Sheeba Praveen,Dr. Rizwan Beg,Dhruba Shankar Ray / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp.3052-3056

# Software Performance Measurement Metrics and Recitation

**Sheeba Praveen**
Dept. CSE,Integral University
Lucknow ,U.P.

**Dr. Rizwan Beg**
Dept.CSE & IT,
Lucknow,U.P.

**Dhruba Shankar Ray**
Integral university Dept. of IT
Lucknow,U.P.

**Abstract— measuring software attributes with the purpose of improving software product quality and project team productivity has become a primary priority for almost every organization that realize on computer. This paper examined the concept software complexity and its effect on software project, concerning productivity and quality of real time software system. And these attribute will help us to increase the performance of system. Software project are influenced by several external and internal factors generally gathered in the term software complexity. So this paper discussed different parts of software complexity and their effects on software productivity and quality. We related mainly algorithmic and structural complexity to productivity and quality. Our hypothesis that Full Function Point analysis (FFA) of Function point Metrics was most suitable for size measurement which effect productivity and McCabe metric suite of Object Point Analysis (OPA) is most suitable for error-proneness measurement which effect of real time system.**

***Keywords— Reliability, structural complexity ,Productivity, algorithmic complexity, FPA, FFP, OPA, McCabe's number metric.***

## I. INTRODUCTION

Software complexity is the degree of difficulty in analyzing, maintaining, testing, designing and modifying software. It is divided into two main classes' complexity of problem and complexity of solution. There are different types of software complexity, computational complexity, Algorithmic complexity, structural complexity, Cognitive complexity

The effect of complexity was divided into two main parts:

- Error –proneness means a program that is more complex than another is also more likely to contain more errors, influences usability, reliability, and need of change of the system.
- Size influences the maintainability, understandability, and computational power needed for implementing the system.

Here we describe software complexity with a model in Fig. 1 that was divided into two main tracks one of the tracks bears upon error-proneness which in the end leads to quality issues. The other track considered size and effort which influences productivity. Our mission of this paper is only to discuss software quality and productivity so according to our software model we only discuss structural complexity and algorithmic complexity.

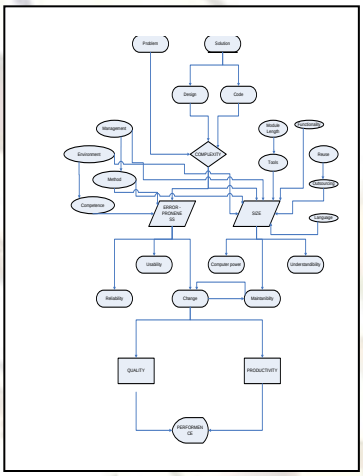


Figure 1. Software Complexity model

## II. PRODUCTIVITY

### A. *Algorthimic complixity*

We measure algorithmic complexity and how it influences software size and different functional metrics that measures productivity of software project.

A measure of productive efficiency calculated as the ratio of what is produced to what is required to produce it[1].

Productivity=Size (LOC)/Effort (p/m)

Some researchers (Fenton & Pfleeger, 1996; Möller&Paulish, 1996; Jones, 1996) have proposed that function-points-based measure reflects more accurately the value of output. It can be used to assess the productivity of software-development staff at any stage in the life cycle [2].

Productivity= Function points implemented (FPA) / Person Months

### B. *Proposed Method of Productivity*

Here we proposed that Full Functional point (FFP) metric gives more accurate result than FPA in terms of productivity.

Productivity=FFP/PM

*C.* ***Different categories of size measurement***

| Size Metrics | Critics |
|---|---|
| Plane Size Metric (LOC) | Language Dependent and using Reusable code. |
| Function Size metric | Not be applicable for real time system and other scientific system. |
| IFPUG's Function point(FPA) | Due to lack of complexities issues it is not be applicable to real time system and other scientific environment. |
| SPR's Function Point | Overall impact of complexity on source code size is not exact. |
| 3D function point | It is Still an experimental approach so it is not widely used. |
| Mark II Method | The acceptance of this method is limited due to the lack of wide scale usage. |
| Full Function Point(FFP) | Not well suited to measuring software that has other software (particularly in other architectural layers) as its primary users. |
| Feature Point Analysis | Still considered as an experimental method. |

*D.* ***Why FFP is most suitable for Real time Software?***

- Works for all types of software (scientific, business apps, web portals, embedded systems, etc.)[4]
- Works for all types of projects (new development, enhancements, maintenance, etc.)
- Language independent and technology independent
- Produces statistically significant results
- Can be applied early in the development life cycle

*E.* ***Field Test of PDC***

We were focusing on the evaluation of the chosen functional measures of software, Function Point Analysis (FPA) and Full Function Points (FFP) to Real time software. We applied both these methods to a project broadband/mobile data system based on the Japanese standard for mobile data communication (PDC). It lasted for about one year and involved about 50 persons.

