A Metric in Global Software Development Environment.

Lauretta O. Osho, B.Tech.[1]; Sanjay Misra, Ph.D.[2]; and Oluwafemi Osho, M.Tech.*[3]

[1] Department of Computer Science, Federal University of Technology, Minna, Nigeria.
[2] Department of Computer and Information Sciences, Covenant University, Otta, Nigeria.
[3] Department of Cyber Security Science, Federal University of Technology, Minna, Nigeria.

E-mail: femi.osho@futminna.edu.ng*

ABSTRACT

Metrics and measurement techniques for managing projects in global software development (GSD) environment. GSD, as a practice, offer a variety of advantages, as well as limitations. This paper mathematically models two common concepts in GSD environment, under the resource requirements of software development, namely coherence and collocation. Both terms have been used informally to explain some results obtained from on-site studies in respect of speed of project execution. The logic consists in exploiting the merits of GSD, whilst mitigating its demerits. Because this paper would only seek to introduce the metric, further studies are recommended to further explore the feasibility of the model, and possible enhancements to aid its efficiency.

(Keywords: global software development, GSD environment, measurement techniques, metrics, coherence, collocation, software engineering)

INTRODUCTION

As the name suggests, global software development involves the development of software across globally distributed sites. This contrasts the traditional one-site location of software developers. Basically, subcontractors, third parties suppliers, and in-house developers work independently in separate geographical locations, albeit collaboratively to develop a product [10] [20].

Globally distributed software development offers some benefits including access to specialized labor pool, reduced development costs due to varying labor costs in different countries, proximity to customers, and round-the-clock development offered by time-zone differences [5][14].

However, it has its accompanying setbacks. Language, cultural and communication barriers, and increased organisational processes complexity as a result of distance [5][7][8][9] are some of these. As a matter of fact, GSD projects have also been found not to be immune to challenges that often face one-site projects: budgets and schedule constraints, and failure to meet overall project targets [11]. These are understandable considering the fact that distance does not aid effective communication, especially among people who hardly know each other. Thus, globally distributed projects can be expected to be more challenging to manage [4]. It therefore becomes necessary, for effective project control, to have an effective monitoring and reporting system in place. Measurements and measurement techniques (metrics) can be used as means to achieve this end [17].

Measurements and measurement techniques provide a platform to monitor and control production, thereby providing some useful basis for management decision making [1].

SURVEY OF FEW RELATED WORKS

We survey the traditional categories of measurement techniques, namely process, product, and resource metrics [6].

Process

Walgers [21] developed the Problem-Goal-Pattern-Measurement (PGPM) technique, which is strongly based on the Goal-Question-Metric approach [2] [6] [19]. In respect to software development, process pattern implies a general solution garnered from documented solutions to problems occurring during software development

that is applicable to similar development process. This approach helps select the right pattern to apply when used. It helps to devise goals for problems areas requiring advancement. However, an evaluation of current situation must be determined.

Herbsleb and Mockus [8] used survey data and data from the change management system to model delay extent in a globally distributed software development organisation. A change management (CM) system was used to manage development work. It provides functionality for code versioning and managing simultaneous changes structurally.

The delay extent model revealed that it took a distributed team about two and one-half the time to complete a work by a collocated team. The findings were confirmed when the change data analysis were replicated in a different organization with dissimilar product and site.

Product

In this section, we consider two amongst the numerous cognitive complexity measurement techniques applicable for the maintaining of quality software.

Misra [15] considered the cognitive weight complexity measure to rely upon the cognitive weights of basic control structures. He defined the total cognitive weight of a software component Wc as "the sum of cognitive weight of its q linear blocks composed in individuals BCS'S." Mathematically, this can be represented by:

$$W_c = \sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{i=1}^{n} W_c(j.k.i)\right] \quad (CWU) \quad (1)$$

Where each block may consists of m layers of nesting BCS's, and each layer with n linear BCS's.

