International Journal of Internet, Broadcasting and Communication Vol.15 No.2 218-226 (2023)
http://dx.doi.org/10.7236/IJIBC.2023.15.2.218

IJIBC 23-2-25

Cost Estimation and Validation based on Natural Language Requirement Specifications

So Young Moon, R. Young Chul Kim*

Visiting Professor, Department of Software and Communication Engineering, Hongik University, Korea
E-mail: whit2@hongik.ac.kr

Professor, Department of Software and Communication Engineering, Hongik University, Korea
E-mail: bob@hongik.ac.kr

Abstract

In Korea, we still use function point based cost estimations for software size and cost of a project. The current problem is that we make difficultly calculating function points with requirements and also have less accurate. That is, it is difficult for non-experts to analyze requirements and calculate function point values with them, and even experts often derive different function points. In addition, all stakeholders strongly make the validity and accuracy of the function point values of the project before /after the development is completed. There are methods for performing function point analysis using source code [1][2][3][4] and some researchers [5][6][7] attempt empirical verification of function points about the estimated cost. There is no research on automatic cost validation with source code after the final development is completed. In this paper, we propose automatically how to calculate Function Points based on natural language requirements before development and prove FP calculation based on the final source code after development. We expect validation by comparing the function scores calculated by forward engineering and reverse engineering methods.

Keywords*: Function Point, Software Cost Estimation, Software Cost Estimation based Requirement, Software Cost Validation Reverse Engineering, Reverse Engineering, Natural Language Requirement Analysis*

1. Introduction

The existing cost estimations are calculated based on various cost estimation models depending on the ability and experience of experts, and among the cost estimation models. Specially, Allen J. Albrecht's Function Point (function point) of IBM is most often used. Function points are often used to predict like software size, schedule, cost, and effort. However, function point measurements produce different results depending on the abilities of experienced experts and those with less experience [3] [8]. In addition, there is a research report that a 30% difference occurred for the same product in the same organization [9]. Therefore, in this paper, an automatic calculation is attempted to increase the consistency of function point calculation. To this end, we propose

Manuscript Received: April. 10, 2023 / Revised: April. 13, 2023 / Accepted: April. 15, 2023
Corresponding Author: bob@hongik.ac.kr
Tel: +82-44-860-2477, Fax: +82-44-865-2460
Professor, Department of Software and Communication Engineering, Hongik University, Korea

equirements-based "cost estimation" and code-based "cost validation". In previous papers, morphological analysis was performed using Python NLTK[10]. Our proposed method is to define natural language requirements, analyze requirements using Stanford Parser, and calculate function points by storing the information in a database. Through this, we estimate costs based on natural language requirements, and make Code-based "cost validation" which develops a Java-based code analyzer to analyze function points and calculates function points based on reverse engineering through the code analyzer.

This paper is mentioned as follows. Chapter 2 describes Abbott's textual analysis method and Stanford parser as natural language analysis methods. Chapter 3 describes the requirements-based cost estimation method. Chapter 4 describes the reverse engineering-based cost verification method. Finally, in Section 5, conclusions and future research are mentioned.

2. Related Work

2.1 Natural Language Analysis Method

We adapt the Abbott's Textual Analysis technique [13] into requirement engineering, which is a very good way to show developers how to identify candidate classes from use cases, domain and problem descriptions, glossaries, legacy models, and even legacy code. Its textual analysis doesn't show that the developers identify code a particular scenario or use case. This can often result in "classes" that are named after the use case or scenario. We develop or reuse classes or modules of code that underlie individual use cases. It is important to identify nouns and verbs with the textual analysis performed on text created by the users of the system. We can adapt Abbott's Textual Analysis to perform to identify the candidate system components in Table 1.

Table 1. Abbott's Textual Analysis

Part of Speech	Component	Example	Part of Speech	Component	Example
Proper noun	Object	Richard Dué	Stative Verb	Invariance	are owned
Common noun	Class	toy	Modal Verb	Data Semantics	must be
Doing Verb	Method	buy		Precondition	
Being Verb	Classification	is an		PostCondition	
Having Verb	Composition	has an	Adjective	Attribute	unsuitable
Intransitive Verb	Exception Event	depend	Transitive Verb	Method	enter

2.2 Stanford Parser

The Stanford NLP Group makes Natural Language Processing available to everyone. We adapt this approach to identify morphology analysis (such as nouns and verbs) of requirement sentence, which solve major computational linguistics problems, that is, applications with human language technology needs. These packages are widely used in industry, academia, and government. In this tree, we identify verb (VB) for mapping methods (be-verb, do_verb, have_verb, and general_verb) of the abbot's textual definition.