We divided the application into modules, each one of these modules was logically coherent and related to the other modules, which meant that they together formed a working unit. In FPA and FFP terms we looked at each of these modules as separate applications. A wide range of module sizes was represented in this application. We wanted this distribution of size, since comparisons between the modules were then easier to make.

Our hypothesis before the tests began was that the FFP would be more useful than FPA for real time system. If we could falsify this hypothesis, then FPA would be better for PDC.

*E.* ***The result***

As mentioned we separated the project of study in modules, to all twelve pieces. For reasons of simplicity we will call them A, B, C, D, E, F, G, H, I, J, K and L. In the Interwork Description (IWD) document module A, B, C and D were treated as one unit. Thus, during the first phase of the tests, we did not count A, B, C and D as separate modules, since that was not possible. However, this does not mean that the sum of FPA and FFP count for these modules in the second round of the counting is comparable to the figure we received after the first phase for A, B, C and D as a unit. This is due to the fact that when applying FPA and FFP some parts of these modules are counted several times, above all the communication internally and externally with other modules.

The results that came out of testing the methods on this project are summarized in Table 1. Some important conclusions can be drawn when looking at this table. First of all the modules H, I, J, and perhaps also modules A and K could be regarded as I/O-heavy modules, i.e. modules with a large degree of communication with other parts of the system, but not so much algorithmic complexity. This can be concluded by looking at the quota between the figures for FPA and FFP. If this quota is relatively high the FPA method has a greater impact, and thus it is probable that these modules contains much input and output, processes that are counted high with the FPA method. The modules B, C and D, on the other hand, could be regarded as modules with many sub-processes and complex algorithms, since the FFP method has a greater impact relatively to FPA. Finally, the figures in the two columns for the second round of counting are higher than in the columns representing the first round.
This means that when we increase the level of detail, regarding the documents and code analyzed, we usually find more functionality to take into account. The exceptions are modules H and I. The reason for this is that they are only concerned with input and output, i.e. the functionality is included in the IWD document.

TABLE I. RESULTS OF COUNTING FPA AND FFP

| | Counting based on IWD's | | Counting based on IS's and source code | |
|---|---|---|---|---|
| Module | FPA | FFP | FPA | FFP |
| A+B+C+D | 231 | 42 | | |
| A | | | 455 | 116 |
| B | | | 296 | 134.4 |
| C | | | 276 | 183 |
| D | | | 268 | 81 |
| E | 200 | 35.6 | 283 | 89.6 |
| F | 184 | 54.2 | 252 | 79.2 |
| G | 153 | 42.2 | 201 | 62.4 |
| H | 117 | 22 | 121 | 17 |
| I | 91 | 21 | 86 | 16 |
| - | 89 | 18 | 170 | 38.4 |
| . | 65 | 13 | 137 | 32.8 |
| L | 65 | 13 | 113 | 37.2 |

### *F. validation of result*

Our main concern was to validate the results we got from our tests from an objective viewpoint. However, a fully objective view of this specific project at PDC was hard to acquire. The approach we chose was therefore to compare our results with the jointed opinion of the system developers involved in the project. When we had concluded our tests, we asked three of the system developers involved in the project to place the modules in order of precedence. We explained that we wanted them to order the modules according to size and complexity, since FPA and FFP combine these factors. The results of our tests were not shown to them until their order of precedence was done. Naturally their knowledge of the modules varied. Some of the modules they had developed themselves, others were examined by them, but there were always one or a few modules that they had very little knowledge of. Thus, we compiled the opinions from these three persons. In this compilation, when a person had greater knowledge of a module than the others, his opinion took precedence over the others regarding this specific module. However, the opinions were very similar, especially regarding which modules that was the most and the least complex. The order of precedence that became the result of the "opinion compilation" can be seen in Fig. 2.

