ATLANTIS PRESS International Journal of Networked and Distributed Computing, Vol. 6, No. 3 (July 2018) 185 - 194

Estimating Procedure for Function Point and Development Cost in South Korea

Byeongdo Kang[1], Jongseok Lee [2]

[1] Department of Computer & Information Engineering, Daegu University,
201, Daegudae-ro, Gyeongsan-si, Gyeongsangbuk-do, 38453, Republic of Korea
E-mail:bdkang@daegu.ac.kr

[2] Department of Computer Engineering, Woosuk University,
66, Daehak-ro, Jincheon-eup,Jincheon-gun, Chungcheongbuk-do, 27841, Republic of Korea
E-mail:jong1007@woosuk.ac.kr

Abstract

This paper introduces the practical guidelines for function point analysis in the Republic of Korea. Korean government recommends that Function Point Analysis should be adopted as a standard method for measuring the functional size of software. Function Point was proposed by an international organization. We will present the procedure of counting Function Point with adjustment factors. The functional size and development cost of software can be estimated from the number of Function Point**.**

Keywords: Software Size, Development Cost Estimation, Function Point Analysis, Korean Guidelines for FP.

1. Introduction

Function Point Analysis (FPA) was proposed by Albrecht to measure the size of data processing systems from the end-users' point of view [1]. Another goal was estimating the development effort. The International Function Point Users Group (IFPUG)[2] was created to maintain the definition of the method and publishes the official Function Point (FP) Counting Practices Manual [3]. FPA is standardized by ISO/IEC 20926:2010.
We introduce Function Point and software development cost in section 2. Then we will show how to apply FPA to Korean software industry and estimating procedure for software development cost in section 3. Korean government provides practical guidelines for FPA under industrial circumstances in Korea. In section 4, we come to the conclusion with summary.

2. Function Point and Software Development Cost

The software development cost can be estimated by multiplying Function Point (FP) by the unit cost of Function Point. The software size can be measured by Function Point.

2.1. Function point

Function Point is an international standard, ISP/IEC 14143 (FSM: Functional Size Measurement) for measuring software size. Function Point is a method of measurement for software size. Function Point represents the size of functions that users require from users' viewpoints and must be delivered to users. Function Point is one of important methods for estimating the cost and resources that are required to develop, maintain, and operate software.
The characteristics of Function Point are as follows:

- It measures the software functional requirements that are provided to users,

- It doesn't measure how to implement software from provider's viewpoints, but what functions users require from consumer's viewpoints,
- It can measure workload before development phase,
- It can be used over all of software life cycle including plan and operation phases,
- It can measure the workload of development and maintenance with paying no regard to development methodologies, physical or technical components, organizations, or implementation technologies.

But, we know from our experience that there are some illogical points in applying Function Point to the following types of project:

- An information project related with contents including homepage design, improvement of accessibility for Web, and movies,
- A R&D oriented software development project,
- A software development project with very high complexity of internal process compared to the functional size known to users,
- Cases such as data tuning and optimization or test work impossible to estimate Function Point,
- A project having budget less than USD50,000 for software development.

Function Point is a metric of measuring the size of software functions from user's viewpoint. Software functions can be classified into data functions and transactional functions according to their logical functionalities from user's viewpoint. Data function has two different types, an internal logical file or an external interface file. Transactional function has three different types, an external input, an external output, or an external inquiry.

2.2. Software development cost

We in South Korea can estimate the software development cost by the following Eq. (1).

Software Development Cost = (FP * the unit cost of FP * adjustment factor) + direct cost + profit. (1)

Software manufacturing cost can be estimated by multiplying Function Point by the unit cost of Function Point. If there are adjustment factors dependent on the characteristics of project, we multiply software manufacturing cost by them. Direct cost is expense for the project. Korean government recommends that the profit should be within the limits of twenty-five percent of software manufacturing cost.

3. Estimating Procedure for Function Point and Software Development Cost

The organization of software industries and Korean government recommend a procedure of FPA under Korean industrial circumstances. The practical guidelines for FPA in Korea are published in [5] and [6].

