Journal of the Nigerian Association of Mathematical Physics
***Volume* 17 (November, 2010), pp 353 - 358**
© J. of NAMP

Metric for Measuring Software Power

Veronica V.N. Akwukwuma and Emmanuel E. Onibere

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

Abstract

The term "power" has been used to describe Software in Software community especially Software vendors. However, there has been no formal definition of Software power, nor has there been any scientific method of determining Software power. It is therefore the objective of this paper to examine the attributes of Software product with a view to determining how they are defined and measured and to formally define precisely Software power, and propose a metric for measuring Software power.

We give a precise definition of Software power, and also propose a function oriented metric for measuring Software power.

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

1.0 Introduction

A large number of measures have been proposed in the literature to measure Software attributes, ([1], [2], [3], [4]). It was observed that power has not been mentioned anywhere as one of the attributes of Software, neither has it been defined or measured in Software engineering though it is commonly used by Computer vendors to describe their software. Power was identified as a composite attribute of Software hence it was added to the growing list of software attributes [5]. The presence of these sub-attributes of power in a software product only indicate the presence of the quality attribute, power, of the software product but it does not in anyway quantify the Software power. In other words it does not state the measure of the attribute that makes it possible to compare the quality of the software with other Software in terms of their powers. Fenton and Pfleeger [2] in quoting Flinkelstan said, "What is not measurable make measurable." This phrase, attributed to Galileo Galilei suggests that one of the aims of science is to measure attributes of things in which we are interested. Thus, as scientists, we should be creating ways to measure our world, and where we can already measure we should be making our measurement better.

There is variety in how we use measurement, but there is a common thread running through in every case. Some aspect (attribute) of a thing (entity) is assigned a descriptor that allows us to compare it with others. The rules for assignment and comparison may not be explicit in all cases but it will always be clear that we make our comparisons and calculations according to a well-defined set of rules. It was noted in [2] that we are now able to measure attributes that were previously thought unmeasurable in the physical sciences, medical sciences, economics, and even some social sciences.

Kafura [6] opined that to make Software an engineerable product, it is important that the designers, implementors and maintainers of Software systems be able to express the characteristics of the systems in objective and quantitative terms. Quantitative measures of quality are collectively referred to as Software metrics.

Unlike other engineering disciplines, Software engineering is not grounded in the basic qualitative laws of physics. Absolute measures are uncommon in the Software world. Instead we attempt to derive a set of indirect measures that lead to metrics. Because Software measures and metrics are not absolute they are open to debate [2, 4].

Corresponding author: E-mail; vakwukwuma@yahoo.com ; Tel. +2348033440003

Our objective in this paper is to propose a function oriented metric for measuring Software "power". The aim is that measurement makes concepts more visible and therefore more understandable and controllable.

2. Background

According to [2], Software measurement was once an obscure and esoteric specialty, but it has now become essential to good software engineering. Many of the best software developers measure characteristics of the Software to get some sense of whether the requirements are consistent and complete, whether the design is of high quality, and whether the code is ready to be tested. An effective project manager measures attributes of process and product to determine when the software will be ready for delivery, and whether the budget will be exceeded. Informed customers measure aspects of the final product to determine if it meets the requirements or expectations and if it is of sufficient quality, and maintainers must be able to assess the current product to see what should be upgraded and improved.

Measurement, according to [7], in most general terms, can be regarded as the assignment of numbers to objects in accordance with some rules (measurement functions). The property of the objects that determines the assignment according to the rules is called magnitude, the measurable attribute; the number assigned to a particular object is called its measure, the amount or degree of its magnitude. It is to be noted that the rule defines both the magnitude and the measure.

According to [8] the importance of measuring attributes of known objects in precise quantitative terms has long been recognized as crucial for enhancing the understanding of an environment. Dhyani et al. [8] went on to say that this note has been aptly summarized by Lord Kelvin:

> "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you can not express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of science."

Rigorous measurement of Software attributes can provide substantial help in the evaluation and improvement of Software products and processes. Symons [9] opined that, to make Software requirements unambiguous, traceable and testable, then make the characteristics measurable.

As a result of this, a few proposals were made [3] based on:

- ✓ Measurement Theory [10]
- ✓ Axiomatic approach [11]

Measurement Theory specifies the general framework in which measures should be defined. First, an empirical relation system should be specified to define the relations among the entities as far as the studied attribute is concerned. Then, a numerical relation system is defined, to provide values for the measures of the attribute and relations among these values. A measure is the link between the empirical relation system and numerical relation system that maps entities into values and empirical relations among entities into formal relations among values. Axiomatic approaches formally define desirable properties for the measures of a given Software attribute. These properties are properties of a numerical relation system of measures. However, they indirectly affect the empirical relation system.