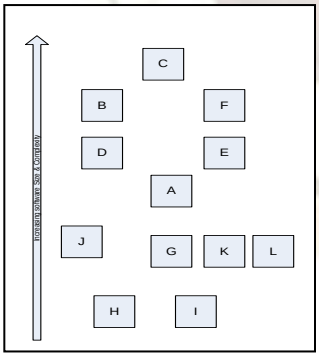


Figure 2. Module complexities according to the system developers

The first thing to notice is that we can distinguish three groups of modules. The most complex group (C, B, F, D, E and A) consists of modules with many lines of code and many algorithms. The middle group (J, G, K and L) is made up of modules that contain a certain amount of input and output, but also fragments with algorithms and complex functionality. The last group (H and I) of modules are units with a pure I/O- functionality.

If we compare our tests of the FPA and FFP methods on these modules with the opinion of the system developers, we find that they are rather congruent. The column in Table:1 that best agrees with Figure:2 is without doubt, the counting of FFP on IS's and source code (4th column). The order of precedence there is C, B, A, E, D, F, G, J, L, K, H and I. The explanation for the exceptions (A and G is counted higher with FFP than according to the system developers, and F is counted lower) can be found in the system developer's perception of the problem. We suspect that they interpreted the order of precedence only as a software complexity issue and disregarded the size factor. If we look at A and G we also find that they are made up of relatively many lines of code, and F is rather complex but is also a low-volume module.

Thus, the outcome of this simple but rather straightforward validation of our results speaks in favor of a detailed counting with the FFP method. As we predicted, the FPA method fails to take into account the complexity factors that are inherent in real-time systems, such as the number of algorithms and the number of sub-processes. Moreover, we need a detailed and comprehensive documentation in order to make accurate use of the FPA and FFP methods.

Finally, the proposed model is justifiable, this will able to choose a measure of algorithmic complexity and a functional size metric , field test were performed for real time software using two main candidate : FPA and FFP. These test showed that FFP is the best adapted method for real time software. Thus,

Productivity=FFP/man time

## III. QUALITY

### *A. Structural Complexity*

According to our model productivity also reflect quality so next we will measure the quality with the help of structural measure of software complexity. A principal objective of software engineering is to improve the quality of software products .Quality attributes in measurable form Structural Measures of Software Complexity; we considered a range of internal attributes believed to influence quality in some way. Many practitioners and researchers measure and analyze internal attributes because they may be predictors of external attributes. There are two major advantages to doing so. First, the internal attributes are often available for measurement early in the life cycle, whereas external attributes are measurable only when the product is complete (or nearly so). Second, internal attributes are often easier to measure than external ones.

Quality=Defect/Function Point

Defect density as a measure of reliability we can consider the defects to be of two types, the known defects that have been discovered through testing, inspection, and other techniques, and the latent defects that may be present in the system but of which we are as yet unaware (Olsson, 1996). Then we can define the defect density as:

Defect Density = Number of known defects / Product

### *B. Proposed method of Quality*

Size According to our hypothesis that if we measure defect by using best error-prone metric then we can increase the quality of software. And after theoretical and practical studies we found that object point metric suit which is based on object oriented analysis(OPA) is best error prone metric. The three different quality metric suites are [10]

- Chidamber and Kemerer (CK) Metrics produces statistical models that is effective in detecting error-prone in classes.
- Robert C. Martin's Metric Suite is good in predicting the faults in terms of the packages.
- McCabe's Metric Suite are good fault-proneness predictors in methods

So The McCabe's Cyclometric Number of OPA is best metric to measures errors in the code.

### *C. Different Categories of structure Metric*

| Structural Metrics | Critics |
|---|---|
| McCabe's Cyclometric Number | It is only psychological complexity not a computational complexity. |
| Halstead's Measures | Lack of Standard definition so it is not applicable for larger systems, error-proneness has also been made on small-scale systems. |
| Henery and Kufra's measures | Not mathematical correct, according measurement theory, and revision of the formula have been made - proneness of a program. |
| Object point Analysis(OPA) | this measure is only compatible with object oriented approach |