According to [2][3][4], the estimating procedure for FPA method and software development cost includes the following steps:

- Phase 1. Prearrangement: we define the domain and requirements for the software to be developed.
- Phase 2. Estimating Function Point: we define software functions, and estimate Function Point with reflexing the complexity of functions.
- Phase 3. Estimating software manufacturing cost without adjustment factors: in order to get software manufacturing cost, we multiply the Function Point by the unit cost of Function Point.
- Phase 4. Estimating software manufacturing cost with adjustment factors: we find adjustment factors according to the characteristics of the project. We defined four different types of adjustment factors, software size, programming language, type of application, and the quality and characteristics of specificity.
- Phase 5. Estimating direct cost and profit: we calculate direct cost related with developing software. Project profit is allowed within the limits of twenty-five percent of software manufacturing cost.
- Phase 6. Estimating software development cost: we can take software development cost by figuring out the total sum of software manufacturing cost, direct cost, and profit.

3.1. Prearrangement

In prearrangement phase, we define the domain of business to be developed by reference to project plan. The first work is to collect available documents such as project proposal, requirements documents, entity-relationship diagram, object model, data model, file or table layout, user or system interface design, screen and output layout, function specification, use case, and manuals for the users and operators.

After studying the available documents, we define the requirements for software system to be developed. Requirements are classified into two different types,

functional requirements and non-functional requirements. Functional requirements are the target of Function Point analysis. Non-functional requirements are quality and technical constraints affecting the development productivity. They are not the target of Function Point analysis, but are reflected as adjustment factors in Function Point analysis.
The second work is to define software functional requirements. Functional requirements are decomposed into recognizable components that can be analyzed by Function Point.
Final work in prearrangement phase is to decide how to estimate the software size. We use Function Point method.

3.2. Estimating function point

In order to estimate Function Point, we define the scope of target of Function Point analysis. The scope of FPA is the whole software system or some of subsystems.
After defining the scope of FPA, we establish the boundary of applications to be measured from other applications or external systems. Because Function Point is counted from external user's viewpoint, applications must be divided according to their independent functionalities from user's viewpoint.
Settlement of boundaries of applications is important work for Function Point analysis. Total Function Point of a system is the sum of Function Point for each application that is comprised in the system. So, we count Function Point for each application, and then take total Function Point for the whole system.
Because Function Point analysis is measured from user's viewpoint, it is independent of technologies or methodologies applied to developing the system. Function Point for a system must be same in spite that its running environment may be a client server system or World Wide Web.
Functional user requirements are modeled by five basic function components (BFC). IFPUG BFCs are data functions and transactional functions. Data functions are classified into internal logical files (ILF) and external interface files (EIF). Transactional functions are classified into external inputs (EI), external outputs (EO), and external inquiries (EQ). Each Function Point of the identified BFC is measured, and then the size of the whole software is calculated by the total sum of Function Point of each BFC.

3.2.1. Measuring data functions

Data Function Point is the amount of internal or external data requirements. Data function is classified into two different types, the internal logical file (ILF) and the external interface file (EIF). Data Function Point is dependent on the number of ILF and EIF and their complexities.

The criteria for recognizing data function is as follows:

- Find the logical data or control information that users can recognize within the scope of an application,
- Exclude entities that are not maintained or referenced in the application,
- Group entities dependent on each other into one entity,
- Exclude entities treated as identifier data. Identifier data generally consists of a key and a name,
- Exclude technical entities that don't include attributes that users require.

Function Point analysis for data function is as follows:

- Step 1. Identify internal logical files (ILF) and external interface files (EIF).

From the scope of software functions, we identify internal logical files and external interface files, and then count the number of ILF and EIF.
The internal logical file is logically connected and single data group or control information that users can recognize, and it is maintained in the application.
The external interface file is logically connected and single data group or control information that users can recognize. It is only referenced inside of the application, and is maintained in other application.

- Step 2. Estimate the complexities for each ILF and EIF.

After identifying the number of data element types (DET) and record element types (RET) for each internal file and external interface file, we decide the complexities of ILF and EIF.
If data subgroup doesn't exist inside of ILF or EIF, ILF or EIF itself is a single record element type. Or if data subgroup exists, it can be one of record element types.
Data element type is a unique attribute that users recognize, and it is not duplicated. If more than one application maintains or accesses to the same ILF or EIF, the number of DET for each ILF or EIF is the

number of attributes that each application accesses respectively.
After identifying RET and DET, we estimate the complexity for ILF or EIF. The complexity of ILF and EIF is given in the following table 1 and table 2 respectively. The ILF complexity value of low, average, or high is 7, 10, or 15 respectively.

Table 1. ILF Complexity.

No. of RETs	No. of Data Element Types (DETs)		
	1~19	20-50	51+
1	Low	Low	Average
2 ~ 5	Low	Average	High
6+	Average	High	High

Note: Low=7, Average=10, High=15.