He expressed the unit of cognitive weight complexity measure as "the cognitive weight of the simplest software component i.e. a linear structured BCS", represented mathematically by:

$$\text{CWCM} = f(W_{bcs}) = 1 \text{ Cognitive Weight Unit (CWU)}$$

Shao and Wang [18] defined the cognitive functional size of a basic software component that only consists of one method, S_f, as "a product of the sum of inputs and outputs ($N_{i/o}$) and the total cognitive Weight." This they expressed mathematically as:

$$S_f = N_{\underline{i}}_{o} \times W_c = (N_i + N_o) * \left\{\sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{i=1}^{n} W_c(j.k.i)\right]\right\} \quad [CWU] \quad (2)$$

The unit of CFS is equivalent to that of the CWU.

Resource

Boehm et al. [3] proposed the Constructive Cost Model, termed COCOMO II, for estimating cost and schedule for projects, which uses the size of the project, for instance Source Lines of Codes (SLOC). This is used actually to determine the effort required to developsoftware. It is expressed as:

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM^{i} \quad (3)$$

Where, PM = Person Month

This is the number of hours that a person spend to complete a given task presented in a calendar month. It is used to directly derive the project's cost.

A = *2.94* (for COCOMO II)

Size is estimated by Kilo Source Lines of Code (*KSLOC*) measure or unit.

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j \quad (4)$$

EM = Effort Multiplier,
B = 0.91,
n = 17 (for Post-Architecture Model).

Kemere [12] presented an algorithm method called SLIM, developed by Larry Putnam, used for approximating project's efforts and schedule. SLIM also utilizes SLOC to measure project's overall size. It is represented by two equations: the first for allocating Productivity Parameter (PP), expressed in man years, which would be required in the second equation for calculating effort.

$$PP = \frac{Size_{SLOC}}{(E_{Man,Year}/B^{1.13}) \times Duration(Y^{4/3})} \quad (5)$$

$$E_{Man,Year} = \left[\frac{Size_{SLOC}}{PP \times (Duration_{Years})^{3/4}}\right]^{3} \quad (6)$$

Where, $E_{Man,Year}$ represents the amount of effort required to accomplish a given task in a man-year unit or measure.

Muhairat *et al* [17] investigated the effects of different factors on the accuracy of effort estimation methods in GSD environments. Precisely, COCOMO II, SLIM, and ISBSG methods of estimating projects efforts were considered. They discovered the estimation methods were less accurate in determining the actual time of completion of some software development projects. The main factor that affected this outcome included the project environment. They concluded that developing a software in a GSD environment always require more, in effort and time, to complete.

PROBLEM DEFINITION

Successful management of projects evidently requires knowledge of required resources. But these resources are often limited. Hence, it is necessary to adopt a mean of effectively allocating these limited resources to actualize project completion. Most software projects are component-based. The overall completion of a software project is the cumulative of the time expended in completing the different software components. Thus, effective allocation of resources to each component is very crucial to the overall completion of any software project.

One of the merits of GSD is access to specialized labour pool [5] [14], which is an indispensable asset to any software project. However, GSD teams have been discovered to produce at a level up to 50% less than those collocated [13] [17]. As a matter of fact, as concluded by the delay extent model in [8], collocated teams could complete a given task expending just two-fifth of the time it would take a distributed time. It therefore becomes necessary to develop a mean to exploit the advantages GSD provides, whilst mitigating its demerits.

COHERENCE-COLLOCATION MODEL (CCM)

Based on several studies that have pointed out the demerits of GSD, a direct consequence of distance and other differences, Mockus and Weiss [16] recommended that "tightly coupled work items that require frequent coordination and synchronization should be performed within one site." Ebert and De Neve [5] advised "building coherent and collocated teams of fully allocated engineers." As a matter of fact, they reported that collocated teams needed less time, precisely less than half, to perform a task compared to that required by distributed teams. Herbsleb and Mockus [8] also corroborated using their delay extent model this assertion.

Coherence denotes that the number of work is split, based on product functionality, that is, it is component-based. Collocation implies the tackling of the same set of functionality or component – coherent tasks – in the same location site, while full allocation means that developers are fully engaged in a project without distraction from other projects [5]. Therefore, we define Coherence-Collocation Ratio (CCR) as the average number of coherent and collocated product developers, who are fully allocated.

