Journal of the Nigerian Association of Mathematical Physics
Volume **18 (May, 2011), pp 429 – 434**

Software Power Metric Model: An Implementation

Veronica V.N. Akwukwuma and Emmanuel A. Onibere

Department of Computer Science,
University of Benin, Benin City, Edo State, Nigeria.

Abstract

In an earlier work, we gave a precise definition of Software power, and also proposed a function oriented metric for measuring Software power. In this paper, we implemented the proposed model. The software application used to implement the proposed model is an application Software called 'DISCREPANCY', developed by the Computer Center of the University of Benin for the Audit department of the University. Five (5) computers with different processor speeds were used to run DISCREPANCY and the execution time (TIME) in each case was recorded. We then obtain the application functions point count. Our result shows that the proposed metric is computable, consistent in its use of unit, and is programming language independent

Keywords: Software attributes, Software power, measurement, Software metric, function point.

1. Introduction

The Audit department of the University of Benin has a Database of all the staff of the University which is normally updated from time to time when there is staff promotion or staff retiring from the services of the University. The audit department of the University has application Software called 'DISCREPANCY', developed by the Computer Center of the University of Benin for the use of the audit department . At the end of every month, the Bursary department sends a copy of the salary information for that month to the audit department. The Software, 'DISCREPANCY', compares the allowances paid to each staff with the data existing in the database in order to determine if there is any discrepancy in the allowances paid. If any, the Software detects such discrepancies and generates a report querying the Bursary department.

This application was chosen as a basis for implementing the proposed model of software power because of the proximity of getting all the information needed about an application before its function points count can be computed. This is useful in its own right to verify that the proposed model meets the specified metric constraints.

2. Background

According to Fenton and Pfleeger [1] Software attributes are induced by principles, and they are associated with components. Attributes are evident in degrees, that is to say they can be present to high degrees, lacking or absent.

Gilb [2], noted that measurement is important, just as it is in other fields of science and engineering. He emphasized the fact that measurement is not a substitute for human insight, but is a tool for aiding our understanding of what we do. Symons [3] opined that, to make Software requirements unambiguous, traceable and testable, the characteristics should be made measurable.

Measurement is not only useful but also necessary. After all, how can you tell if your project is healthy if you have no measures of its health?

Green [4], referring to the Little Oxford Dictionary, explained that Quality is a noun meaning "degree of excellence". Excellence is defined as "surpass merit", Merit as "goodness", and goodness as "virtue". He further explained that, the Software has quality to the extent that it provides value to some living, breathing people with choices and options. If another program solves a similar problem in a way that the person values more, it has quality. Green [4] is of the opinion that just as quality is not an intrinsic value, it is not static. Quality is dynamic and an attribute is "identifiable.

Although the terms measure, measurement and metrics are used interchangeably, it is important to note the subtle differences between them. Within the Software context, a measure provides a quantitative indication of the extent, amount, dimension, capacity or size of some attribute or process. Measurement is the act of determining a measure. In [5], the IEEE standard Glossary of Software Engineering Teams {IEEE93} defined metric as "a quantitative measure of the degree to which a system, component or process possesses a given attribute". According to Pressman [5], Software engineer collects measures and develops metrics so that indicators will be obtained. He further explained that an indicator is a metric or a collection of metrics that provides insight into the Software process, a Software project or the product itself, and that insight leads to informed decision making.

Corresponding authors: ***Veronica V.N.: E-mail:*** vakwukwuma@yahoo.com, Tel: +2348033440003

According to Calvert [6], metrics are management tools that are used to estimate the cost and resource requirements of a project. Software metrics deal with measuring various aspects of computer Software and its development. He explains that Software metrics can be used to estimate project costs, to manage resources, to evaluate the effectiveness of programming

methods and the reliability of a system; it involves measure of system change, program complexity, and programming effort, correctness, testability, maintainability, reliability, and so on [7].

Metrics help us understand the technical process that is used to develop a product. The process is measured to improve it and the product is measured to increase quality [6]. Productivity and quality are measures of the "output" as a function of effort and "fitness of use" of the output respectively. For planning and estimation, historical data are used to aid in predicting more accurately. He [6] explained that Software is measured to:

- Indicate the quality of the product
- Assess the productivity of the people who produce the product
- Assess the benefits derived from new software engineering tools and methods
- Form a base line for estimation
- Help justify request for tools or training.

3. Methodology

Akwukwuma and Onibere [8] proposed a model for measuring the Software Power (P_s), as

$$\mathbf{P_s} = \frac{FP}{TIME * SPEED} \tag{1}$$

where:

$P_s \Rightarrow$ Software Power,
FP $\Rightarrow$ Number of function point count,
TIME $\Rightarrow$ Execution time,
SPEED $\Rightarrow$ Processor speed.

