INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY (IJCET)

ISSN 0976 – 6367(Print)
ISSN 0976 – 6375(Online)
Volume 4, Issue 3, May-June (2013), pp. 410-418
© IAEME: www.iaeme.com/ijcet.asp
Journal Impact Factor (2013): 6.1302 (Calculated by GISI)
www.jifactor.com

IJCET

©IAEME

OO ESTIMATION THROUGH AUTOMATION OF THE PREDICTIVE OBJECT POINTS SIZING METRIC

Shubha Jain[1], Vijay Yadav[1] and Prof. Raghuraj Singh[2]

1 Department of Computer Science & Engineering, Kanpur Institute of Technology, Kanpur (India)

2 Department of Computer Science & Engineering, Harcourt Butler Technological Institute, Kanpur -208001(India)

ABSTRACT

To efficiently manage the resources various software characteristics like size, cost, quality etc is being estimated through different techniques and during different phases of software development. One of the techniques is function points measurement but inadequate for object oriented software for prediction of efforts. PRICE Systems has developed a metric called Predictive Object Points which was designed specifically for Object oriented software and result from measurement of the object-oriented properties of the system. It fulfilled almost all the criteria of OO concepts but it was not validated and has not adopted by industry thus has not gained sufficient recognition from practitioners to be used on a regular basis. Predictive Object points have been developed in the context of companies (e.g. Price Systems). But no details about their actual usage in these companies are publicly available. In this paper we discuss the theory behind POP, the problems with it and present an automation tool for measuring Predictive Object Points with more accuracy. The tool and results of its application in an industrial environment are presented and discussed.

KEYWORDS: Object Orientation, Software size Measurement, Software Metrics, Predictive Object Point, Automation.

I INTRODUCTION

With the rapid growth in software industries, corporate developers faced an interesting conflict between two emerging trends: Object Oriented development and metrics. They found that object oriented technology is, in many ways, incompatible with traditional metrics. Good measurement program is the choice of good metrics. The metrics are guidelines and not rules. They give an indication of the progress that a project has made and the quality of the design [13]. Industries are still struggling with the question of what to measure in their object oriented implementations.

The Source Lines of Code (SLOC) metric and the Function Point metric were both conceived in an era when programming required dividing the solution space into data and procedures. This notion conflicts with the object-oriented paradigm. Traditional design techniques separate data and procedures while object-oriented designs combine them. It is important to measure the amount of raw functionality the software delivers, but it is equally important to include information about communication between objects and reuse through inheritance in the 'size' as well [1].

Researchers studied ways to maintain software quality and developed object-oriented programming in part to address common problems by strongly emphasizing discrete, reusable units of programming logic. By the implementation of OOP the researchers modified and validated the conventional metrics theoretically or empirically. Sizing and complexity metrics were the most impressive contributions for effort and cost estimation in project planning [2][12].

II POP – AN OBJECT-ORIENTED SOFTWARE SIZING METRICS

POP was introduced by Minkiewicz in 1998 PRICE systems [1] has developed the predictive object point (POP) metric for predicting effort required for developing an object oriented software system. This was based on the counting scheme of function point (FP) method. POPs are intended as an improvement over FPs, which were originally intended for use within procedural systems, by drawing on well-known metrics associated with an object-oriented system [3].

Predictive Object Points (POPs) [1] incorporate three dimensions of OO systems: the amount of functionality the software delivers, communication between objects and reuse through inheritance.

These aspects can then be used to give rise to a single metric in order to indicate the amount of effort involved in the production of a software system. POPs are based on objects and their characteristics.