The EIF complexity value of low, average, or high is 5, 7, or 10 respectively.

Table 2. EIF Complexity.

No. of RETs	No. of Data Element Types (DETs)		
	1~19	20-50	51+
1	Low	Low	Average
2 ~ 5	Low	Average	High
6+	Average	High	High

Note: Low=5, Average=7, High=10.

- Step 3. Estimate data Function Point.

The complexity of total ILF can be obtained by the following Eq. (2).

FP(ILF) = Total ILF complexity = (<number of low ILF> * 7) + (<number of average ILF> * 10) + (<number of high ILF> *15). (2)

The complexity of total EIF is calculated by the following Eq. (3).

FP(EIF) = Total EIF complexity = (<number of low EIF> * 5) + (<number of average EIF> * 7) + (<number of high EIF> *10). (3)

From the complexity of total ILF and EIF, we can get data Function Point by the following Eq. (4).

Data Function Point = FP(ILF) + FP(EIF). (4)

When we calculate data Function Point, it is convenient to use the form shown in table 3.

Table 3. Calculation Form for data Function Point.

Function Type	Weight			Total
	low	average	high	
ILF	( ) * 7	( ) * 10	( ) * 15	( )
EIF	( ) * 5	( ) * 7	( ) * 10	( )
Data Function Point				( )

3.2.2. Measuring transactional functions

A transactional function is a unit process to provide users with functions to handle data of interest. A unit process is a smallest activity of functional requirements, and it must be meaning to users, self-complete, and complete transaction to keep business consistent. A transactional function is classified into three different types of BFC: an external input (EI), an external output (EO), and an external inquiry (EQ).
Function Point analysis for transactional functions is as follows:

- Step 1. Identify external input (EI), external output (EO), and external inquiry (EQ).

We identify external input, external output, and external inquiry.
External input is data coming from outside of boundary for the application or a unit process to handle control information. It changes more than one internal logical file or system behavior. The unit process such as enrollment, modification, deletion, establishment, or approval of data changes internal logical files. A change of business logic may change system behavior.
External inquiry is the unit process that finds for data or control information. And then it provides users outside of application with the results.
External output includes additional computational logic in external inquiry. Computational logic handles internal logic files, or changes system behavior.

- Step 2. Estimate the complexities of EI, EO, and EQ.

File type referenced (FTR) is an internal logical file or an external interface file. ILF is read or maintained by transactional function. EIF is read by transactional function.
The following are not data element types:

- Report title, screen or panel id, literals,
- Date and time attributes that application produces,
- Symbols for navigation such as 'previous', 'next', 'first', or 'last',
- Attributes in ILF or EIF that don't cross boundary.

After counting the number of DET and FTR for each EI, EO, and EQ, we estimate the complexities for EI, EO, and EQ.

The complexity of EI, EO, and EQ are shown in table 4, table 5, and table 6 respectively.

Table 4. EI Complexity.

No. of FTRs	No. of Data Element Types (DETs)		
	1~4	5~15	16+
0~1	Low	Low	Average
2	Low	Average	High
3+	Average	High	High

Note: Low=3, Average=4, High=6.

The EI complexity value of low, average, or high is 3, 4, or 6 respectively.

The complexity of total EI is calculated by the following Eq. (5).

FP(EI) = Total EI complexity = (<number of low EI> * 3) + (<number of average EI> * 4) + (<number of high EI> * 6). (5)

Table 5. EO Complexity.

No. of FTRs	No. of Data Element Types (DETs)		
	1~5	6~19	20+
0~1	Low	Low	Average
2~3	Low	Average	High
4+	Average	High	High

Note: Low=4, Average=5, High=7.

The EO complexity value of low, average, or high is 4, 5, or 7 respectively.

The complexity of total EO is calculated by the following Eq. (6).

FP(EO) = Total EO complexity = (<number of low EO> * 4) + (<number of average EO> * 5) + (<number of high EO> * 7). (6)

Table 6. EQ Complexity.

No. of FTRs	No. of Data Element Types (DETs)		
	1~5	6~19	20+
0~1	Low	Low	Average
2~3	Low	Average	High
4+	Average	High	High

Note: Low=3, Average=4, High=6.

The EQ complexity value of low, average, or high is 3, 4, or 6 respectively.
Also, the complexity of total EQ is calculated by the following Eq. (7).