This represents the rate at which the Software performs work. Processor speed (SPEED) is normally given, Execution time can be recorded using a stop watch, number of function point count can be computed following the International Function Point Users Group **(IFPUG)** guidelines provided in the Function Point Counting Practices Manual **(FPCPM)** version 4.1[9] and [10].

From the model, the numerator, function point count is just a number. The processor speed is measured in hertz (Hz), which is the number of cycles per second, while the execution time is measured in seconds, in other words;

$$P_S \text{ (unit)} = \frac{\text{No. of function point count}}{SPEED\ (Cycles/\text{sec}) * \text{TIME (sec)}} = \frac{\text{No. of function point count}}{Cycles} \tag{2}$$

It is obvious here that software power is a function of function point counts and the execution time, which in turn is a function of the processor speed of the computer used to execute the program.

3.1 FRAMEWORK

Measurement Theory specifies the general framework in which measures should be defined. Intuitively, Power is a measurement concept that is considered extremely relevant to engineering products. In our earlier work [8], the framework "Property Based Software Engineering Measurement" proposed by [11] was adapted as a guide in our search for the new measure as follows:

Applying the framework, power cannot be negative (property Power l), and we expect it to be null when a system does not contain any elements (property Power.2). When modules do not have elements in common, we expect Power to be additive (property Power.3). Consequently, we defined the Power of a system S, as a function Power(S) that is characterized by the following properties Power.l to Power.3.

PROPERTY Power1: Nonnegativity. The power of a system S = <E, R> is nonnegative. $\Rightarrow$ Power(S) $\geq$ 0 (Power.I)

PRPERTY Power2: Null Value. The Power of a System S = < E, R> is null if E is empty, E = $\varnothing$ $\Rightarrow$ Power(S) 0 (Power II)

PROPERTY Power.3: Module Additivity. The Power of a system S = <E, R> is equal to the sum of the Power of two of its modules m_1 = <E_{m1}, R_{m1}> and m_2,= <E_{m2}, R_{m2},> such that any element of S is an element of either m_1 or m_2 ($m_1 \subseteq$S and $m_2 \subseteq$S and E= $E_{m1} \cup E_{m2}$ and $E_{m1} \cap E_{m2}$,=$\varnothing$)$\Rightarrow$ Power(S) = Power (m_1) + Power (m_2) (Power.III)

For instance, the power of the system S with three disjoint modules m_1, m_2, and m_3 is the sum of the powers of the three modules m_1, m_2, and m_3.

3.2 Implementation and Results

The proposed model **implementation** is as follows:
Five (5) computers with different processor speeds were used to run the application, DISCREPANCY and the execution time (**TIME**) in each case was recorded. Table 1 gives processor speed (SPEED) and their corresponding execution time (TIME).

Table 1: Processor speed and their corresponding time

COMPUTER	SPEED	TIME
Computer 1	2gHz	42mins. 59secs
Computer 2	2.26gHz	38mins.2secs
Computer 3	1000mHz	1hr. 26mins
Computer 4	851mHz	1hr. 41mins
Computer 5	796mHz	1hr. 48mins

To compute the function point counts, the application documentation and the function point analysis rule were needed. The application documentation was collected from users of the Software while the function point analysis rule was obtained from International Function Point User Group counting practices manual [9]. With the two documents made available, the computation of the function point count was carried out.
The actual calculation process of function point itself is accomplished in three main stages. These stages are to:
(i) Determine the unadjusted function points (UFP);
(ii) Calculate the value adjustment factor (VAF);
(iii) Calculate the final adjusted function points (AFP).

3.2.1 Unadjusted function points (UFP)

The first stage, determining the unadjusted function points (UFP), reflects the functionality of modules delivered to the user that they have requested and defined. The unadjusted function points (UFP) includes data (file) and transactional functions.
In this application, there were eighty (80) transactional function types and three (3) file types (3FTR). Out of the eighty transactions nineteen (19) of them were External Inputs (EI), one (1) was External Output (EO) and sixty (60) External Inquiries (EQ). A file can either be Internal Logical Files (ILF) or External Interface Files (EIF). A summary of their contributions to UFP is given in Tables 2.

