Function Point Analysis

Joe O Flaherty

T00155775

Contents

[Background 1](#_Toc403054664)

[Detail of FPA based on IFPUG rules 3](#_Toc403054665)

[Calculating Function Points 3](#_Toc403054666)

[Using Function Point Values 7](#_Toc403054667)

[KLOC Comparison 9](#_Toc403054668)

[Conclusions 13](#_Toc403054669)

[Useful Learning Links 13](#_Toc403054670)

[References 13](#_Toc403054671)

# Background

Function Point Analysis (FPA) was developed by Allan Albrecht in his time with IBM over a period of five years and first reported on in October 1979. (Behrens, 1983)The method was developed by IBM because of “serious mathematical and economic problems associated with the older ‘lines of code’ metric or LOC” (Jones, 2013). Jones argues that LOC penalizes high-level programming languages and cannot be used to evaluate non coding activities. Albrecht initially reported on 22 projects, an improvement in productivity of approximately three to one was achieved. (Behrens, 1983) The International Function Point Users Group (IFPUG) was established in 1984 to standardize FPA, promote its use and to clarify its rules. (Abran & Robillard, 1996)The FPA method administered by IFPUG has become a recognised standard and is governed by ISO/IEC 20926:2009. (ISO/IEC, 2009). The IFPUG administered metric can be seen as a first generation FPA method and has been joined as a standard by the second generation COSMIC Function Size Measurement (FSM) method. (Santillo, 2012)

According to Abran and Robillard FPA provides standardized methods for measuring the functionality of a software system. Measurements are taken from the perspective of a user/customer, specifically what is requested and what is actually delivered. The IFPUG summarize Function Point Analysis and the function point technique as *“the function point technique provides an objective, comparative measure that assists in the evaluation, planning, management and control of software production”* (IFPUG, 2014)Furthermore Behrens explains that Albrecht’s initial objective was development of *“a relative measure of function value delivered to the user”*, independent of any technology or methodology. Albrecht is quoted describing his hopes for what FPA would deliver as, *“FPA aims then, to provide a consistent measure of system size that is; independent of the development technology, is simple to apply, can be estimated (reasonably well) from the requirements specification and is meaningful to the end user.”* (Vickers, 2003)Interestingly much of the literature reviewed mentions that the focus of FPA is the user/customer and what value is delivered to them, *“the FPA technique quantifies the functions contained within software in terms that are meaningful to the software users”*. (IFPUG, 2014) Research also emphasises that FPA is applicable to the entire project lifecycle and an estimate of ultimate size can be generated once requirements have been gathered, *“the measure relates directly to the business requirements that the software is intended to address. It can therefore be readily applied across a wide range of development environments and throughout the life of a development project, from early requirements definition to full operational use.”* (IFPUG, 2014)

Estimating the effort and time-frame required to develop and maintain a software system is one of the most if not the most critical activity in software project management. (Azath & Wahidabanu, 2012) Most models attempt to firstly generate a measure of effort, which is subsequently used to establish both project duration and cost. Software size is the most important factor that influences the overall cost, one of two metrics is usually used to capture this size, source lines of code (SLOC) or Function Points (FPs). SLOC measures the physical size of software but is unavailable until coding has begun thus is not suitable as an estimation tool. FPA in contrast is available from the requirements phase and incorporates historical information that provides a more accurate view of software size. (Azath & Wahidabanu, 2012) Function point metrics according to Jones *“are the most accurate and effective metrics yet developed for software sizing and also for studying software productivity, quality, costs, risks, and economic value.” (Jones, 2013)*

The main software cost drivers according to Capers Jones are:

1. Finding and fixing defects.

2. Producing paper documents.

3. Coding or programming.

4. Meetings and communications.

5. Management.

6. Dealing with software requirements changes.

(Jones, 2013)

By 2013 function point analysis had become the predominant method for estimating and managing costs. Jones mentions that by 2013 approximately 50,000 software projects had been measured using IFPUG methods, 5,000 of these are available from the International Software Benchmark Standards Group (ISBSG) at [www.ISBSG.org](http://www.ISBSG.org).

