $(\forall P)(\forall Q) \quad (|P| \leq |P;Q| \text{ and } |Q| \leq |P;Q|)$
This property is about 'monotonicity' of program complexity. More specifically, the components of a program are no more complex than the program itself. The candidate complexity metrics satisfy this property. In case study 2, the overall complexity of the polymorphic tree C3, C4, C5, C6, C7 is greater than the complexity of each sub tree like C5, C6, C7, as the number of classes and methods is less than the overall tree.

Property 6: $(\exists P)(\exists Q)(\exists R) \mid (|P| = |Q|)\&(|P;R| \neq |Q;R|)$
$(\exists P)(\exists Q)(\exists R) \mid (|P| = |Q|)\&(|R;P| \neq |R;Q|)$
This property states that if a new class is appended to two classes with the same class complexity, the class complexities of the two new combined classes are different. In other words, the interaction between P and R can be different than the interaction between Q and R resulting in different complexity values for P + R and Q + R. In the case of CWPF complexity metric, adding a new class to two different polymorphic tree will not change the complexity value since CWPF depends on the number of classes and new methods in each class in each tree. Hence, this property is not satisfied.

Property 7: $(\exists P)(\exists Q)$ *such that Q is formed by permuting the order of the statements of P and* $(|P| \neq |Q|)$
The property seven states that the changing of the order of the program changes the complexity metric value. The Polymorphism metric depends on the number of methods in the parent and child classes irrespective of the order of the method in each class. Hence the complexity metric value of CWPF will not change if we change the order of the methods. So, the property is not satisfied.

Property 8: If P is renaming of Q, then $(|P| = |Q|)$
The property eight says that renaming a program does not change the complexity metric value of the program. It is clearly evident that the complexity metric values of CWPF is not affected by the name change of the program, as it depends only on the number of children and new methods in each class of the tree. Hence, this property is satisfied.

Property 9: $(\exists P)(\exists Q) \mid (|P| + |Q|) < (P;Q)$
The last property tells that the complexity of a program formed by concatenating two programs may be greater than the sum of their complexities. The rationale behind this logic is that there may be interaction between the concatenated subprograms. The complexity metric CWPF depends on the number of classes and new methods in each class. When two programs with same complexity values are combined, the number of children will change resulting in change of complexity value. Therefore, this property is satisfied by the candidate complexity metric.

Table 1: Summary of Weyuker's Properties

Weyuker's	PF	CWPF
Property 1	Yes	Yes
Property 2	Yes	Yes
Property 3	Yes	Yes
Property 4	Yes	Yes
Property 5	Yes	Yes
Property 6	No	No
Property 7	No	No
Property 8	Yes	Yes
Property 9	Yes	Yes

The Table 1 shows the summary of the validation of CWPF against the Weyuker's nine properties for a good software metric. The property 7 does not fit, in general, to any object-oriented software system. Out of the nine properties, seven properties are satisfied by CWPF complexity measures. The high number of satisfied properties proves that CWPF is theoretically sound and valid complexity metric.

5. Validation with Abreu's Criteria

A set of seven criteria for evaluation of an object-oriented complexity metric is proposed by Abreu *et al* [8]. Here the proposed metric is validated against these criteria.

Criterion 1: Metrics determination should be formally defined.
The first criterion requests to avoid the subjectivity in the measurement of complexity of the software. The objective formalization of the complexity metric enables comparisons with other similar complexity metrics. The CWPF complexity metric is formally defined with a mathematical equation. Hence, it satisfies the first criterion.

Criterion 2: Non-size metrics should be system size independent.
The second criterion is about the usefulness of the complexity metric. The complexity metric should be applied over different types and sizes of projects. Regarding the CWPF complexity metric, the case study shows that it is applicable to single, multi-level, and hierarchical inheritance trees with varying structural complexities and sizes, in terms of number of classes. Hence this criteria is satisfied.

Criterion 3: Metrics should be dimensionless or expressed in some consistent unit system.
The third criterion states that the subjective or artificial measurement units for the complexity metrics should be avoided, in order to escape from the possibility of misinterpretations. The measurement units for CWPF complexity metric is in ratio scale and hence it is dimensionless as given in this criterion.

Criterion 4: Metrics should be obtainable early in the life-cycle.
This criterion talks about the reduction of cost and human effort in developing the software system. The polymorphism factor complexity metric values can be captured early in the software life cycle. The number of polymorphic functions can be surmised even at the basic design phase, even though more accurate number can be got at the coding level. So, CWPF complexity metric satisfies this criterion.