FP(EQ) = Total EQ complexity = (<number of low EQ> * 3) + (<number of average EQ> * 4) + (<number of high EQ> * 6). (7)

- Step 3. Estimate transactional Function Point.

The transactional Function Point is obtained by the following Eq. (8).

Transaction Function Point = FP(EI) + FP(EO) + FP(EQ). (8)

We can use a form shown in table 7 for counting transactional Function Point.

Table 7. Calculation Form for Transactional FP.

Function Type	Weight			Total
	Low	Average	High	
EI	( ) * 3	( ) * 4	( ) * 6	( )
EO	( ) * 4	( ) * 5	( ) * 7	( )
EQ	( ) * 3	( ) * 4	( ) * 6	( )
Transactional Function Point				( )

Table 8 is the form for counting total Function Point.

Table. 8. Calculation Form for FP.

Function Type	Weight			Total
	Low	Average	High	
ILF	( ) * 7	( ) * 10	( ) * 15	( )
EIF	( ) * 5	( ) * 7	( ) * 10	( )
EI	( ) * 3	( ) * 4	( ) * 6	( )
EO	( ) * 4	( ) * 5	( ) * 7	( )
EQ	( ) * 3	( ) * 4	( ) * 6	( )
Total Function Point				( )

3.3. Estimating software manufacturing cost without adjustment factors

The Unadjusted Function Point (UFP) is total sum of Function Point for each type of component. The formula is the following Eq. (9).

$$UFP = FP(EI) + FP(EO) + FP(EQ) + FP(ILF) + FP(EIF). \quad (9)$$

Pure software manufacturing cost (PSMC) without adjustment factors is estimated by the following Eq. (10).

$$PSMC = UFP * (\text{unit cost per FP}). \quad (10)$$

The unit cost (UC) per FP is dependent on the average salary for software engineers, their productivity, and the price level.

Total Function Point can be distributed over the development phases: analysis, design, implementation, and test phase. Table 9 show the distribution of Function Point over the software development phases.

If we want Function Point over part of development phases, the following Eq. (11) gives us the partial Function Point for the part of development phases.

$$PSMC = UFP * (\text{unit cost per FP}) * \sum (\text{FP weight for the phase}). \quad (11)$$

Table 9. FP Weight for each Development Phase.

Phase	Analysis	Design	Implementation	Test	Total
Weight per Phase	0.19	0.24	0.32	0.25	1.00
Cost per Phase	Unit cost * 0.19	Unit cost * 0.24	Unit cost * 0.32	Unit cost * 0.25	Unit cost

3.4. Estimating software manufacturing cost with adjustment factors

Pure software manufacturing cost is on the assumption that the project complexity is normal. But the complexity of the project is dependent on the size of projects, types of application, programming languages, and the quality and characteristics of project.

3.4.1. Size adjustment factor

As the size of a project is bigger, the productivity is lower because the number of developers is growing and the communication among them is more complicated.

We adjust pure software manufacturing cost to the size of project. Size adjustment factor (SAF) is the following Eq. (12).

$$SAF = 0.108 * \log(FP) + 0.2229.$$
$$\text{But, } SAF=0.65 \text{ if } FP < 300. \quad (12)$$

In case that one project includes several applications, total size is applied to the size adjustment factor.

3.4.2. Application type adjustment factor

Although the size of applications is all the same, their productivity is different from each other if application types are different.

An application for business process usually has lower complexity than an application for command control for the military.

Applications can be classified into eight different types: business process, science and technology, multimedia, intelligent information, system, communication control, process control, command control. Table 10 shows the application type adjustment factor (ATAF).

Table 10. Application Type Adjustment Factor.

Application Type	Domain	Adjustment factor
Business Process	Personal affairs, accounts, pay, salary, management.	1.0
Science & Technology	Scientific computation, simulation, spread sheet, statistics, OR, CAE.	1.2
Multimedia	Graphic, image, audio, GIS, education, entertainment.	1.3
Intelligent Information	Natural language processing, AI, expert system.	1.7
System	OS, language process, DBMS, man-machine interface, Windows system, CASE, utilities.	1.7
Communication Control	Communication protocol, emulation, switching software, GPS.	1.9
Process Control	Production management, CAM, CIM, machine control, robot control, real-time embedded software.	2.0
Command Control	Army, police command control software.	2.2

In case that one project includes various types of application, each application type factor is applied with respect to its portion of the project.
ATAF is the following Eq. (13).