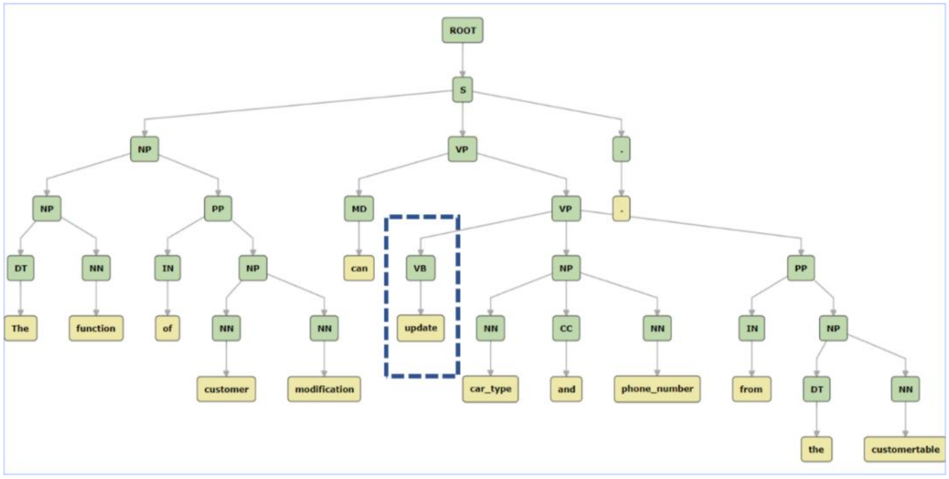


Figure 1. Language Analysis about Requirement

3. Cost Estimation based on Natural Language based Requirements

Estimating costs with requirements has been a lot of work. However, it is difficult to assure that these estimated costs are appropriate and accurate costs and sizes of the project. Currently, experts predict its size and cost compared to similar systems based on experience [11]. Predicting the scale of requirements is the first step in software development and an important task associated with cost, scheduling, and delivery.

However, in cost prediction, there are significant differences in size prediction between inexperienced and experienced groups [12]. If development costs are calculated differently depending on experience and knowledge, despite the same requirements, trust will be lost in the position of paying or receiving costs. In addition, it is difficult to predict and verify the cost for natural language-based requirements. Therefore, we propose cost prediction and verification automation based on natural language requirements.

This procedure to calculate Cost estimation for a project processe as follows:

1) Define functional requirements based on the customer’s needs
2) Adapt the abbott’s textual analysis into requirement sentence
3) Identify morphemes with requirement sentence based on Morphological analysis
4) Construct DB tables and store extracted data information into them
5) Extract Information of Function Point Factors in DB tables
6) Calculate Function Points

3.1 Functional Requirements (FR) Definition

We use some requirement specifications of the car dealer’s integration management system.

Table 2. Functional Requirements

FR1. The function of customer modification can update car_type and phone_number from the customertable. FR2. The function of customer registration can insert customer_name, phone_number, car_type and car_number to the customertable. FR3. The function of sale delete can delete customer_name, car_number, car_type, and phone_number from the saletable.

3.2 Abbott’s Textual Analysis

Abbott [13] used heuristic methods to map parts of speech to identify objects, attributes, and associations in the requirements specification for natural language analysis. In this paper, the method is modified to define requirements-defined conversion rules for function scores. Table 2 shows the rules for converting customer requirements into requirements for functional scores.

Table 3. Abbott's Textual Analysis

Part of speech	Function Component	Examples
Noun(Subject)	Function Name	customer registration
Verb	External Output(EO) External Input(EI) External Query(EQ)	export, calculate insert select
Noun(Object)	Data Element Type(DET)	customer_name
Preposition(in, to)+Noun	Record Element Type(RET)	to te customertable
Preposition(on) + noun	External Interface File(EIF)	on external MPVMS

In a sentence, the subject noun is designated by the functional name. The verb is designated as a transaction function types (EI, EO, EQ). The rest of the nouns that are not the subject nouns are designated as Data Element Types (DET). The Nouns used with prepositions (in, to) are designated as Record Element Types (RET). Nouns used with prepositions (on) are designated as External Interface Files (EIF). The Designated DETs and RETs are used as data function types.