# Detail of FPA based on IFPUG rules

## Calculating Function Points

Function Points (FPs) can be used to “estimate the relative size and complexity of software in the early stages of development” i.e. the analysis and design phase. (Angel, et al., 2012)Therefore you have to know the functionality your system proposes to provide before you start using Function point analysis. (Koirala, 2004)Size is determined by identification of the functions of a system from the point of view of a user/customer, *“One function point (FP) is one end-user requested business function”.* (Grupe & Clevenger, 1991) The components are rated simple, average or complex, scored and totalled, the result described as unadjusted FPs (UFPs). (Azath & Wahidabanu, 2012) This process is quite complex and accurate FP estimation requires a high level of expertise as inaccuracy can be catastrophic, *“Miscalculating data items, report totals, etc., can unknowingly undermine the accuracy of the function point analysis count, and thus, render it useless.”* (Southard, 2000)Resultantly experienced FB estimators and experts can command high salaries. Koirala points out that software costing is “still a very grey area” but FPA can help to calculate a close estimate. A considerable amount of the execution of FPA and counting of function points is common sense and improves with experience, “experience is the only thing that comes to the rescue in software costing and evaluation.” (Koirala, 2004)

The formula to calculate function points is:

**FP= UFP\*CAF**

CAF = Complexity adjustment factor.

UFP = Unadjusted function points.

UFPs are calculated from a count of five functional components.

**External Inputs (EI):** The process of data entering the system from outside. This data can be user input or come from an external application and may be used to maintain one or more internal logical files.

**External Outputs (EO):** The process by which data from inside a system to outside. Internal logical files may also be updated by an EO.

**External Inquiry (EQ):**  A process resulting in the retrieval of from one or more internal logicalfiles and/or external interface files.

**Internal Logical Files (ILF’s):** Groups of logically related data that exist inside the application boundary and are maintained via inputs external to the system.

**External Interface Files (EIF’s):** Group of logically related data used solely for reference purposes and exist entirely outside the application boundary; EIF’s are the ILF’s of other applications.

(Angel, et al., 2012)

The first three components are further referred to as Transactional Function Types (TFT), the last two as Data Function Types (DFT). Ratings are applied to TFT’s based on how many Data Element Types (DET) and File Types Referenced (FTR) they have associations with. DFT’s are rated according to Record Element Types (RET), and Data Element Types (DET).DET’s are unique non-repeating fields of dynamic information; a dynamic field can be read from a file or created by DET’s within an FTR. RET are groupings of data within an ILF or an EIF. FTR’s are files referenced by transactions that must be an ILF or EIF. (Kurmanadham, n.d.)

Importantly it must be noted that prior to calculation of FP’s counting boundaries must be established. Boundaries are the borders between the application or project and external applications or user domains, boundaries specify what is included in the function count.



(HeydarNoori, n.d.)

Once categorised as one of the 5 major components a rating of Low, Average or High is applied which in turn receives a weight based on the category of function. Inputs and inquiries receive the lowest rating, interactions with files or databases the highest. (Grupe & Clevenger, 1991) These ratings are termed degrees of influence and are shown in the following scale.