Criterion 5: Metrics should be down-scalable.
This criterion talks about the applicability of complexity metrics both in the system level and subsystem or module level, as the software development process is generally done by breaking the large software system into many manageable modules and finally integrated. The down-scalable criterion is applicable to CWPF complexity metric, as it is a class level complexity metric and its value is calculated for each class separately before adding them to give the system level complexity metric value.

Criterion 6: Metrics should be easily computable.
The sixth criterion deals with the practicality of complexity metric collection, that is tedious, time consuming and costly. This problem can be solved if the complexity metric formulation is simple and hence easy to calculate the values. It will be better if it yields itself to automatize the complexity metric value collection process. Calculation involved in the CWPF equation can easily be computerized since it involves only simple multiplication, addition and division in order to find the ratio value. Hence, this criterion is satisfied by CWPF complexity metric.

Criterion 7: Metrics should be language independent.
The last criterion speaks about the independence of the complexity metric from different programming language constructs. In other words, the complexity metric should be applicable to different languages with their particular bindings. Though the CWPF complexity metric is designed with Java language, it can be applied to other object-oriented languages with language specific bindings. Abreu has done it for PF complexity metrics with C++ language [12] and Eiffel language [13] with different bindings specific to the language.

Table 2: Summary of Abreu's Criteria

Abreu's	PF	CWPF
Criterion 1	Yes	Yes
Criterion 2	Yes	Yes
Criterion 3	Yes	Yes
Criterion 4	Yes	Yes
Criterion 5	Yes	Yes
Criterion 6	Yes	Yes
Criterion 7	Yes	Yes

The Table 2 gives the summary of the validations with Abreu's criteria. The table shows that all the seven criteria of Abreu are satisfied by the CWPF complexity metric. Hence, the complexity metric is theoretically sound and valid metric.

6. Comparative Study

In order to prove that the proposed complexity metric CWPF is better than the existing complexity metric PF, the comparative study is done [8]. When Abreu proposed the Polymorphism Factor complexity metric, he did not consider the cognitive complexity due to the polymorphism. He had considered only the structural or the architectural complexity. Therefore, PF complexity metric does not depict the true picture of the complexity that is existing in the program. This is precisely the difference between the PF and CWPF complexity metrics. The CWPF complexity metric is more sophisticated and accurate than PF complexity metric of Abreu, as it includes the cognitive complexity that arises due to three different types of polymorphisms, namely pure, static, and dynamic [14]. The cognitive weights of each type is calibrated through a series of experiments [9]. These cognitive weights are multiplied with the corresponding type of polymorphism to yield the accurate measurement of the complexity metric. The polymorphic cognitive weights are the effort needed by the programmer or the user to understand the different types of polymorphism embedded in the program.

Table 3: The Complexity Metric Values and CMT

Programs#	PF	CWPF	CMT
P1	8.3	11.7	315.2143
P2	7.69	9.2	274.5714
P3	12	15.2	362.8571
P4	10.3	11.7	3521429
P5	9.52	9.52	259.9286

The comprehension tests were conducted in order to compare the proposed CWPF complexity metric with the already existing PF complexity metric. A group of forty students doing their master's degree in Computer Science was employed for this purpose. The students were supplied with five different programs,

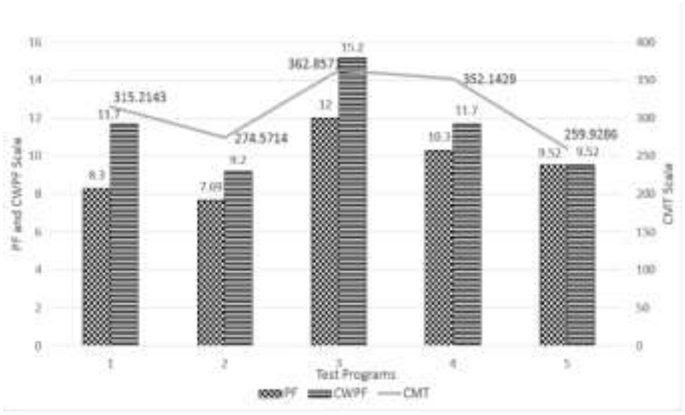

Figure 4: Comparison of PF and CWPF with CMT

P1 to P5 in Java for the comprehension test. The time taken to comprehend the program and complete the test in seconds was captured. To maintain the accuracy, the time spent was recorded online with the help of test program interface tool. The average time taken to comprehend each of the five programs by all the students was calculated and placed in the Table 3 under the column head Comprehension Mean Time (CMT). The complexity values of PF and CWPF complexity metrics were calculated manually for each of the five programs as shown in the experimentation and case studies in the section 3. To capture the differences between the two complexity metrics, the values found in Table 3 are translated into the graphical representation as shown in Figure 4. In this graph, the values of CWPF is much closure to CMT than the values of PF. This shows that CWPF complexity metric is more accurate complexity indicator than PF complexity metric.

