International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-4 Issue-13 December-2013

Assessment of Predictive Object Points (POP) Values for Java Projects

Shubha Jain[1], Vijay Yadav[2], Raghuraj Singh[3]

Abstract

There are number of software development estimation techniques exist for sizing software systems and to support cost measurement like SLOC, Function Point etc. but none is directly applicable to object-oriented software. They all work for specific development environment. PRICE systems has developed the predictive object point (POP) metric for predicting effort required for developing an object oriented software system. This is based on the counting scheme of function point (FP) method. POP count results from the measurement of the object-oriented properties of the system. This paper explains how to calculate the POP values for Java Projects. An automation tool for measuring Predictive Object Points with more accuracy has been built. The tool and results of its application for Java Projects are presented and discussed.

Keywords

Software Measurement, Object Orientation, Functional size measurement, Software Metrics, Predictive Object Point, Automation.

1. Introduction to POP Software Sizing Metric

POP was introduced by Mickiewicz in 1998. PRICE systems [2] has developed the predictive object point (POP) metric for predicting effort required for developing an object oriented software system. It was designed specifically for Object oriented software and fulfilled almost all the criteria of OO concepts. POPs are intended as an improvement over FPs, which were originally intended for use within procedural systems [3]. POPs are a metric suitable for estimating the size, subsequently effort of object oriented software, based on the behaviours that each class is delivering to the system along with top level inputs describing the structure of a system [2].

Shubha Jain, Department of Computer Science and Engineering, Kanpur Institute of Technology, Kanpur, India.
Vijay Yadav, Department of Computer Science and Engineering, Kanpur Institute of Technology, Kanpur, India.
Raghuraj Singh, Computer Science and Engineering, Harcourt Butler Technological Institute, Kanpur, India.

What Contribute to POP Software Sizing Metric?
By the implementation of Object Oriented Paradigm the researchers modified and validated the conventional metrics theoretically or empirically [9]. The following metrics measure object-oriented systems in POP Count: Number of top level classes (TLC), Average number of weighted methods per class (WMC), Average depth of inheritance tree (DIT), and Average number of children per base class (NOC). WMC, DIT, and NOC are taken from the MOOSE metrics suite [7].

How to Calculate POP Count?
The above mentioned metrics are then gathered to form the equation (1), giving the number of POPs for a system [2].

$$f1(TLC, NOC, DIT) = TLC * (1 + ((1+NOC)*DIT)^{1.01} + (|NOC-DIT|)^{.01})$$
$$f2(NOC, DIT) = 1.0$$
$$POPs(WMC, NOC, DIT, TLC) = \frac{WMC * f1(TLC, NOC, DIT)}{7.8} * f2(NOC, DIT) \quad (1)$$

Where, f1 attempts to size the overall system, and f2 applies the effects of reuse through inheritance.

2. POP Metric to Assess Java Projects

Sizing and complexity metrics were the most impressive contributions for effort and cost estimation in project planning [8].

The POP metric relies heavily on the availability of the object design. An incorporation of New Top Level Class (classes which have no parent within the system) may lead to change in design. TLC contributes significantly to the POP count as if such top-level classes are less in number, the WMC value will also decrease and hence reducing the overall POP count value.

The methods were split into various types according to proportions taken from Minkiewicz [2] through the manual investigation of source code. However an organization would be benefitted by using a split based on its own past data. In the same way the

method types can be categorized into complexity types. It was felt that this categorization gives better results when applied to a language with high number of pure abstract classes, which is true for Java. Here interfaces might be thought of as pure abstract classes.
The reason for this is that each additional interface contributes significantly to the top-level class count. With increase in TLC, POP count will also increase. This is not true for projects based on C++, which tend to use pure abstract classes less extensively.

In true OO environment, a system should be divided into several subsystems and each subsystem could be divided into several stages according to time. This could be considered as the refinement of use of POP [2]. Thus systems should be split into modules and further divided into sub modules. Thus POP count of each java project can be accurately calculated on the basis of its individual java file which gives better results for the overall estimation of POP.

POP Count Measurement Process for Java Projects
An easy to use automation tool APA (Automated POP Analyzer) is built for counting POPs by splitting the whole Java Project into files and calculating POP on the basis of its individual java file. In the True OO environment as in java projects, the level of reusability through Inheritance is always considered to be high and hence function of NOC and DIT can be considered as 1.0. Thus the correction factor f_2 taken by Mickiewicz [2] can be omitted while estimating Java projects. However this may not be true for other environments.