ATAF = ∑ [(each application size/project size) * each application type factor] (13)

It is convenient to use the form like table 11 when we calculate ATAF.

Table 11. Calculation Form for ATAF.

Type	ATAF			Value	Note
	Type	Adjustment Factor	Portion		
Application Type	Business process	1.0	( )%		100%
	Science & technology	1.2	( )%		
	Multimedia	1.3	( )%		
	Intelligent information	1.7	( )%		
	System	1.7	( )%		
	Communication control	1.9	( )%		
	Process control	2.0	( )%		
	Command control	2.2	( )%		

3.4.3. Programming language adjustment factor

Software development productivity is dependent on programming languages used in implementation phase. The types of programming languages are five different classes. Table 12 shows the programming language adjustment factor (PLAF).

Table 12. Programming Language Adjustment Factor.

Class	Languages	Adjustment Factor
Class	assembly, machine language, natural language	1.9
Class 2	C, CHILL, C++, JAVA, C#, PROLOG, UNIX Shell Scripts	1.2
Class 3	COBOL, FORTRAN, PL/1, PASCAL, Ada	1.0
Class 4	ABAP4, Delphi, HTML, Power Builder, Program Generator, Query default, SmallTalk, SQL, Visual Basic, Statistical default, XML default, Script default(JSP, ASP, PHP, Flash)	0.8
Class 5	EXCEL, Spreadsheet default, Screen painter default	0.6

PLAF is not applied to analysis and design phase, but is applied to the implementation and test phases only.
In case that various languages are used in the project, PLAF is the following Eq. (14).

PLAF = ∑ [(each application size/project size) * each language type factor] (14)

Table 13. Calculation Form for PLAF.

Type	Class	PLAF	Portion	Value	Note
PL	Class 1	1.9	( )%		Total portion is 100%.
	Class 2	1.2	( )%		
	Class 3	1.0	( )%		
	Class 4	0.8	( )%		
	Class 5	0.6	( )%		

3.4.4. Quality and characteristics adjustment factor

Required quality and characteristics of application affect the productivity of the project. Quality and characteristics adjustment factor (QCAF) is divided into four categories: distributed processing, performance, reliability, and multiple sites.
Table 14 shows QCAF. The level of influence for the quality and system characteristics is 0, 1, or 2.

Table 14. Quality and Characteristics Adjustment Factor.

QCAF		Criteria	Influence
Distributed Processing	Characteristics of data transmission between applications	No explicit requirements.	0
		On-line data transmission by Client/Server or Web-based App.	1
		Dynamic mutual cooperation on multiple servers or processes.	2
Performance	Response time or throughput	No explicit requirements and basic performance.	0
		Response time or throughput is important at all times. Limit of process time for related systems.	1
		Performance analysis is required from design, and performance analysis tools are used in design, development and implementation.	2
Reliability	Degree of influence on failure	No explicit requirements and basic reliability.	0

		Easy recovery and a little bit of loss.	1
		Hard recovery, much financial or lives loss.	2
Multiple sites	Support for different type of hardware and software environments	Requirements for single site. Applications run on the same kind of HW or SW.	0
		Requirements for more than one site. Applications run on similar kind of HW and SW.	1
		Requirements for more than one site. Applications run on different kinds of HW and SW.	2

Total amount of influence is the following Eq. (15).

Total amount of influence = ∑ (each QCAF). (15)

After estimating total amount of influence, we can get the QCAF for the project by the following Eq. (16).

QCAF = 0.025 * (total amount of influence) + 1. (16)

In case that one project includes several applications having different QCAF, project QCAF is the following Eq. (17).

Project QCAF = ∑ [(each application size/project size) * each application's QCAF] (17)

Then, software manufacturing cost (SMC) with adjustment factors is the following equation:.

SMC = PSMC * SAF * ATAF * PLAF * QCAF.

Table 15 is the calculation form for SMC with adjustment factors

3.5. Estimating direct cost and profit

Direct cost is the expense related with software development. Korean government recommends that the profit should not exceed 25% of software manufacturing cost.

Table 15. Calculation Form for SMC.

Total FP	Phase	FP weight per phase	Unit cost per FP	Unit cost Per phase	AF				Cost
					size	type	PL	QC	
( )	Analysis	0.19		( )					( )
	Design	0.24		( )					( )
	Implement	0.32		( )					( )
	Test	0.25		( )					( )
Total(with AF)									( )

3.6. Estimating Development Cost

Software development cost (SDC) is estimated by the following equation:

SDC = SMC + direct cost + profit.