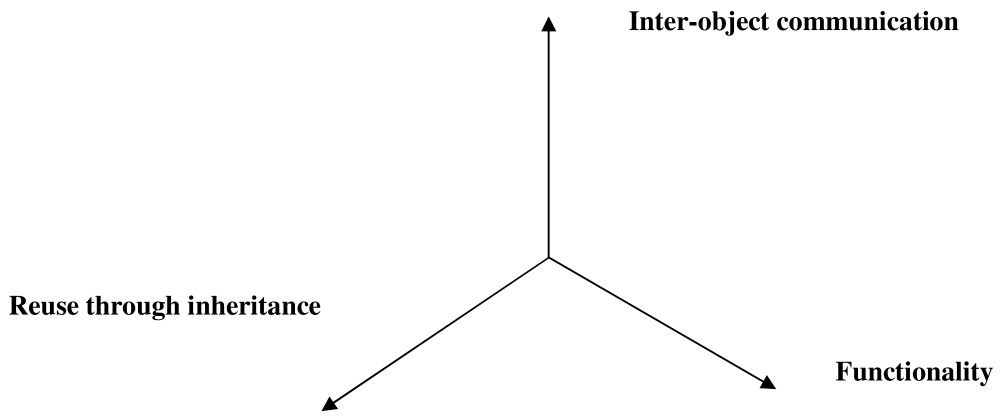

Figure 1 - Aspects of an Object-Oriented System

Abstract Model: POPs combine the following metrics presented in the literature to measure object-oriented systems: number of top level classes (TLC), average number of weighted methods per class
(WMC), average depth of inheritance tree (DIT), average number of children per base class (NOC). WMC, DIT, and NOC are taken from the MOOSE metrics suite of Chidamber and Kemerer [10].

Measurement process: The following formula was proposed to calculate the size of the overall system[1]:

$$f1(TLC, NOC, DIT) = TLC * (1 + ((1 + NOC) * DIT)^{1.01} + (|NOC - DIT|)^{.01})$$

$$f2(NOC, DIT) = 1.0$$

$$POPs(WMC, NOC, DIT, TLC) = \frac{WMC * f1(TLC, NOC, DIT)}{7.8} * f2(NOC, DIT)$$

Equation 1 - POP formula

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

First, the WMC is calculated for each kind of method suggested by Booch [11]: constructor/destructor, selector, modifier, and iterators. These methods are classified as having a low, average or high complexity. Then, a weight is assigned depending on the method's complexity. To do this, the author proposed a complexity table for each method type based on data collected from 20 developed software
systems. The complexity assignments are made considering the number of properties of the method and the number of message responses. The total weighted methods per class are calculated by summing the values obtained for each kind of method.

III POP METRICS AUTOMATION

Unfortunately use of POP has been discontinued. This research was conducted nearly 20 years ago and has not been adopted by industry. Though it was an interesting theoretical development, its use in real life was impractical as it required entirely too much detail for top down estimation. POPs focus on implementation aspects and lack generalization. Also, require detailed knowledge about system processing that is not available early enough for accurate counting. A tool called Predictive Object Point Builder was built as an effort to automate the measurement process but it has not been validated and hence not gained sufficient recognition from practitioners to be used on a regular basis. Thus an easy to use automation tool is required for counting POPs. It should be automated in a way so that it is suitable for existing traditional metrics also and hence useful for industries.

Here an *Automation Tool APA (Automated POP Analyzer)* for measuring Predictive Object Points is prepared. The tool and results of its application in an industrial environment are presented and discussed in this paper.
In building the tool, the following process was followed:

Step 1

The first step was to obtain the Source Lines of Code (SLOC) metric for projects. This was achieved through the use of CCCC, an object oriented metric gathering tool. This is a free metric tool for C++ and Java (CCCC) developed by Tim Littlefair [4].

Step 2

CCCC was once more used to obtain metrics necessary for the calculation of the POP metric. 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 DIT = (Sum of Class DITs) / (Number of Classes with gives DIT count)

Average NOC = (Sum of Base Class NOCs) / (Number of Base Classes which gives NOC count)

Step 3

It was then necessary to determine for each project the number of methods of each method type for use in the WMC calculation. In order to achieve this, the average method count per class was found by dividing the method count by the class count. Sum of the WMC1 value generated by CCCC is considered as the total number of methods and the sum of the classes which gives WMC1 count is considered as the total number of classes. This was verified from the result of Analyst4j Tool [9] which gives Average Weighted Method of class for a Project equivalent to the value of AMC calculated below:

Average Methods per Class = (Number of Methods) / (Number of Classes)

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)

This spread of method types arose from a manual investigation of source code by Minkiewicz[1].

Step 4

The TLC metric for each java file and for overall project was then calculated. The output of CCCC is a set of where each row represents the metrics for a particular class. Only those rows showing positive NOC counts were filtered i.e. the base classes. Sum of those base classes is considered as the TLC.