The coherence-collocation model can be categorized as one of the techniques for measuring resources. This is due to the fact that it is based on the effort required to develop a software, which is the sum total of the efforts for each of the software components.

Basically, the software is broken into different components, where each component incorporates one or a group of functionality. This determines what is called the coherence ratio. The effort of each component is then determined, using any of the established means. It is necessary to note that the efforts may or may not vary for the different components. All depend on their sizes. Based on this, the required number of developers, from the pool of developers, is then

allocated fully. These processes are represented below:

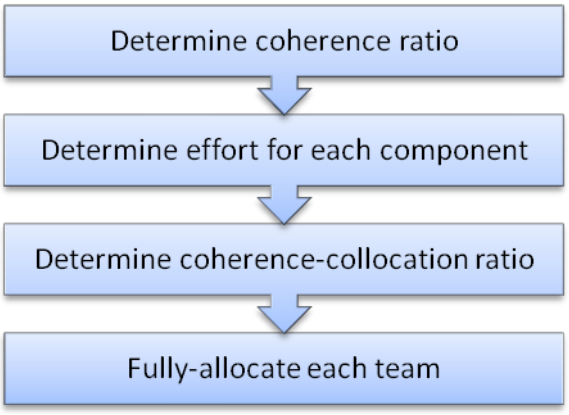

Figure 1: Coherence-Collocation Model.

Mathematical Representation

We assume the following:

i. Every software can be broken into at least two components. That is, coherence ratio is always greater than 1 for every software, and

ii. For each component, the developers are all collocated (and fully allocated). This means that collocation ratio equal 1

Consequently, we define the following terms:

$[x, y]\{x \geq 1 \text{ and } y \geq 1\} =$ x software developers are located in y different sites. In this case, we may two or more developers collocated in one of the locations.

From the above, we have the following subsets:

$[1,1] =$ only one developer is located on a site. This implies that no two developers are collocated.

$[x, 1]\{x \geq 1\} =$ one or more developers are collocated.

$[x_{max}, 1] =$ all developers are collocated.

$[1, y]\{y \geq 1\} =$ a developer is located on one or more sites. This happens when a developer moves between locations.

Component-wise, we have the following:

$h =$ Coherence ratio (this is the number of components, tasks, or sets of functionality that make up the software). This simply implies that, $h < \infty$, i.e., it is finite.

$z_i \{i = 1, \dots, h\} =$ individual component, task or set of functionality.

$[c_i, 1] \{i = 1, \dots, h\} =$ coherence-collocation ratio, i.e., number of fully allocated developers for each component, task or set of functionality.

We can deduce that, assuming the result of Herbsleb and Mockus [6], to achieve the same completion time for a software component,

$[c_i, 1] = 2.5[c_i, b]$, where $c_i = b$ (that is, no two developer of the software component are located in the same site).

The total number of software developers,

$$c_{total} = \sum_{i=1}^{h} c_i \qquad (7)$$

We assume that the number of developers that would be assigned for each software component should be determined by the estimated effort required or the complexity of the software component. We therefore say that number of developers is a function of software component. This is represented as:

$$[c_i, 1] = f(z_i) \{i = 1, \dots, h\} \qquad (8)$$

If we consider our software components in terms of complexity, using, say the cognitive weight complexity measure (CWCM) [15], we say that:

$$[c_i, 1] \propto W_c = KW_c \qquad (9)$$

Where

$$W_c = \sum_{j=1}^{q} \left[\prod_{k=1}^{m} \sum_{i=1}^{n} W_c(j.k.i) \right] (CWU) \quad (1)$$

$$k = \frac{c_{total}}{W_c^{total}} \qquad (10)$$

K is the number of product developers required per Cognitive Weight Unit (CWU), and W_c^{total} is the cognitive weight complexity measure of the entire software.