Degrees of influence (DI) scale:

Not present/no influence = 0

Insignificant = 1

Moderate = 2

Average = 3

Significant = 4

Strong Influence Throughout = 5

(Vickers, 2003)

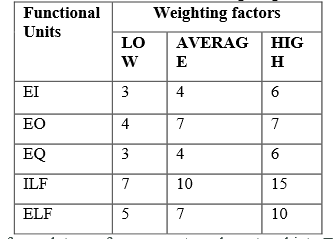
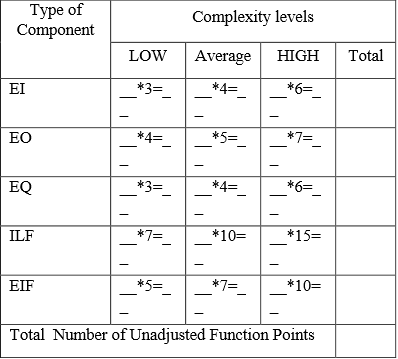


Table showing the weights applied to each function. (Yongchang, et al., 2011)

The counts for each type of component are entered into a separate table and multiplied by the appropriate numerical rating, this determines the rated value. These totals are totalled per row, giving a combined value for each type of component then summed to give the Total Number of Unadjusted Function Points.

 (Vickers, 2003)

**CAF - Complexity Adjustment Factor calculation**

The CAF, sometimes referred to as the Value Adjustment Factor (VAF) is based on 14 general system characteristics (GSC's) that “rate the general functionality of the application being counted.” (IFPUG, 2010) Each GSC has an associated description that aids in determining its degree of influence (0 – 5).

**General System Characteristics as listed by the IFPUG Function Point Counting Practices Manual (CPM) v4.3.1**

**1. Data communications:** How many ways can data be entered and the system interacted with?

**2. Distributed data Processing**: How are distributed features of the system handled?

**3. Performance** What performance is specified by the user e.g. response times?

**4. Heavily used configuration:** Is the platform where the application will run heavily used?

**5. Transaction rate**: What is the frequency of transactions?

**6. On-Line data entry**: How much data originates from an online source?

**7. End-user efficiency**: Has the system been designed for end-user efficiency?

**8. On-Line update:** How many ILF’s are updated by online entries?

**9. Complex processing:** Is there complex logical or mathematical processing?

**10. Reusability**: Is the system designed to serve multiple users?

**11. Installation ease**: Is installation complex?

**12. Operational ease**: Does the system require technical proficiency of the user or can operations be automated e.g. Back-ups?

**13. Multiple sites:** Is the application designed for installation at more than 1 location or for more than 1 organisation?

**14. Facilitate change:** Has the system been designed to facilitate change?

(Angel, et al., 2012) (IFPUG, 2010)

CAF = 0.65+0.01(∑Fi)

Note: i = 1 to 14 representing each GSC.

Note: Fi = Total of DOI for all 14 GSC’s

**Example:**

Function points have been counted for each function.

Number of user inputs (EI) = 100

Number of outputs (EO) = 80

Number of Inquiries (EQ) = 70

Number of ILF = 12

Number of EIF = 8

Complexity adjustment factors and weightings are assumed to be average.

**Calculate Unadjusted Function Points:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Weight Factors | | | |
|  | LOW | AVERAGE | HIGH | TOTAL |
| EI | 0\*3 = 0 | 100 \* 4 =400 | 0\*6 = 0 | 400 |
| EO | 0\*4 = 0 | 80 \* 5 =400 | 0\*7 = 0 | 400 |
| EQ | 0\*3 =0 | 70 \* 4 = 280 | 0\*6 = 0 | 280 |
| ILF | 0\*7 = 0 | 12\*10 = 120 | 0\*15 = 0 | 120 |
| EIF | 0\*5 = 0 | 8\*7 = 56 | 0\*10 = 0 | 56 |
| Total Unadjusted Function Points | | | | 1256 |

**Calculate CAF:**

Assume all CAF are average

CAF = 0.65+0.01(∑Fi)

i= 1 to 14 = (0.65+0.01\* (14\*3)