7. Statistical Analysis

The statistical analysis over the PF and CWPF with respect to CMT was done using the Pearson correlation test. In the Pearson correlation test, the coefficient 'r' is calculated to check the type and the strength of the association between PF & CMT and CWPF & CMT. Here, the correlation 'r' can take value between +1 to -1. The positive value shows that there is direct correlation between the variables in which as the X value increases the Y value increases. The negative value indicates that there is inverse correlation in which as the X value increases, the Y value decreases.

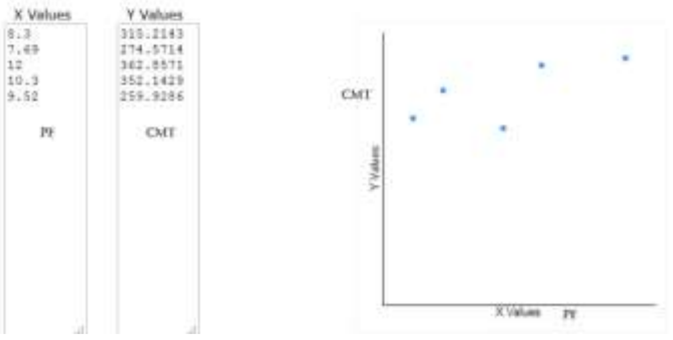

Figure 5: Pearson Correlation between PF and CMT

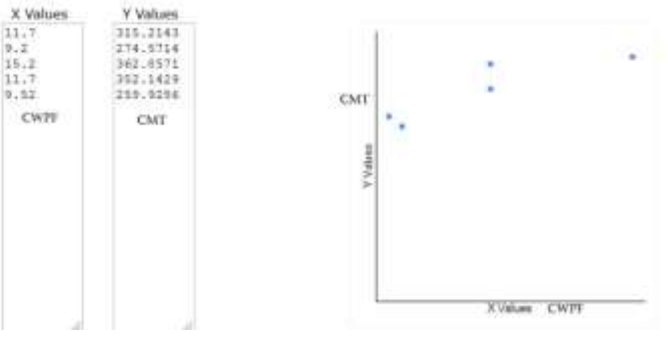

Figure 6: Pearson Correlation between CWPF and CMT

The calculated 'r' value for the PF and CMT is 0.7146 and for CWPF and CMT is 0.8836. Both the correlation values are positive and so the complexity metrics PF as well as the CWPF are directly correlated with CMT. In other words, as the values of PF and CWPF increase, the CMT values also increase and vice versa. Thus there is a correspondence between the PF & CMT and CWPF & CMT. The value of 'r' points to the strength of the correlation between the variables. The 'r' value between CWPF and CMT is bigger than the 'r' value between PF and CMT. It implies that the correspondence between CWPF and CMT is stronger than the correspondence between PF and CMT. The Pearson correlation graphs in Figure 5 and Figure 6 also reveal this fact clearly. Hence, CWPF is statically proved to be the better complexity measure than the PF measure.

8. Conclusion

The validation of CWPF complexity metric is done using five different methods. The applicability of the metric is proved with case studies. The theoretical validation against Weyuker's nine properties and Abreu's seven criteria emphatically asserted that CWPF complexity metric is a sound and robust metric. The comparative study done with PF complexity metric has exhibited that CWPF complexity metric is more accurate than PF complexity metric. Finally, it is checked with the statistical analysis using the Pearson correlation which has proved CWPF complexity metric is a better and more comprehensive indicator than the PF complexity metric. All the five validations proved the practical applicability, theoretical veracity, empirical repeatability and statistical accuracy of the complexity metric CWPF beyond doubt.

As the future work, large scale empirical validation can be done over open source software systems. Also, validations can be done against specific aspect of the software quality like maintainability that plays a high priority role in software industry.

References

[1] K. P. Srinivasan, and T. Devi, "Software metrics validation methodologies in software engineering," International Journal of Software Engineering & Applications vol. 5, no. 6, pp. 87-102, 2014.

[2] Jacquet, Jean-Philippe, and Alain Abran, "From software metrics to software measurement methods: a process model," In Software Engineering Standards Symposium and Forum, 1997, Emerging International Standards. ISESS 97, Third IEEE International, pp. 128-135, 1997.