Table 2: Summary of transactions' contribution to UFP

TYPE OF TRANSACTIONS	**COMPLEXITY LEVEL**			
	Low	Average	High	Total
External Inputs (EI)	-	3*4 =12	16*6 = 96	108
External Outputs (EO)	-	1*5 = 5	-	5
External Inquiries (EQ)	-	-	60*6 = 360	360
		Total No of UFP		473

In this application, three (3) file types (3FTR) were identified, two of them were Internal Logical Files (2ILF); namely Data2.Recordset (Audit file2) and Data3.Recordset (Audit file1) and one (1) External Interface File (1EIF); Data1.Recordset (Bursary file). Each of the ILFs has more than six (6) Record Element Types (RET) and each of them contained more than 51 Data Element Type (DET), hence the files were rated high and scored 15 each, making a contribution of 30 to UFP. The EIF contains over 6 RET and over 50 DET, hence it was rated high and scored 10. A summary of their contributions to the UFP count is shown in Table 3. While Table 4 gives a summary of the Unadjusted Function point (UFP) for the application.

Table 3: Summary of file's contribution to UFP

FILE TYPE	RET	DET	COMPLEXITY	SCORE
Data1.Recordset (EIF)	4,763	237	High	10
Data2.Recordset (ILF)	416	83	High	15
Data3.Recordset (ILF)	4,705	241	High	15
			File UFP count	40

Table 4: Summary of The Function Points of The Components

TYPE OF COMPONENT	COMPLEXITY OF COMPONENTS			
	Low	Average	High	Total
External Inputs (EI)	-	3*4 = 12	16*6 = 96	108
External Outputs (EO)	-	1*5 = 5	-	5
External Inquiries (EQ)	-	-	60*6 = 360	360
Internal Logical Files (ILF)	-	-	2 * 15 = 30	30
External Interface Files (EIF)	-	-	1* 10 = 10	10
		Total No of UFP		513

3.2.2. Value adjustment factor (VAF)

- The second stage determining the value adjustment factor (VAF) is an earmark of the general functionality provided to the user. The standard Documentations needed are:
  - General Specification Documentations
  - Interview with the users.

The VAF is derived from the sum of the degree of influence (DI) of the 14 general system characteristics (GSCs). The DI of each of these characteristics ranges from zero to five (0 to 5), from no influence to strong influence [10]. Each characteristic is assigned the rating based upon the interview with the users. This sum is substituted into the International Function Point Users Group (IFPUG) equation for VAF.

The IFPUG equation for Value Adjustment Factor (VAF) is:

$$VAF = 0.65 + [(\sum_{i=1}^{14} Ci) / 100] \text{.....(3)}$$

Where: C_i = degree of influence for each General System Characteristics

i = 1 to 14, representing each GSC.

$\sum$ = summation of all 14 GSC's

The 14 general system characteristics (GSCs) that rate the general functionality of the application being counted is shown in table 5.

Table 5: GSC's at a Glance (Longstreet 2004)

	General System Characteristics	Brief Description
1.	Data communication	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
2.	Distributed data processing	How are distributed data and processing functions handled?
3.	Performance	Did the user require response time or throughput?
4.	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
5.	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?
6.	On-Line data entry	What percentage of the information is entered On-Line?
7.	End-User efficiency	Was the application designed for end-user efficiency?
8.	On-Line update	How many ILF's are updated by On-Line transactions?
9.	Complex processing	Does the application have extensive logical or many user's needs?
10.	Reusability	Was the application developed to meet one or many user's needs?

11.	Installation ease	How difficult is conversion and installation?
12.	Operational ease	How effective and/or automated are start-up, backup, and recovery procedures?
13.	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple organizations?
14.	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

The detailed General System Characteristics (GSCs) for each of the fourteen questions as given in [10] were given to two of the users of the application to respond to and the average of their ratings was used to compute the VAF for the application using the IFPUG equation for Value Adjustment Factor (VAF).
Table 5 gives the average ratings by the users of the application.

Table 6: Average Score of GSC

GENARAL SYSTEM CHARACTERISTICS		SCORE As
1	Data communication	0
2	Distributed data processing	1
3	Performance	1
4	Heavily used configuration	2
5	Transaction rate	1
6	On-Line data entry	4
7	End-User efficiency	5
8	On-Line update	1
9	Complex processing	2
10	Reusability	1
11	Installation ease	1
12	Operational ease	5
13	Multiple sites	0
14	Facilitate change	2
	Total	26

The average rating score, 26 was then substituted into the equation 3 for Value Adjustment factor (VAF) as follows;

$$VAF = 0.65 + [(\sum C_i) / 100] \quad (3)$$

$$VAF = 0.65 + (26 / 100) = 0.91$$

3.2.3 Adjusted function points (AFP):
The third stage is the calculation of the final adjusted function points (AFP): the total function point count of an application is represented as follows:

$$AFP = UFP * VAF \quad (4)$$

Where AFP = adjusted function points;
UFP = unadjusted function points; and
VAF = value adjustment factor.