### *D. Why Object point metric is is most suitable for quality measurement of real time system Real time system?*

- OPA founds three types of Fault mainly Object Oriented Faults, Object Management Faults, and Traditional Faults.
- OPA helpful to group OO metrics into System size, Class or method size, Coupling and inheritance, Class or method internals.
- OPA has a metric suite which will help to measure the different level of complexity and give different efficiency in fault prediction.
- The metrics concentrate on the internal object structure of each individual entity that exposes internal complexity and on the interactions among entities that exposes external complexity.
- Computational complexity surrounding the efficiency of an algorithm, utilization of machine resources in addition to intellectual complexity issues that affect the capability of a programmer to create, alter, and understand software and the capability of end user to successfully utilize the software are measured by metrics [3].
- Most reliable testing and yet minimize redundant testing effort.
- if classes are expected to be the most fault-prone then defect detection activities will help to remove these fault before the software is released. So it predicted from Fault early in the life cycle of software development.
- We can also predict the labor required to build the software. So OPA metric will increase the performance by implemented with less effort and lower cost.

## IV. PERFORMENCE

The last section of the paper is devoted to a combined measure of project performance, where the two dimensions of project productivity and software quality is connected to get an overall picture of the project (fig:3) and what it has produced. The measure builds on a technique called Data Envelopment Analysis (DEA), and can be expressed both graphically and in an equation.

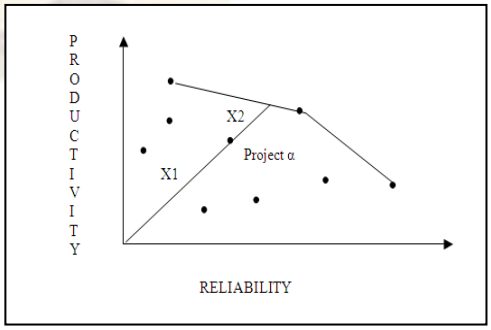


Figure 3. Graphical presentation of the performance measure

Performance of project α = x1 / (x1 + x2)

## V. CONCLUSION

The goal of this proposal is to improved performance by choosing the best Productivity and Quality metrics so we have proposed that FFP and McCabe's Cyclometric Number will improve the performance of real time system

## References

[1] Abran, A., Desharnais, J.-M., Maya, M., St-Pierre, D., & Bourque, P. (1998). Design of Functional Size Measurement for Real-Time Software. Montréal, Université du Québec à Montréal [www document]..

URL http://www.lrgl.uqam.ca/publications/pdf/407.pdf

[2] Bohem, R. (1997). Function Point FAQ. Metuchen,USA, Software Composition Technologies, Inc URL http://ourworld.compuserve.com/homepage/softcom/

[3] Desharnais, J.-M. & Morris, P. (1996). Validation Process in Software Engineering: an Example with Function Points. In Forum on Software Engineering Standards (SES'96), Montreal [www document]..

URL http://www.lrgl.uqam.ca/publications/pdf/104.pdf

[4] Introduction to Function Point Analysis (1998). GIFPA, Issue 2, summer [www document]. URL http://www.gifpa.co.uk/news/News2Web.pdf

[5] International Standards Organisation (ISO) (1991). Information Technology ± Software Product Evaluation ± Quality Characteristics and Guidelines for their Use (ISO/IEC IS 9126). Geneve: ISO/IEC.

[6] Software Metrics - why bother?.(1998). GIFPA, Issue 1, spring

URL http://www.gifpa.co.uk/news/Issue1_ed2.pdf

[7] St-Pierre, D., Maya, M., Abran, A., Desharnais, J.-M. & Oligny, S. (1997a). Full Function Points: Counting Practices Manual. Montréal, Université du Québec à Montréal [www document].

URL http://www.lrgl.uqam.ca/publications/pdf/267.pdf

[8] St-Pierre, D., Maya, M., Abran, A., Desharnais, J.-M. & Oligny, S. (1997b). Measuring the functional size of real-time software. Montréal, Université du Québec à Montréal [www document].

URL http://www.lrgl.uqam.ca/publications/pdf/330.pdf

[9] Fenton, N. E. & Pfleeger, S. L. (1996). Software Metrics - A Rigorous & Practical Approach. London: International Thomson Computer Press.

[10] Functional Size Measurement for Real-Time Software. Montréal, Université du Québec à J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[11] An Empirical Validation of Software Quality Metric Suites on Open Source Software for Fault-Proneness Prediction in Object Oriented System I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.