According to [12], measurements can be either **direct or indirect.** Direct measures are taken from a feature of an item. Indirect measures associate a measure to a feature of the object being measured. **Direct measures in a Software product include:** lines of codes (LOC), execution speed, memory size, and defects reported. **Indirect measures include:** functionality, quality, complexity, efficiency, reliability, and maintainability. **Size-oriented metrics** are used to collect direct measures of software engineering output and quality. **Function-oriented metrics** provide indirect measures which focus on functionality and utility.

Rule [13] opined that what the Software community needs, is a measure of the quantity of information processing functionality the customer requires of the Software, independent of the technology used and the people who produce it. This is what [14] called a measure of 'functional size'. According to [13] Functional size measurement methods are such a family of techniques. This includes Albrecht's 'Function Point Analysis' (FPA), Symons' 'MKII Function Point Analysis', Boeing's '3D Function Points' (3DFP), and 'Common Software Metric International Consortium - Full Function Points' (COSMIC-FFP). The first function-oriented metric was originally proposed and developed by Alan Albrecht of IBM in 1970. His insight was to measure the size of the functional requirements,

Corresponding author: E-mail; vakwukwuma@yahoo.com ; Tel. +2348033440003

rather than to count the number of lines of code, thus moving away from a dependence on specific technology, people and development methods. Later work by Albrecht, in conjunction with John Gaffney, stabilized and popularized the technique, leading in 1984 to the formation of the International Function Point Users Group (IFPUG) in the USA. Function points represent the functionality of a system. Many empirical studies point to an existing relationship between these points and amount of work-effort necessary to develop them. [15, 16 and17]

3. Framework

Intuitively, Power is a measurement concept that is considered extremely relevant to engineering products. Here, 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) ≥ 0 (Power.I)

PROPERTY Power.2: 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.

3.1 The proposed defined metric of Software power

In order to give Software Power an identity and thus make it easily recognizable, the theoretical definition of Software power was proposed as follows:

Software power (P_s) is defined as the effectiveness with which the Software is able to interact with or use various items of the Computer System during its operation in relation to the volume and complexity of the items per unit time.

Effectiveness is defined as the **ability** to bring about the result intended [18]. Notationally, we use the symbol $\mathbf{P_s}$ (upper case P, subscript s) to represent Software power. Effectiveness is defined as the ability to bring about the result. This **ability or effectiveness** of the Software can be likened to the energy or the ability to do work in Mechanical power. This ability of the Software can be identified if we can break the Software system into smaller components. This can be achieved with the Function Point Analysis **(FPA)** technique. **FPA** is used to break systems into smaller components, so they can be better understood and analyzed. In Software systems, these components are called elementary processes. When these elementary processes are combined, they interact to form what we call a Software system or application. Function Points can be used to size Software applications. Sizing is an important component in determining productivity (Output/Input), predicting effort, understanding unit cost, and so on and so forth [19 and 20]. Similarly, sizing (functional size) is an important component in determining ability to do. Software Power ($\mathbf{P_s}$) in this sense is the ability of the Software to solve or the effectiveness to perform a given task per unit time relative to the processor speed. We say relative to the processor speed in that external quality can only

Corresponding author: E-mail; vakwukwuma@yahoo.com ; Tel. +2348033440003

be assessed for a complete hardware/Software system of which the Software product is a part. According to [21], external metrics are applied when executing the Software. The values of external measures necessarily depend on more than the Software, so the Software has to be evaluated as part of a working system. Hence Software power can be measured in terms of the function point count of the Software.

We give the proposed **metric** as follows:

$$\mathbf{P_s} = \frac{FP}{TIME * SPEED} \qquad \text{... (3.1)}$$

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 while Processor speed (SPEED) is normally given, Execution time can be recorded using a stop watch, and 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[22]. Hence, we can derive the power of any software.

3.2. The unit of Software power

From equation 3.1, 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, that is to say;

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

$$= \frac{\text{No. of function point count}}{Cycles}$$

That is, Number of function point (FP) count (the numerator) per cycle (the denominator).