On the other hand, if we consider the components in terms of required completion effort, using SLIM [3], we would have that:

$$[c_i, 1] \propto \mathrm{E}_{\mathrm{Man,Year}} = K\mathrm{E}_{\mathrm{Man,Year}} \qquad (11)$$

Where,

$$\mathrm{E}_{\mathrm{Man,Year}} = \left[\frac{\mathrm{Size}_{\mathrm{SLOC}}}{\mathrm{PP} \times (\mathrm{Duration}_{\mathrm{Years}})^{3/4}}\right]^3 \qquad (6)$$

(SLOC is Source Lines of Code, and PP is known as the Productivity Parameter).

$$k = \frac{c_{total}}{E_{man,years}^{total}} \qquad (12)$$

K is the number of product developers required per unit estimated effort, and $E_{man,years}^{total}$ is the required effort for the entire software.

ANALYSIS OF MODELS

Looking at the two variations of the coherence-collocation ratio, that is, equations (9) and (11), the latter equation seems to be easier to deal with considering the fact that amount of effort required can be estimated more easily than one can calculate the complexity measure while the software is yet to be developed. Hence, we adopt equation (11). Combining therefore equations (11) and (12), we have that the coherence-collocation ratio is given as:

$$[c_i, 1] = \frac{c_{total}\,\mathrm{E}_{\mathrm{Man,Year}}}{E_{man,years}^{total}} \qquad (13)$$

For example, according to David A. Wheeler [22], assuming conventional proprietary means were used to develop the Red Hat distribution of the Linux operating system, 8,000 man-years would have been required. If we further assume a developer size of 500, we simply have that the number of product developers required to expend a unit effort (in man-years) is:

$$K = \frac{500}{8000} = 0.0625$$

If our coherence ratio, $h = 6$, and we assume different values to represent the estimated effort required for each component, $z_i\{i = 1, \ldots, 6\}$, that is, $\mathrm{E}_{\mathrm{Man,Year}}$, the coherence-collocation ratio for each component should be required as tabulated below:

Table 1: Determination of Coherence-Collocation Ratio per Product Component Estimated Effort.

h	$\mathrm{E}_{\mathrm{Man,Year}}$	$[c_i, 1] = 0.0625 \times \mathrm{E}_{\mathrm{Man,Year}}$
1	800	50
2	1200	75
3	2000	125
4	1200	75
5	800	50
6	2000	125
	$E_{man,years}^{total} = 8000$	$c_{total} = 500$

CONCLUSION

The main thrust of the coherence-collocation model suggests that parts of software should be outsourced or handled by teams, partners or organizations that guarantee collocation and full-allocation of developers. The model can be optimised to determine the best mix of developers, since it utilises effort needed vis-à-vis the amount of developers available. However, it must be emphasised, the developers for a particular component must be collocated and fully allocated.

Collocating developers can reduce significantly the cost of projects since the issue of communication gap between developers handling similar component of the project is virtually eliminated. Also, being fully allocated can help avoid distractions that would have emanated as a result of multi-component development.

REFERENCES

1. Basili, V.R. 1992. “Software Modeling and Measurement: The Goal/Question/Metric Paradigm”.

2. Basili, V.R., G. Caldiera, and H.D. Rombach. 1994. “The Goal Question Metric Approach”. *Encyclopedia of Software Engineering*, 528-532. John Wiley & Sons: New York, NY.

3. Boehm, B.W., C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, and B. Steece. 2000. Software Cost Estimation with COCOMO II. Printice-Hall: Princeton, NJ.

4. Da Silva, F.Q.B., C. Costa, A.C.C. França, and R. Prikladinicki. 2000. “Challenges and Solutions in Distributed Software Development Project Management: A Systematic Literature Review”. In: *Proceedings of International Conference*.

5. Ebert, C. and P.De Neve. 2001. “Surviving Global Software Development”. *IEEE Software*. March/April, 2011.

6. Fenton, N.E. and S.L. Pfleeger. 1998. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing: Boston, MA.

7. Herbsleb, J.D. and D. Moitra. 2001. “Special Issue on Global Software Development”. *IEEE Software*. 18(2) (March/April 2001).

8. Herbsleb, J.D. and A. Mockus. 2003. “An Empirical Study of Speed and Communication in Globally Distributed Software Development”. *IEEE Transactions on Software Engineering*. 29(6):481-494.

9. Herbsleb, J.D., A. Mockus, T.A. Finholt, and R.E. Grinter. 2003. “An Empirical Study of Global Software Development: Distance and Speed”.