By substituting the values of computed UFP (=513) and VAF (=0.91) into equation (4) we obtain the application functions point count (FP) as;

$$FP = UFP * VAF \quad (5)$$

$$FP = 513 * 0.91 = 466.8$$

All definitions, rules for counting and classifying, that illustrate this process can be found in FPCPM version 4.1 [9].
The computed value of the application function point (FP = 466.83) was then substituted into the proposed model (eq.1), as well as each of the processor speeds (SPEED) and their corresponding execution times (TIME) to obtain the Software power (P_s). Table 7 gives a summary of the results with varied processor speeds.

TABLE 7: Summary of the results (varied processor speed).

COMPUTER	SPEED	TIME	Power (V_n)
Computer 1	796mHz	1hr. 48mins. = 6480secs.	90.49879

Computer 4	851mHz	1hr. 41mins. = 6060secs.	90.51669
Computer 3	1000mHz	1hr. 26mins. = 5160secs.	90.465512
Computer 4	2gHz	42mins. 59sec. = 2579secs.	90.505612
Computer 5	2.26gHz	38mins.2secs. = 2282secs.	90.51213

Table 7: reveals that with varied processor speed, the power of any given software will remain approximately constant. The little variation we see in the result resulted from the recording of execution time from different Computers used to run the application hence we accept a variation of $\pm 0.2V_n$. In implementing the proposed metric we see that the metric is computable; it is consistent in its use of unit; it is independent of the processor speed of the Computer used to run the software; it consistently measures what it is supposed to measure which is an indication that the metric is reliable. This is in line with attributes that characterize effective Software metrics (both the derived metrics and the measures that lead to it) [12]

4. Conclusion

We have used just one application to test our model because Software developers will not release the backend of their Software and without it we cannot run the program. Most importantly, what we needed was to show that the proposed metric is independent of the processor speed of the Computer used to run the Software hence we have used five different Computers with different processor speeds to run the same application to show the consistency of the proposed metric in measuring what it is proposed to measure. The obligation to measure Software power in order to improve our understanding of it is as powerful in software engineering as it is in any discipline. This is like challenging previous assumptions and ideas or concepts related to Software power and its use in the Software community. Gilb [2] wrote, "Control over our activity is proportional to our ability to measure." It is desirable to know how "usable, maintainable, reliable etc." a Software product is. By being able to also measure the power of a Software product, cost associated with it may be monitored and benefits and liabilities may become more visible.

Based on this, we suggest that the software developers compute the power of the software they develop and display same on the product as well as the cost associated with it as in other engineering products to:

- Indicate the quality of the product and
- Form a base line for estimation [6].

References

[1] N.E. Fenton and S.L. Pfleeger (1997), "Software Metrics" A Rigorous and Practical Approach, second Edition PWS publishing company.

[2] T. Gilb (1975), "Software Metrics: State of the Art," Computer Weekly, September 11, 1975, p.6.

[3] C.R. Symons (1988), Function Point Analysis: Difficulties and Improvements, in IEEE Transactions on Software Engineering [ISE] vol.14, pp.2 – 11.

[4] R.G. Green (1990), Improving Software quality. "Steps to Software quality". pp.1-12. Available on line at http://www.robelle.com/quality.html.

[5] R.S. Pressman (2001), Software Engineering: A Practitioner's Approach, 5th Edition McGraw Hill. pp. 81 – 89.

[6] D. Calvert (1996) "Software Metrics". Pp.1 – 7. Available online at http://hebb.cis.ugguelph.ca/deb/27320metrics1.html,

[7] M.L. Cook (1982), Software Metrics: An Introduction and Annoted Bibliography. In ACM SIGSOFT Software Engineering Notes vol. 7, Issue 2 (April). Pp 41- 60.

[8] V.V.N. Akwukwuma and E.O. Onibere (2010) "Metric for Measuring Software Power" Journal of the Nigerian Association of Mathematical Physics. Vol. 17 November, pp.353-358.

[9] FPCPM. (1999) Function Point Counting Practices Manual, Version 4.1, January. 104 pp available online at http://www.ifpug.org/publications/manual.htm

[10] D. Longstreet (2004) Function Point Analysis Training Course. Longstreet Consulting Inc. 11 pp. Available online at http://www.SoftewareMetrics.Com.

[11] L. Briand, S. Morasca and V. Basli (1996), "Property Based Software Engineering Measurement." IEEE Transactions on Software Engineering, vol. 22, no.1.pp. 68-86, Jan.

[12] L.O. Ejiogu (1993), Five Principles for the Formal Validation of Models of Software Metrics. ACM SIGPLAN Notices, vol. 28, No. 8, August. Pp. 67-76.