We adopted the name **Vion** (pronounced as v-e-e-on, and written for short as $V_{n;}$ upper case 'V' and subscript 'n'), as the unit of Software Power. The term **Vion** is defined as the number of function point count of a Software per mega cycle (the amount of work done by the Software per 1,000 000 cycles). That is to say, 10^6 V_n is equivalent to 1FP/mcycle. Multiplying what ever value obtained in fp/mcycle by 10^6 will convert the value to $V_{n.}$

The numerator, Function point (FP) of equation (3.2) can be computed using the following equation;
AFP = UFP * VAF **... (3.2)** [23]

Where **AFP (FP)** = adjusted function points
UFP = unadjusted function points
VAF = value adjustment factor

Conclusion

Our proposed metric:

Corresponding author: E-mail; vakwukwuma@yahoo.com ; Tel. +2348033440003

- is computable
- is consistent in its use of unit
- is programming language independent
- Empirically and intuitively persuasive (i.e. metric increases in value as the number of elementary processes increases)

These characteristics are in line with what Ejiogu in [4] called the attributes that characterize effective Software metrics (both derived metrics and the measures that lead to it). Conclusively, this research work is viewed as an initial step. We believe the metric we proposed is generally desirable and relevant. Hopefully this research work will provide a foundation which can be built upon.

References

[1] 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.

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

[3] S. Morasca and L.C. Briand (1997) "Towards a Theoretical Framework for Measuring Software Attributes." IEEE Proceedings of the 4th International Software Metrics Symposium (METRICS '97), pp. 119 -126

[4] R.S. Pressman (2001). Software Engineering: A Practitioner's Approach, 5th Edition McGraw Hill. New York.

[5] E.E. Onibere and V.V.N. Akwukwuma (2010) Power: An Attribute of Software? (In Press)

[6] D. Kafura (1985), "A survey of Software Metrics": In Communications of the ACM 0-8991-10-9/85/1000-0502. PP.502 – 506.

[7] B.R. Boyce, C.T. Meadow and D.H. Kraft (1994), Measurement in Information Science. Academic Press Inc. Orlando, Fla.

[8] D. Dhyani, W.K. Ng and S.S. Bhowmick (2002), "A Survey of Web Metrics". ACM Computing Surveys, vol.34, no.4 December, pp.469-503.

[9] C. Symons (2001), Software Measurement and process in Software Measurement Service 143 High Street, Edimbough, Kent TNB SAX. UK. Issue 8, spring p.3

[10] N.E. Fenton and B. Kitchenham (1991) "Validating Software measures." Journal of Software Technology, verification and Reliability, vol. 1, no. 2, pp. 27-42.

[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.

Corresponding author: E-mail; vakwukwuma@yahoo.com ; Tel. +2348033440003

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

[13] P.G. Rule (2001) “The Importance of the Size of Software Requirements”. Presented at the NASSCOM Conference, Hotel Oberoi Towers, Mumbai, India, 7th -10th February, pp. 15. Available online at http://www.software-measurement.com/

[14] ISO/ICE 14598 International Standard, “Standard for Information Technology – Software product evaluation – part 1: General overview”. Available online at http://www.hmaster.com.glossary/175.html

[15] A. J. Albrecht and J. Gaffney Jr. (1983) “Software Functions, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE Transactions on Software Engineering, vol.9, n.6, Nov., pp. 639 – 648

[16] J.M. Desharnais and G. Hudon (1990), Adjustment Model for Function point Scope Factor – A Statistical Study, in IFPUG, Montreal. 29 pp.

[17] C.F. Kemerer (1987), “An Empirical Validation of Software Cost Estimation Models”, Communication of ACM. Vol. 30, no. 5, May. Pp. 416 – 429

[18] A.S. Hornby (2001), Oxford, Advanced learner’s Dictionary of current English. Oxford University Press.

[19] C.F. Kemerer (1992) Measurement for Improved Software Development, in Conference at the Computer Research Institute of Montreal (CRIM), Montreal, Canada, pp.14.

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

[21] N. Bevan (1997) Quality in use: “Incorporating human factors into the Software engineering lifecycle”. pp.9. Available on line athttp://www.usabilitynet.org/papers/qiuhf97.pdf

[22] FPCPM. (1999) Function Point Counting Practices Manual, Version 4.1, January. 104 pp.

[23] IFPUG. (1999) Function Point Counting Practices Manual, Release 4.0, Westerville, Ohio.

Corresponding author: E-mail; vakwukwuma@yahoo.com ; Tel. +2348033440003