10. Hyysalo, J., P. Parviainen, and M. Tihinen. 2006. “Collaborative Embedded Systems Development: Survey of State of the Practice”. In: *13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006)*. IEEE, 2006. 1-9.

11. Ilincic, R. 2008. “Examining Agile Management Methods and Non-Agile Management Methods in Global Software Development Projects”. Engineering Management Master’s Theses. Paper 1. http://hdl.handle.net/2047/d10018591.

12. Kemere, C.F. 1987. “An Empirical Validation of Software Cost Estimation Models”. In: *Communications of ACM.* 30(5):416-429.

13. Kormeren, R. and P. Parvianen. 2007. “Philips Experiences in Global Distributed Software Development”. *Empirical Software Engineering*. 12(6):647-660.

14. Lanubile, F., D. Damian, and H.L. Oppenheimer. 2003. “Global Software Development: Technical, Organizational, and Social Challenges”, ACM SIGSOFT Software Engineering Note, 28(6) Nov. 2003.

15. Misra, S. 2006. “A Complexity Measure Based on Cognitive Weights”. *International Journal of Theoretical and Applied Computer Sciences*. 1(1):1–10.

16. Mockus, A. and D.M. Weiss. 2001. “Globalization by Chunking: A Quantitative Approach”. *IEEE Software.* 18(2):30–37.

17. Muhairat, M., S. Aldaajeh, and R.E. Al-Qutaish. 2010. “The Impact of Global Software Development Factors on Effort Estimation Methods”. *European Journal of Scientific Research*. ISSN 1450-216X 46(2):221-232.

18. Shao, J. and Y. Wang. 2012. “A New Measure of Software Complexity Based on Cognitive Weights”. Retrieved on July 2, 2012, from www.ucalgary.ca/icic/files /icic/59-JECE-IEEE919.pdf

19. Solingen, van, R. and E. Berghout. 1999. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill Publishing Company: Maidenhead, UK, ISBN: 0077095537

20. Tihinen, M., P. Parviainen, R. Kommeren, and J. Rotherham. 2011. “Metrics in Distributed Product Development”. *The Sixth International Conference on Software Engineering Advances*.

21. Walgers, M.M. 2007. “Towards a Method for Improving Globally Distributed Software Development Using Quantitatively Measurable Process Patterns (The Problem-Goal-Pattern-Measurement Approach)”. Twente Student Conference on IT. Enschede (June 25, 2007).

22. http://en.wikipedia.org/wiki/Source_lines_of_code

ABOUT THE AUTHORS

Osho Lauretta Oluwafemi, is a postgraduate student of Computer science in the Department of Computer Science, Federal University of Technology, Minna, Nigeria. She holds a B.Tech. degree in Mathematics/Computer Science. Her research interests include cloud computing and software development.

Sanjay Misra, is a Professor of Computer Engineering in the Department of Computer and Information Sciences, Covenant University, Otta, Nigeria. He holds an M.Tech. degree in Software Engineering from Motilal Nehru National Institute of Technology, Allahabad India and a D.Phil. from the University of Allahabad, India. He is the author of more than 100 papers and has chaired several annual international workshops. Presently, he is Chief Editor of an *International Journal of Physical Sciences* and *International Journal of Computer Science and Software Technology* and serves as Editor/Associate Editor/Editorial Board Member for several journals of international repute. His current research covers the areas of software quality, software measurement, software metrics, software process improvement, and software project management, object oriented technologies, XML, SOA, Web Services, and cognitive informatics.

Osho Oluwafemi, is currently an Assistant Lecturer in the Department of Cyber Security Science, Federal University of Technology, Minna, Nigeria. He holds a B.Tech. degree in Mathematics/Computer Science and an M.Tech. degree in Mathematics. Before joining the institution, he served as Head of the IT Department of one of the leading mortgage banks in Nigeria. His research interests include information security, data mining/machine learning, and software development.

SUGGESTED CITATION

Osho, L.O., S. Misra, and O. Osho. 2013. "A Metric in Global Software Development Environment". *Pacific Journal of Science and Technology*. 14(2):213-219.

Pacific Journal of Science and Technology