Thus the factor $|NOC\text{-}DIT|^{.01}$ may be omitted and f2 may be neglected while calculating POP Count values for Java Projects. The POP Count formula may be reduced to the equation (2).

$$f1(TLC, NOC, DIT) = TLC * (1 + ((1 + NOC) * DIT)^{1.01})$$
$$POPs(WMC, NOC, DIT, TLC) = \frac{WMC * f1(TLC, NOC, DIT)}{7.8} \quad (2)$$

Where, f1 attempts to size the overall system.
The following process was followed for calculation of POP Count:
Step 1
The first step was to obtain the Source Lines of Code (SLOC) metric for projects through APA tool [4] based on CCCC, an object oriented metric gathering tool [5].
Step 2
Using the generated DIT metrics for each class it was possible to calculate the average DIT (one of the metrics required for POPs). Similarly the generated NOC metrics for each class were averaged to obtain the average NOC.
Average NOC = (Sum of Base Class NOCs) / (Number of Base Classes giving +ve NOC count.)
Average DIT = (Sum of Classes having DITs) / (Sum of the rows of NOC and DIT giving +ve count).
Step 3
Average Method count (AMC) is calculated by dividing the method count by the class count [4].
Step 4
The TLC metric for each java file and for overall project was then calculated. This includes the base classes (with no parents) and the class which is at level 0. This metric is a count of the classes that are roots in the class diagram, from which all other classes are derived [2].
Step 5
Finally WMC is calculated as suggested by Minkiewicz [2]. As in order to determine the average number of methods in each type, weightings should be applied against this as per the following calculations [1]:
Average Constructor/Destructor Method Count = 20% (Average Methods per Class)
Average Selector Method Count = 30% (Average Methods per Class)
Average Modifier Method Count = 45% (Average Methods per Class).
Average Iterator Method Count = 5% (Average Methods per Class).
Now, each method type was divided into three categories of complexity using weightings.
Low Complexity Method Count = 22% of Average Method Count
Average Complexity Method Count = 45% of Average Method Count
High Complexity Method Count = 33% of Average Method Count For each java file all twelve calculations were performed and their sum gives the value of WMC [4]. The same method is used for the calculation of WMC for the overall project.

3. Description of Empirical Study

The proposed refinement in POP formula for Java Projects can be checked in reference with the projects taken by T. R Judge and A. Williams [6]. They proved POP metric as better indicator of software size in

comparison to FP metric using comparative study of two projects through Table 1.

Table 1: Summary of Project Metrics [6]

Project Attributes	Project Alpha	Project Beta
Source Lines of Code (SLOC)	38854	20570
Total Number of Classes	404	147
Total Number of Methods	2412	833
Average of the Methods per Class	5.971	5.667
Average Depth of Inheritance	0.941	0.701
Average Number of Children	3.700	2.688
Top Level Classes	201	73
Constructors/Destructors (20%)	1.194	1.133
Selectors (30%)	1.791	1.700
Modifiers (45%)	2.687	2.550
Iterators (5%)	0.299	0.283
WMC	62.564	59.379
Number of POPs	10478	2566

They summarize the effort metrics that were originally estimated by the development team and the effort that was actually expanded. These statistics are presented as ratios [6]:

(Actual Effort Project Alpha) : (Actual Effort Project Beta) = 4.58
This gives information that project alpha actually took about 4.58 times as many days to develop as the project beta. This information is useful for comparing with a similar ratio of POPs between the two projects.

Comparing the POP count for the two projects reveals project alpha is approximately four times as large, in terms of the POP count, compared to project beta, using the following percentage calculation:

$$\frac{POP\alpha}{POP\beta} = \frac{10478}{2566} \approx 4.08 \qquad (3)$$

Table 2 shows the value of the factor $|\mathbf{NOC\text{-}DIT}|^{.01}$ and proposed refined POP Count values for the Projects Alpha and Beta.

Table 2: Refined POP Count for Projects

Project Attributes	Project Alpha	Project Beta
$\|NOC\text{-}DIT\|^{.01}$	1.0102	1.006
Number of refined POPs	8849.422	2006.1515

$$\frac{POP\alpha}{POP\beta} \text{ Re } fined \approx 4.411 \qquad (4)$$

On Comparing the POP count for any two projects say P_1/P_2 reveals how much times the project P_1 is of project P_2 in terms of size. The closer this ratio to that of the efforts ratio, the more accurate the POP technique is. This is because Effort and POP count are proportional, where the constant of proportionality is the POP productivity rate [6]. The result from equation (4) is more close to the Actual effort ratio (4.58) in comparison to the original POP Ratio from equation (3).

Fig 1.1 shows the sample POP original and refined values obtained through APA Tool.

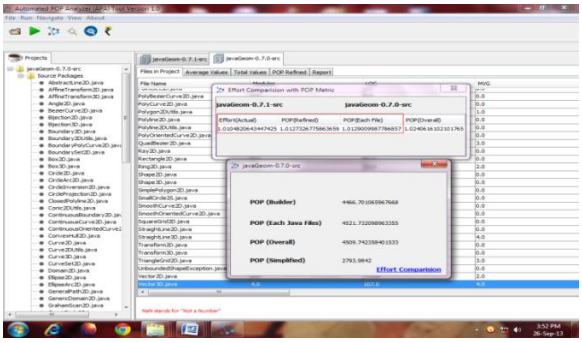

Fig.1.1 POP original and refined values through APA Tool

From the results it may be seen that on neglecting the factor $|NOC\text{-}DIT|^{.01}$ in original formula of POP calculation and omitting the function f_2 give better results near to effort. Thus this validates the proposed refinement in POP Count formulation for Java Projects