CAF for average = 0.65 + 0.42 = 1.07

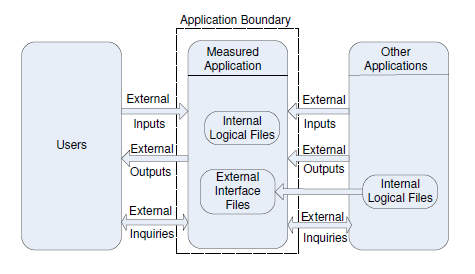
**Calculate Function Points:**

FP = UFP \* CAF

1256 \* 1.07 = 1343.92 rounded to 1344 FP

(Angel, et al., 2012)

The result of these computations (1344 FPs) it an accurate measure of the functional size of the system, *”Although the FP metric does not correspond to any actual physical attribute of a software system it is a relative measure for comparing projects, measuring productivity, and estimating development effort and time needed for a project*”. (Azath & Wahidabanu, 2012) Because FPA is often used as an estimate the introduction of a ‘buffer’ as a practical safeguard for under-estimation or a customer requesting extra functionality or features is suggested. (Koirala, 2004)



(Nassif, et al., 2010)

High level view of the FP model which is useful as a visual aid in understanding the concepts of the model.

## Using Function Point Values

The power of function points is that comparisons are possible from project to project and from application to application, comparisons are independent of language or technology. Similarly an estimation of development time, developer productivity, or quality improvement by reducing defects in software can be performed via function points regardless of language or technology factors using the following rates. (Southard, 2000)

Delivery Rate - function point delivered per hour (FP/Hours)

Support Rate – support effort per function point (Hours/FP)

Cost Rate - cost per function point (€/FP) shown here in Euro per FP but commonly the dollar is used regardless of geographical location.

Defect Density – defects present per function point (Defect/FP).

During the requirements gathering phase of a project a FP analyst or analysis team will estimate or count the functional requirements of the system in conjunction with the customer. Prior to the project proceeding to the design phase and subsequently development the customer must accept or reject the results of the function point analysis. Communication of the amount of time and cost to the customer that this functionality entails is essential as expectations are created and must be managed from this point. Accurate FP analysis facilitates this communication because, *“Function Points are easily understood by a non-technical user”*. (Kurmanadham, n.d.)The most vital measures derived from FPA from a customer/user point of view are project cost and project development time. (Southard, 2000)

Situations can arise where function point count is an issue prior to any effort being applied to a software system. Using a simple scenario of 1 software developer with a FP delivery rate 0.20 FP/Hour these situations are:

**Project Viability:** Analysis has established that the proposed project has a size of 100 FPs implying that the project will take 500 hours to complete. The customer can then reject the project, request an increase in delivery rate per developer to improve the delivery rate (unlikely), or remove some of the proposed functionality thus reducing the FP count.

**Managing Project Change (Development phase)**: The customer wishes to add functionality to the system. FP’s must be calculated for these proposed changes as well as any increase this entails to the original functionality. These changes must then be communicated to the customer plus the extra cost incurred, in this example the customer requests changes that amount to an extra 10 FPs thus increasing development time by 50 hours.

**Supporting Applications:** How much support time is required for an application. This is calculated by the analyst based on experience and the history of support time on previous projects. The developer in this case supports an existing project at the rate of .20(Hours/FP) per month. 100 function points per month to support means 20 hours per month support time by the developer.

**Defect Density:** Defect density per FP can be used as a means of monitoring application quality and historical quality delivery i.e. improvement or deterioration. The problem with this is that a developer or company must have previous products to make a comparison against. The advantage is that a comparison using FP count can be made between products regardless of language, environment or technology. Using a simple example is 3 projects that each has 100 defects and FP counts of 1000, 10000, 100000 respectively, a trivial calculation can establish the defect density.

Project A: 1000 /100 = Defect Density of 0.1

Project B: 10000/100 = Defect Density of 0.01

Project C: 100000/100 = Defect Density of 0.001

Clearly from this calculation Project C is the highest quality product followed by Project B with Project A significantly lower. The power of this analysis is that if Project A was developed first followed by B and C then quality can be seen to have increased over time. However if C was first followed by B and A then there is a problem which needs to be addressed as quality is degrading over time.