Step 5

Finally WMC is calculated as follows:

First each method type was divided into three categories of complexity using weightings, which arose from a manual investigation of source code, by Minkiewicz[1].

Low Complexity Method Count = 22% Average Method Count.

Average Complexity Method Count = 45% Average Method Count.

High Complexity Method Count = 33% Average Method Count.

This observation indicates that for a given method type, 45% of the methods are of average complexity, for example. At this stage, the average methods per class had been split into four categories according to method type and these four categories had been split into three further categories according to complexity.

It then remained to apply weightings to each of the twelve categories, again using values obtained from an examination of past project data by Minkiewicz[1].

For each java file all twelve calculations were performed and their sum gives the value of WMC. The same is calculated for the overall project also.

Finally POP count is computed according to the formula given in equation 1.

IV DESCRIPTION OF EMPIRICAL STUDY

Two projects developed in java were selected from http://1000projects.org/ [8] have been considered as inputs for the study. The first project is "Civilisation_game_java and the other is Payroll system. For both the projects POP count was calculated through APA tool considering the count of individual class as well as considering the project as a whole. The results are compared with the POP count from the POP builder [1].

Table :1 Project: Civilisation_Game_Java (Analysis of Each Java File of Project)

						Through APA Tool		Through POP Builder		
Java File Name ↓	**NOM**	**LOC**	**Methods**	**Classes**	**AMC= Methods /classes**	**WMC**	**POP**	**WMC**	**POP**	**Actual Effort**
1. AboutPanel	7	28	2	1	2	20.96	16.19	0	0.0	0.056
2. City	2	25	4	1	4	41.91	0.0	36	0.0	0.0499
3.CustomCellRend'r	7	222	2	1	2	20.96	16.19	0	0.0	0.494
4.CustomDataModel	3	18	4	1	4	41.91	16.17	36	13.9	0.035
5. GameCore	6	64	6	1	6	62.87	0.0	63	0.0	0.134
6. GWMap	2	1462	15	1	15	157.2	0.0	155	0.0	3.58
7. LaughButton	4	46	4	1	4	41.36	16.17	36	13.9	0.095
8. MainGUI	17	178	5	1	5	52.39	60.73	51	59.1	0.392
9. MapCreation	6	70	1	1	1	10.49	4.05	0	0.0	0.147
10. MapPanel	13	1093	27	1	27	282.9	327.9	301	349	2.635
11. MiniMap	4	76	5	1	5	52.39	20.24	51	19.7	0.16
12. Nation	4	101	11	1	11	115.3	0.0	110	0.0	0.237
13. PlayMidi	8	58	4	2	2	20.96	8.099	0	0.0	0.12
14. ScreenManager	8	163	14	2	7	73.35	0.0	63	0.0	0.357
15. Unit	2	153	11	1	11	115.3	0.0	110	0.0	0.334
16.UnitInfoPanel	4	70	8	1	8	83.82	32.39	90	34.8	0.147
17. UnitTable	6	64	2	1	2	20.96	8.099	0	0.0	0.134
18. WorldPanel	5	230	14	1	14	146.7	56.68	133	51.4	0.513
Values through APA Tool considering each file individually	108	4121	139	20	130 Avg: 7.22	1362.3 Avg:75.68	583.01	1235 Avg: 68.61	542	9.62
Over All Project Value ignoring individual file					6.95	72.82	506.7	63	438	10.62

TABLE2: Project: Payroll_System (Analysis of Each Java File of Project)

						Through APA Tool		Through POP Builder		
Java File Name ↓	**NOM**	**LOC**	**Methods**	**Classes**	**AMC= Methods /classes**	**WMC**	**POP**	**WM C**	**POP**	**Actual Effort**
1.AddWindow	15	209	4	1	4	41.91	32.39	36	27.8	0.464
2.clsConnection	3	44	1	1	1	10.49	0.0	0	0.0	0.0903
3. clsSettings	11	70	8	2	4	41.91	0.0	36	0.0	0.1471
4. DeleteWindow	14	195	3	1	3	31.43	24.29	0	0.0	0.431
5. EditWindow	15	246	4	1	4	41.91	32.39	36	27.8	0.55
6. Emprptwindow	16	489	4	1	4	41.91	32.39	36	27.8	1.132
7. LoginFrame	16	137	5	2	2.5	26.20	20.25	0	0.0	0.298
8. MainMenu	26	382	12	2	6	62.87	48.59	63	48.7	0.874
9. PrintWindow	13	257	5	2	2.5	26.19	30.37	0	0.0	0.576
10.SettingWindow	18	718	14	1	14	146.7	170.1	133	154	1.69
Values through APA Tool considering each file individually	147	2747	60	14	45 Avg: 4.5	471.5 Avg: 52.38	390.7	340	286.3	6.25
Over All Project Value ignoring individual file					4.29	44.9	312.33	36	250.4	6.93