3.3 Morphological analysis with Stanford Parser

Figure 2 is the result of analyzing functional requirements using morphological analysis. The verb ‘delete’ was branched from VP (Verb Phrase) to VB (Verb), and in this case, ‘delete’ corresponds to the EQ of the function point.

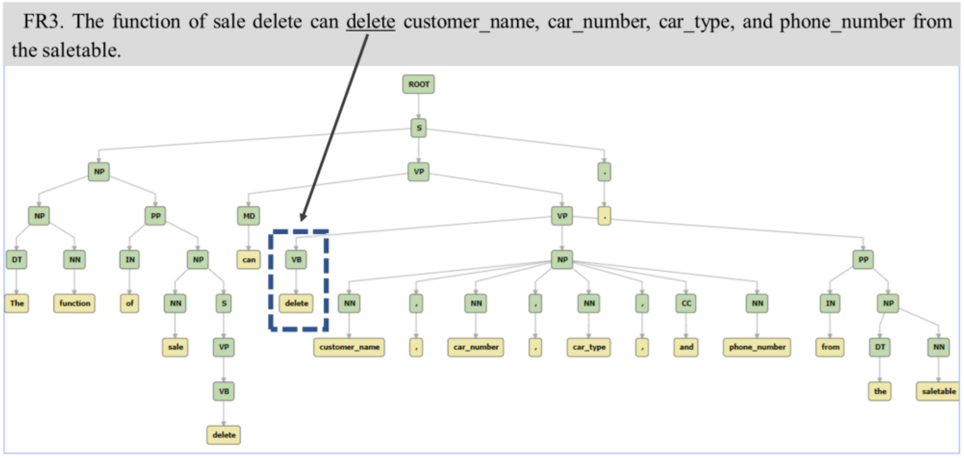

FR3. The function of sale delete can delete customer_name, car_number, car_type, and phone_number from the saletable.

Figure 2. Morphological Analysis of the Functional Requirement

3.4 DB table construction with extracted data information

Functional requirement FC-02-02 contains “The function of customer modification can update car_type and phone_number from the customertable.”. The FP_DATA table is a table for storing content corresponding to a data function of a function point. Project ID (P0001), function requirement ID (FC_02_02), data (car_type, phone_number), entity (customizable), and type (ILF) are stored in the FP_DATA table from function requirements (FC-02-02). The FP_FUNCTION_NAME table is a table for separately managing only the names of functions, and stores the project ID (P0001), function requirement ID (FC-02-02), and function name (customer_modification). The FP_TRANSACTIONS table stores the project ID (P0001), function ID (FC-02-02), transaction classification verb (update), and transaction type (EI) corresponding to the transaction function of the Function Point. Figure 3 is an example of a process of extracting information from functional requirements and storing it in a DB.

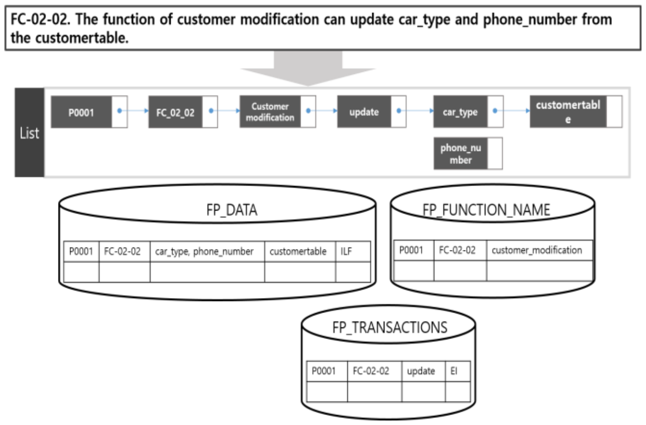
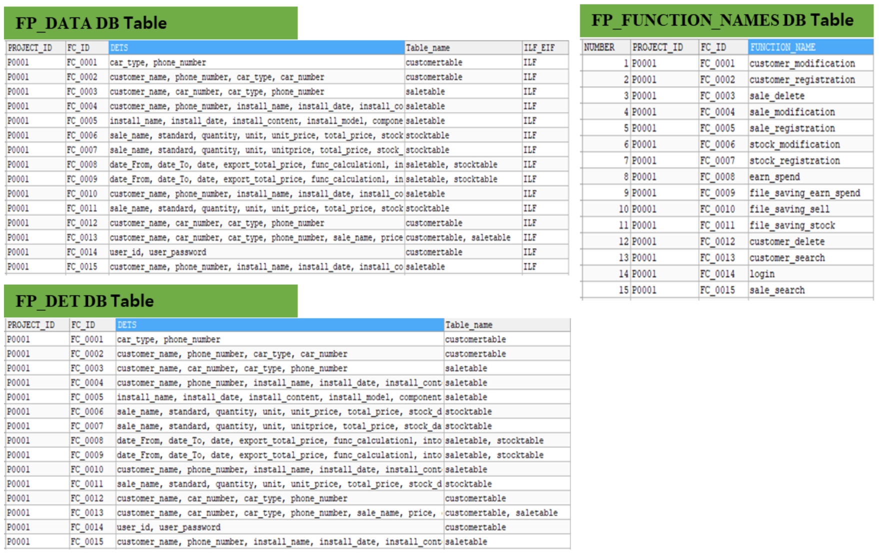