Cost to the customer can also be calculated based on FP and how much a developer or company earns per function point. This is a relatively simple calculation to perform for a single developer project. Based on the previous example of a 100 FP project with a developer rate of .20/FP per hour.

Developers rate = €50 per hour.

Cost to complete 1FP = €50 \* 5(Hour to complete 1FP) = €250 per FP

Cost of project = No of Function Points \* Cost per Function Point

For this example 100FP \* €250 = €25000.

This simple calculation can be expanded upon for a larger multi-developer project where multiple factors affect cost per FP. Based on development costs alone as an example of a larger project the following example applies:

10 developers @ €250 per FP.

5000FP project and each developer completes 500FP = €125000 per developer.

€125000 \* 10 developers = €1250000 solely on development costs.

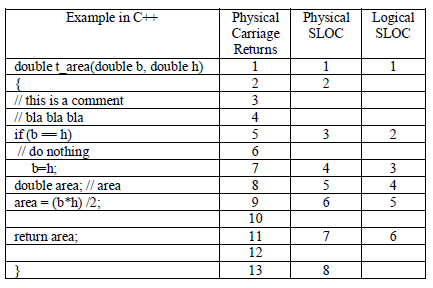
# KLOC Comparison

According to Bill Gates, "Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs.”

Lines of Code (LOC) is a traditional measure used to evaluate the size of a software application, LOC is commonly expanded and can be referred to as Source(S) Lines of Code (SLOC) and Thousand(K) of Lines of Code(KLOC). LOC in one form or another has “been used to measure software size since the inception of software.” (Kurmanadham, n.d.)

LOC in any of its incarnations is conceptually a very simple process which at a basic level is simply counting lines of source code. The major problem with using lines of code as size estimation is deciding what will actually be counted, as Einstein said *"Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted.”* There are three potential solutions to this problem, count every physical line where a physical line of code is not blank and not a comment. (Kurmanadham, n.d.) Alternatively attempt to count logical statements which is much more difficult as the definition of a statement varies from language to language. (Nguyen, et al., n.d.) Adding to the difficulty of what will be counted is the concept of counting physical carriage returns which is the inclusion of every line including comments and white spaces.

The counts from each of these variations can vary considerably even in a very small example:

 (Nassif, et al., 2010)

Essentially the difference between KLOC/LOC and FP’s as system sizing metrics is that LOC is calculated internally whereas FP’s are an external estimate/count of value to an end-user or customer. Generally the customer has no interest in the internal operations of a system; he/she only cares about the functionality visible to them.

FP Measures

Functionality Visible to User

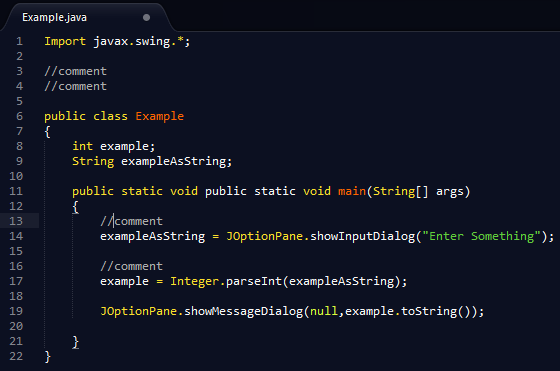
System Software

LOC/KLOC Measures

KLOC does not measure the functional complexity of a system as opposed to FP’s which can clearly be used to demonstrate how complex a given system will be from very early in the SDLC. KLOC as a metric is not comparable across different programming languages, neither does it account for different coding styles or standards nor developer experience, FP’s are independent of all these factors. KLOC statistics are not available early enough in the requirements phase for estimating purposes,”LOC can only be counted during or at end of coding phase”. (Houston, 2003) – FP’s can be calculated from requirements phase.