Figure 2: POP calculation for Payroll system through APA tool for individual java files

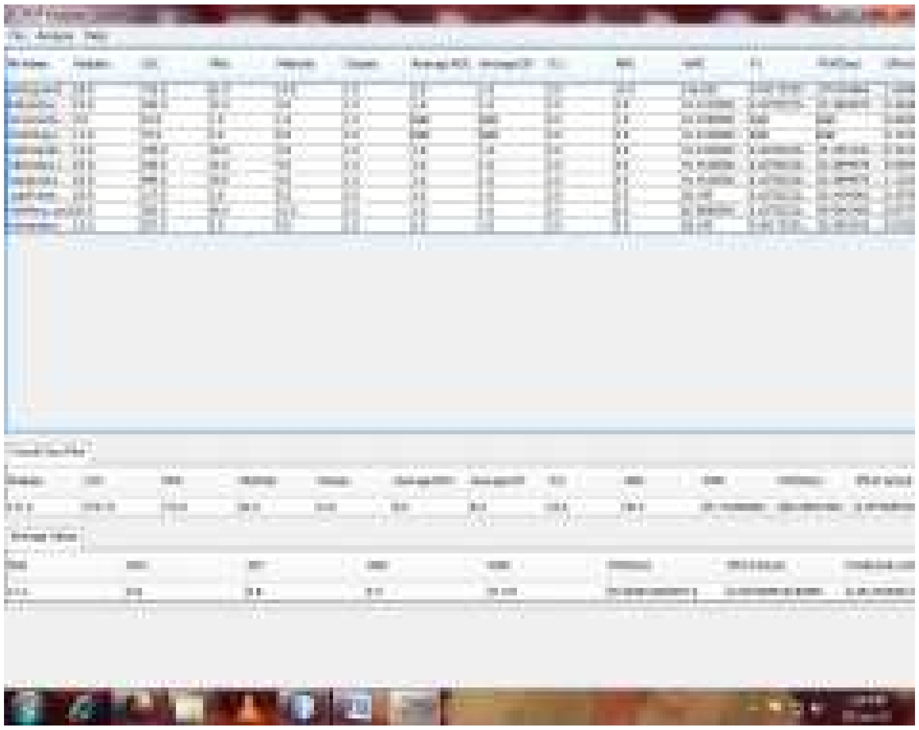

Figure 3: Overall Analysis of Payroll system through APA tool

V ANALYSIS RESULTS

The following table provides the values for the SLOC, POP metrics from APA Tool considering all files and actual effort for the two projects considered for the study.

Table 3: Summary of Project Metrics

Project Attributes	Civilisation_game_java (Project A)	Payroll System (Project B)
Source Line of Code (SLOC)	4121	2747
Total Java Files	18	10
Total Number of classes	20	14
Total Number of Methods	139	60
Average DIT	1	1
Average NOC	1	1
WMC	72.82	44.9
TLC	18	18
POP count from APA considering each java file individually	583	390.7
POP count from APA considering whole project together	506.47	312.33
POP count from POP builder developed by Price system	438	250.4
Actual Effort	10.62	6.93

Since the project size of both the projects considered for study is in the range 2-50 KLOC hence we considered them as organic type [6] [7] and used the below mentioned formula for actual effort calculation:

$$\text{Actual Effort} = 2.4(\text{KLOC})^{1.05}$$

At this point we summarize the size in terms of SLOC, actual effort and estimation through POP count. These statistics are presented as ratios:

(SLOC Project A) : (SLOC Project B) = 1.53
(Actual effort Project A) : (Actual effort Project B) = 1.532
(POP through APA for overall Project A) : (POP through APA for overall Project B) = 1.62
While considering individual java files:
(POP through APA for Project A) : (POP through APA for Project B) = 1.49
However POP count ratio through POP builder of price system =1.75