[3] Aloysius A., "A Cognitive Complexity Metrics Suite for Object Oriented Design," PhD Thesis, Bharathidasan University, Tiruchirappalli, India, 2012.

[4] T. Francis Thamburaj and A. Aloysius, "Validation of Cognitive Weighted Method Hiding Factor Complexity Metric," International Journal of Applied Engineering Research (IJAER), Print ISSN: 0973, Online ISSN 1087-1090, vol. 10, no. 82, .pp. 91-96, December, 2015.

[5] T. Francis Thamburaj and A. Aloysius, “On Validating Cognitive Weighted Attribute Hidiing Factor Complexity Metric,” International Conference on Computing Communication and Information Science (ICCCIS’16), July 29th 2016.

[6] T. Francis Thamburaj and A. Aloysius, “Cognitive Weighted Method Hiding Factor Complexity Metric,” International Journal of Computer Science and Software Engineering (IJCSSE), ISSN (Online): 2409-4285, vol. 4, no. 10, pp. 272-279, October 2015.

[7] T. Francis Thamburaj and A. Aloysisus, “Cognitive Perspective of Attribute Hiding Factor Complexity Metric,” International Journal of Engineering and Computer Science, ISSN: 2319-7242, vol. 4, no. 11, pp. 14973-14979, November 5th 2015.

[8] Abreu, Fernando Brito, and Rogério Carapuça., "Object-oriented software engineering: Measuring and controlling the development process,” Proceedings of the 4th international conference on software quality. vol. 186, 1994.

[9] T. Francis Thamburaj and Aloysius, A., "Cognitive Weighted Polymorphism Factor: A Comprehension Augmented Complexity Metric". World Academy of Science, Engineering and Technology, International Science Index 107, International Journal of Computer, Electrical, Automation, Control and Information Engineering, PISSN:2010-376X, EISSN:2010-3778, vol. 9, no. 11, pp. 2199 – 2204, November 12th 2015.

[10] Elaine J. Weyuker, “Evaluating Software Complexity Measures”, IEEE Transactions on Software Engineering, 9, 1357-1365, Sep. 1988.

[11] Chidamber S.R., Kemerer C.F., “Towards a Metrics Suite for Object-Oriented Design,” Object-Oriented Programming Systems, Languages and Applications (OOPSLA), 26, 197–211, 1991.

[12] F. Brito e Abreu, M. Goulao, and R. Estevers, "Toward the Design Quality Evaluation of OO Software Systems," Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, pp. 44-57. 1995.

[13] Abreu, Fernando Brito, Rita Esteves, and Miguel Goulão, “The design of Eiffel programs: Quantitative evaluation using the mood metrics,” In Proceedings of TOOLS’96. California, July, 1996.

[14] Benlarbi, Saïda, and Walcelio L. Melo, "Polymorphism measures for early risk prediction,” IEEE Software Engineering, 1999. Proceedings of the 1999 International Conference, pp.334-344, 1999.

Author Profile

T. Francis Thamburaj is working as Assistant Professor in Department of Computer Science, St. Joseph’s College, Trichy, Tamil Nadu, India. He has obtained the Master of Computer Applications degree in 1987 and Master of Philosophy degree in 2001 from Bharathidasan University, Trichy. He has 27 years of experience in teaching Computer Science. His research areas are Artificial Neural Networks and Software Metrics. He has published many research articles in the national & international journals and conferences. He has acted as a chair person for many national and international conferences. Notably, in 2011 he has chaired sessions and presented a research paper in the World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP’11), Las Vegas, USA. In 2015, he has chaired sessions in the international conference, World Academy of Science, Engineering and Technology (WASET), Kyoto, Japan and also presented a research paper on cognitive weighted polymorphism factor. A list of his research articles can be found in Google Scholar website. He is currently pursuing Doctor of Philosophy program and his current area of research is Object Oriented Cognitive Aspects in Software Metrics.

A. Aloysius is working as Assistant Professor in Department of Computer Science, St. Joseph’s College, Trichy, Tamil Nadu, India. He has got the Master of Computer Science degree in 1996, Master of Philosophy degree in 2004, and Doctor of Philosophy in Computer Science degree in 2013 from Bharathidasan University, Trichy. He has 15 years of experience in teaching and research. He has published many research articles in the National/ International conferences and journals. He has also presented 2 research articles in the International Conferences on Computational Intelligence and Cognitive Informatics in Indonesia. He has acted as a chair person for many national and international conferences. Currently, he is guiding many research scholars in Computer Science. His current areas of research are Cognitive Aspects in Software Design and Big Data,