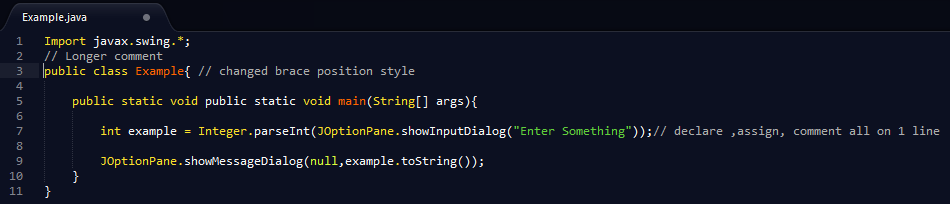
KLOC is a very low effort means of estimating or sizing and as lines of code are physical entities can be easily automated. Function point analysis in contrast is high effort, high cost, cannot be easily automated and is a process that requires highly skilled and experienced staff.

A simple example of the difference coding styles and developer efficiency can make to LOC counts.



Count using physical non-blank, non-comment lines = 12.

Count using carriage returns method = 22



Re-written code that provides identical functionality.

Count using physical non-blank, non-comment lines = 7

Count using carriage returns method = 11

The simple example above portrays a fundamental weakness of LOC/KLOC in comparison to FP’s as a means of estimating the effort an individual developer is capable of exerting. FP’s can be used to measure this effort by using historical data to work out what percentage of a FP a developer can deliver per hour. Using LOC the developer in the first example above would be seen as the more productive as they have produced more LOC even though the second developer is obviously more efficient and proficient, “*skilled developers may be able to develop the same functionality with far less code, so one program with less LOC may exhibit more functionality than another similar program. In particular, LOC is a poor productivity measure of individuals, since a developer can develop only a few lines and still be more productive than a developer creating more lines of code.”* (Kurmanadham, n.d.)

Significant amounts of code are now automatically generated by various IDEs, Microsoft’s Visual Studio for example. LOC/KLOC cannot distinguish this automatically generated code, causing large variations in productivity and other metrics because code is counted that no developer has written.Finally in the world of OOP (Object Oriented Programming) and the philosophy of code reuse LOC makes very little sense, there is nothing to be gained in counting the same code repeatedly. Kurmandaham argues that, “*object is a true representation of data and functionality and so is a Function Point, FPA remains more relevant for Object-Oriented software development.”*

# Conclusions

Firstly something that is not entirely clear from the literature reviewed, there is a subtle but important difference in the way FP calculation is described, there is FP ‘counting’ and FP ‘estimation’. Counting means exactly that counting the Functional Points of a system and is a very intensive process. Estimating is much more lightweight and requires much less effort and delivers an approximation often based on experience of the functional size of a similar system. Distinguishing these differences should be extremely important to a company and functional point ‘counting’ can only accurately take place if a standard process has been followed e.g. IFPUG.

While initially quite a complex topic once adequate research is undertaken the process of calculating function points at their most basic level is relatively straight-forward and logical. Subsequently many of the examples and explanations in this report are intended to be as basic as possible and are easily understandable, once the basics are mastered more complex aspects of FPA can be investigated with more confidence. The number of clearly defined rules and standards set out a clear process to follow regardless of which implementation of FPA you are following. The ability to utilise historical information that FPA provides is a major strength of the system, as it enables companies to adjust their processes and systems based on concrete metrics, leading to higher productivity and higher quality.

Applied correctly FPA is a very powerful tool and technique that gives a clear picture of a project or system at a very early stage and enhances communication with customers and within teams. Conversely applied incorrectly FPA is very powerful in a negative way. Which leads to the main conclusion, that FPA is as good, accurate and useful as the quality and expertise of both the staff performing the analysis and any staff involved in requirements gathering and verification.