VI EVALUATION RESULTS

1. A system should be divided into several subsystems and each subsystem could be divided into several stages according to time. This could be the refinement of use of POP[2]. In our tool we split our systems or modules into sub modules as we have calculated POP count of each project on the basis of its individual java file which gives better results for the overall estimation of POP.
   From the SLOC ratio it seems that project A is 1.5 times bigger than project B. From the calculated efforts project A seems to take 1.532 times as many days to develop than project B. When we compare with the similar ratio of POPs obtained from our APA tool for overall project, the result is 1.62. However this is 1.49 when considering POP calculation for individual java file, which is more close to efforts ratio. **Hence by considering each java file of a project for calculation of POP and then combining the result give better estimation of effort and size of the project.**
2. The POP count ratio obtained from the previous tool POP builder for the same projects is 1.75. As effort and POP count are proportional hence we can claim that our APA tool gives more accurate POP count than the POP builder.
3. The previous POP builder made by Price system considered only those files of a project for POP calculation whose AMC values comes out to be>= 3.72 otherwise the file is ignored by the tool, by giving 0 POP count which is not the case in APA tool, hence give more accurate result. This can be verified from Table 1 and 2 for certain java files.
4. Lastly, the APA tool is more user friendly, readily accessible to the user and having better GUI to see more details for POP Calculation.

VII CONCLUSION AND FUTURE WORK

Here an APA (Automated Predictive Object Point Analyzer) tool has been made. POP metrics have been measured for two java projects through this tool. The results were analyzed in terms of size and efforts. The conclusion that could be drawn from this study is that the POP metric is a good indicator of software size which can be easily seen through the results of POP calculations of APA tool. Hence validate the POP metrics. This is so easy to use that the tool would ease the introduction of POPs into an environment that currently has historical data in terms of some more traditional metrics. It was felt during analysis of various projects through APA that the POP count calculations can be simplified further to give more understandability of the system and results should also been validated. Lastly, this study can be followed up with another which includes the model necessary to map the POP metrics to measure software cost and quality. Another future study prospect would be to have the data set as projects with identical requirements done in different object oriented languages. This would help us to ascertain that the POP metrics are capable of predicting the quality of software across the object oriented language.

REFERENCES

[1] A.F. Minkiewicz, ”Measuring object-oriented software with predictive object points’, Proc. ASM'97-Applications in Software Measurement, Atlanta.

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

[3] M.Haug, E.W. Olsen,L. Bergman,”Software Best Practices”, Springer 2001, Pg 159-170.

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

[5] Tools Rüdiger Lincke, Jonas Lundberg and Welf Löwe, ”Comparing Software Metrics Tools”, ISSTA’08, July 20–24, 2008, Seattle, Washington, USA.

[6] Roger S. Pressman, Software Engineering – A Practitioner’s Approach, McGraw Hill International Edition, 5th Edition, 2001

[7] Ian Sommerville, software Engineering, Pearson Education Asia, 6th Edition, 2000.

[8] http://sourceforge.net.(For java and c++ Projects)

[9] Analyst4j tool at http://www.codeswat.com

[10] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite fo Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476 -493, June 1994.

[11] G. Booch, Object Oriented Analysis with Applications - 2nd Edition. Redwood City, CA, USA, Benjamin/Cummings Publishing Co. Inc., 1994,

[12] 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.

[13] Lorenz Jeff Kidd,” Object Oriented Software Metrics : A Practical Guide, Prentice Hall, Englewood, NJ.

[14] Arup Kumar Bhattacharjee and Soumen Mukherjee, “Object Oriented Design for Sequential and Parallel Software Components”, International Journal of Information Technology and Management Information Systems (IJITMIS), Volume 1, Issue 1, 2010, pp. 32 - 44, ISSN Print: 0976 – 6405, ISSN Online: 0976 – 6413.

[15] Anand Handa and Ganesh Wayal, “Software Quality Enhancement using Fuzzy Logic with Object Oriented Metrics in Design”, International Journal of Computer Engineering & Technology (IJCET), Volume 3, Issue 1, 2012, pp. 169 - 179, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.