Table 16 shows the form for calculating software development cost.

Table 16. Calculation Form for SDC.

Total FP	Phase	FP weight per phase	Unit cost per FP	Unit cost Per phase	AF				Cost
					size	type	PL	QC	
( )	Analysis	0.19		( )					( )
	Design	0.24		( )					( )
	Implement	0.32		( )					( )
	Test	0.25		( )					( )
Total(SMC)									( )
profit ( )%									( )
Direct cost									( )
Software Development Cost									( )

3.7. General System Characteristics

After the calculation of UFP, we can adjust it by applying the Value Adjustment Factor (VAF). The VAF is calculated with respect to 14 general system characteristics (GSC) for software. The 14 GSCs are 1) data communications, 2) distributed data processing, 3) performance, 4) heavily used configuration, 5) transaction rate, 6) on-line data entry, 7) end-user efficiency, 8) on-line update, 9) complex processing, 10) reusability, 11) installation ease, 12) operational ease, 13) multiple sites, and 14) facility change.

The degree of influence on each GSC is represented by a numerical indicator from 0 to 5. Rating 0 means "Not present, or no influence". Rating 1 means the Incidental influence. Rating 2 means the Moderate influence. Rating 3 means the Average influence. Rating 4 means

the significant influence. Rating 5 means the strong influence throughout. Table 17 represents the 14 GSCs and their brief description.

Table 17. General System Characteristics.

General System Characteristics	Brief Description
Data Communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
Distributed Data Processing	How are distributed data and processing functions handled?
Performance	Did the user require response time or throughput?
Heavily Used Configuration	How heavily used is the current hardware platform where the application will be executed?
Transaction Rate	How frequently are transactions executed daily, weekly, monthly, etc.?
On-Line Data Entry	What percentage of the information is entered online?
End-user Efficiency	Was the application designed for end-user efficiency?
Online Update	How many ILFs are updated by online transaction?
Complex Processing	Does the application have extensive logical or mathematical processing?
Reusability	Was the application developed to meet one or many users' needs?
Installation Ease	How difficult is conversion and installation?
Operational Ease	How effective and/or automated are start-up, back-up, and recovery procedures?
Multiple Sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
Facilitate Change	Was the application specifically designed, developed, and supported to facilitate change?

The formula for getting VAF is as follows:

VAF = (TDI * 0.01) + 0.65, where TDI is the total sum of degree of influence for each GSC. VAF can vary in the range from 0.65 to 1.35.

Finally, we can get the Adjusted Function Point (AFP) by the following formula:

AFP = UFP * VAF.

4. Conclusions

IFPUG FPA is widely used method for estimating the functional size of software. Functional user requirements are modeled by five basic function components. IFPUG basic function components are data functions and transactional functions. Data functions are classified into internal logical files and external interface files. Transactional functions are classified into external inputs, external outputs, and external inquiries. Each Function Point of the identified basic function components is measured, and then the size of the whole software is calculated by the total sum of Function Point of each basic function components.

Korean government recommends public industries to use FP with 4 kinds of adjustment factors such as project size, type of applications, programming languages, and quality and system characteristics reflecting Korean industrial circumstances. We presented the procedure of counting Function Point and estimating development cost. The functional size and development cost of software can be estimated from the number of Function Point.

Acknowledgements

This research was supported by Daegu University Grant in 2017.

References

1. A. J. Albrecht, "Measuring application development productivity," *Proceedings of IBM Application Development Symposium*, Montana, CA, USA, pp.83-92, 1979.
2. IFPUG(International Function Point Users Group), *http://www.ifpug.org/*, 2017.
3. IFPUG, "Function point counting practices manual, release 4.3.1," *International Function Point Users Group*, Westerville, OH, USA, 2010.
4. Marcos de Freitas Junior, Marcelo Fantinato, and Violeta Sun, "Improvements to the Function Point Analysis Method: A Systematic Literature review," *IEEE Transactions on*

Engineering Management, Vol. 62, No. 4, pp.495-506, Nov., 2015.

5. Software Engineering Research Team, "A Practical Guide for Function Point Analysis: based on Software Development Project," Korean Edition, *National Information Promotion Agency*, Korea, pp. 1-13, Nov., 2013.
6. KOSA, *A Guide for Software Development Cost Estimation*, Korean Edition, *http://www.sw-eng.kr*, *National Information Promotion Agency*, Korea, 2016.