# Useful Learning Links

<https://www.youtube.com/watch?v=X0WLX8iAFb0>

<https://www.youtube.com/user/functionpoints/videos>

<http://www.codeproject.com/Articles/8151/Using-function-point-to-quote-a-software>

<http://www.softwaremetrics.com/freemanual.htm>

<http://www.washingtoniceaa.com/Presentations/34_Function%20Point%20Analysis.pdf>

# References

Abran, A. & Robillard, P. N., 1996. Function Points Analysis: An Empirical Study of Its Measurement Processes. *IEEE Transactions on Software Engineering,* 22(12), pp. 895 - 910.

Albrecht, A. J. & Gafeney Jr., J. E., 1983. Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering,* 9(6), pp. 639 - 648.

Angel, S. T., Rodrigues, P. D., Dhas, M. J. T. & Samy, S. S., 2012. *Limitations of Function Point Analysis in eLearning System Estimation.* s.l., International Journal of Computational Engineering Research, National Conference on Architecture, Software System and Green Computing (NCASG).

Azath, H. & Wahidabanu, R., 2012. Efficient effort estimation system viz. function points and quality assurance coverage.. *IET Software,* 6(4), pp. 335 - 341.

Behrens, C. A., 1983. Measuring the Productivity of Computer Systems Development Activities with Function Points. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,,* SE 9(6), pp. 648 - 652.

Grupe, F. H. & Clevenger, D. F., 1991. Using Function Point Analysis as a Software Development Tool. *Journal of Systems Management,* pp. 23 - 26.

HeydarNoori, A., n.d. *www.cs.uwaterloo.ca.* [Online]   
Available at: https://cs.uwaterloo.ca/~apidduck/CS846/Seminars/abbas.pdf  
[Accessed 4 November 2014].

Houston, T. K., 2003. *Evolving Standards in Function Points/Lines of Code Ratios.* Los Angeles, Center for Software Engineering, University of Southern California.

IFPUG, 2010. *IFPUG Function Point Counting Practices Manual (CPM) v4.3.1.* Princeton Junction, New Jersey: IFPUG.

IFPUG, 2014. *About Function Point Analysis.* [Online]   
Available at: http://www.ifpug.org/about-ifpug/about-function-point-analysis/  
[Accessed 27 October 2014].

ISO/IEC, 2009. *Software and systems engineering — Software measurement — IFPUG functional size measurement method 2009.* s.l.:ISO/IEC.

Jones, C., 2013. *Function Points as a Universal Software Metric,* s.l.: Namcook Analytics LLC.

Koirala, S., 2004. *www.codeproject.com.* [Online]   
Available at: http://www.codeproject.com/Articles/8151/Using-function-point-to-quote-a-software  
[Accessed 3 November 2014].

Kurmanadham, V., n.d. *Function Points or Lines of Code? – An Insight,* s.l.: Global Microsoft Business Unit, Wipro Technologies.

Nassif, A. B., Capretz, L. F. & Ho, D., 2010. *Software Estimation in the Early Stages of the Software Life Cycle.* Nanded, Maharashtra, India,, International Conference on Emerging Trends in Computer Science, Communications and Information Technology.

Nguyen, V., Deeds-Ru, S., Tan, T. & Barry, B., n.d. *A SLOC Counting Standard,* Los Angeles: Center for Systems and Software Engineering, University of Southern California.

Santillo, L., 2012. *Easy Function Points -- 'Smart' Approximation Technique for the IFPUG and COSMIC Methods.* Assisi, IEEE, pp. 137 - 142.

Southard, R., 2000. *www.umsl.edu.* [Online]   
Available at: http://www.umsl.edu/~sauterv/analysis/function\_point/index.html  
[Accessed 3 November 2014].

Vickers, P. D., 2003. *An Introduction to Function Point Analysis,* Newcastle-upon-Tyne: Northumbria University.

Yongchang, R., Tao, X. & Qiang, Q., 2011. *International Conference on Information Science and Technology.* Nanjing, Jiangsu, China, IEEE.