4. Conclusion and Future Work

In conclusion, the POP metric when applied to past data on two of Parallax's deployed projects gave a better indication of their size than did the previous POP count values. POP is a metric, which takes into account several well-defined metrics within an object-oriented system. However, it is difficult to develop the OO design for the entire system in an industrial setting. A careful design and analysis is required for complex subsystems. This suggests that the POP metric might be best applied to the estimation of the Java Projects which can be further categorized into critical subsystems and then modules

which can be further evaluated for each java files in the entire project. Here the refinement in POP Count formula has been proposed for Java Projects. As in the True OO environment for e.g. in java projects, the level of reusability through Inheritance is always high. Thus the factor $|NOC\text{-}DIT|^{.01}$ may be omitted and f2 may be neglected while estimating Java projects. The suggested refinement in POP calculation showed more accurate results in terms of effort estimation. This makes system more understandable hence validate the proposed refined formula for POP metrics calculations for Java Projects. The projects taken for empirical study from research work of T. R Judge and A. Williams [6], presented more accurate results however the data to be studied may include additional java projects. This will further ensure the validity for this refinement for Java Projects and hence accuracy of the measurement.

References

[1] C. Ravindranath Pandian, "Software Metrics: A Guide to Planning, Analysis and application". Auerbach Publications A CRC Press Company Boca Raton London New York Washington, D.C., 2005.

[2] Arlene F. Minkiewicz, "Object- Oriented Metrics" Software Development. Wiley Computer Publishing, pp. 43-50., 1997 At: http://www.sdmagazine.com.

[3] Haugh. M, E. W Olsen and Bergman. L., "Software Process Improvement: Metrics Measurement and Process Modelling", Vol 4, New York, Springer, pp. 159-170., 2001.

[4] Shubha Jain, Vijay Yadav and Prof. Raghuraj Singh, "OO Estimation Through Automation of Predictive Objective Points Sizing Metric"., International Journal Of Computer Engineering and Technology (IJCET) Volume 4, Issue 3, pp. 410-418, May-June 2013.

[5] CCCC Metric Tool by Tim Littlefair. http://www.fste.ac.cowan.edu.au/~tlittlef/.

[6] T. R Judge, A. Williams, "OO Estimation – an Investigation of the Predictive Object Points (POP) Sizing Metric in an Industrial Setting". Parallax Solutions Ltd, Coventry, UK, 2001.

[7] Shyam R. Chidamber and Chris F. Kemerer,"A Metrics Suite for Object Oriented Design". IEEE transactions On Software Engineering, Vol. 20, No. 6, pp. 476-493, June 1994.

[8] Dr. Rakesh Kumar and Gurvinder Kaur, "Article: Comparing Complexity in Accordance with Object Oriented Metrics". International Journal of Computer Applications, 15(8), pp. 42–45: February 2011.

[9] M. Xenos and D. Stavrinoudis and K. Zikouli and D. Christodoulakis, "Object- oriented metrics – a survey", proceedings of the FESMA 2000, Federation of European Software Measurement Associations, Madrid, Spain, 2000.

Shubha Jain is BE (Electronics and Communication) and M.Tech. (CS). She is pursuing PhD in Computer Science from Uttrakhand Technical University, Dehradun in area of Software Engineering. She has about 20 years of experience in teaching. Currently she is working as Associate Professor and Head in CSE/IT Dept at Kanpur Institute of Technology, Kanpur. She is the Secretary, Kanpur Chapter, CSI and member of IETE society. She has guided 5 M.Tech. Projects and several B.Tech projects. She has 10 papers in National/International Conferences/Journals.

Vijay Yadav is B.Tech in (I.T) from C.S.J.M University Kanpur (U.I.E.T) and M.Tech in (C.S.E) from U.P Technical University. Currently he is working as Assistant Professor in Department of Computer Science & Engg. at KIT, Kanpur. He has undergone projects like Online Entertainment world, Enterprise Resource Planning System during his B.Tech (I.T) curriculum. He has done his M.Tech (C.S.E) thesis in Object Oriented Software Metrics (POP Automation). He is meritorious student of B.Tech (I.T) and M.Tech (C.S.E) and have got merit excellence award. He has 2 papers in International Conferences/Journals.

Prof. Raghuraj Singh is B.Tech. (CSE), M.S. (Software Systems) and Ph.D. in Computer Science and Engineering from U.P. Technical University. He has about 23 years of experience in teaching. Currently he is working as Professor and Head in the Department of Computer Science & Engg. at HBTI, Kanpur. He has guided 7 PhDs and 17 M.Techs and several B.E./B.Tech projects. He is the Chairman, Kanpur Chapter, CSI, Life Member of ISTE, Member of the Institution of Engineers (India), Fellow Member of IETE, Professional member of ACM and Senior Member of International Association of IACSIT. He has more than 80 papers in National / International Conferences and Journals to his credit. Currently 4 students are working for PhD and 4 are pursuing M.Tech. under his guidance.