FC-02-02. The function of customer modification can update car_type and phone_number from the customertable.
List
P0001
FC_02_02
Customer modification
update
car_type
phone_number
customertable
FP_DATA
P0001 | FC-02-02 | car_type, phone_number | customertable | ILF
FP_FUNCTION_NAME
P0001 | FC-02-02 | customer_modification
FP_TRANSACTIONS
P0001 | FC-02-02 | update | EI

Figure 3. DB tables of Requirements

3.5 Information Extraction of Function Point Factors in DB table

The information stored in the FP_DATA table is later used to calculate the complexity of Internal Logic Files according to the number of ILF and DETs, and the complexity of External Information Files according to the number of EIF and DETs. Figure 4 is contents stored in the FP_DATA table. Information corresponding to EIF or ILF is stored in the FP_DATA table. Figure 4 is contents of the FP_DET table for searching for the DET of each function. Project ID P0001's functional requirements ID FC_0008 stores data_from, date_to, date, export_total_price, func_calculation1, total_price, func_calculation2, revenge, func_calculation3 as DETs. And salable and stocktable are stored as table names.



FP_DATA DB Table

PROJECT_ID	FC_ID	DETS	Table_name	ILF_EIF
P0001	FC_0001	car_type, phone_number	customertable	ILF
P0001	FC_0002	customer_name, phone_number, car_type, car_number	customertable	ILF
P0001	FC_0003	customer_name, car_number, car_type, phone_number	saletable	ILF
P0001	FC_0004	customer_name, phone_number, install_name, install_date, install_co	saletable	ILF
P0001	FC_0005	install_name, install_date, install_content, install_model, compone	saletable	ILF
P0001	FC_0006	sale_name, standard, quantity, unit, unit_price, total_price, stock	stocktable	ILF
P0001	FC_0007	sale_name, standard, quantity, unit, unitprice, total_price, stock_	stocktable	ILF
P0001	FC_0008	date_From, date_To, date, export_total_price, func_calculation1, in	saletable, stocktable	ILF
P0001	FC_0009	date_From, date_To, date, export_total_price, func_calculation1, in	saletable, stocktable	ILF
P0001	FC_0010	customer_name, phone_number, install_name, install_date, install_co	saletable	ILF
P0001	FC_0011	sale_name, standard, quantity, unit, unit_price, total_price, stock	stocktable	ILF
P0001	FC_0012	customer_name, car_number, car_type, phone_number	customertable	ILF
P0001	FC_0013	customer_name, car_number, car_type, phone_number, sale_name, price	customertable, saletable	ILF
P0001	FC_0014	user_id, user_password	customertable	ILF
P0001	FC_0015	customer_name, phone_number, install_name, install_date, install_co	saletable	ILF

FP_FUNCTION_NAMES DB Table

NUMBER	PROJECT_ID	FC_ID	FUNCTION_NAME
1	P0001	FC_0001	customer_modification
2	P0001	FC_0002	customer_registration
3	P0001	FC_0003	sale_delete
4	P0001	FC_0004	sale_modification
5	P0001	FC_0005	sale_registration
6	P0001	FC_0006	stock_modification
7	P0001	FC_0007	stock_registration
8	P0001	FC_0008	earn_spend
9	P0001	FC_0009	file_saving_earn_spend
10	P0001	FC_0010	file_saving_sell
11	P0001	FC_0011	file_saving_stock
12	P0001	FC_0012	customer_delete
13	P0001	FC_0013	customer_search
14	P0001	FC_0014	login
15	P0001	FC_0015	sale_search

FP_DET DB Table

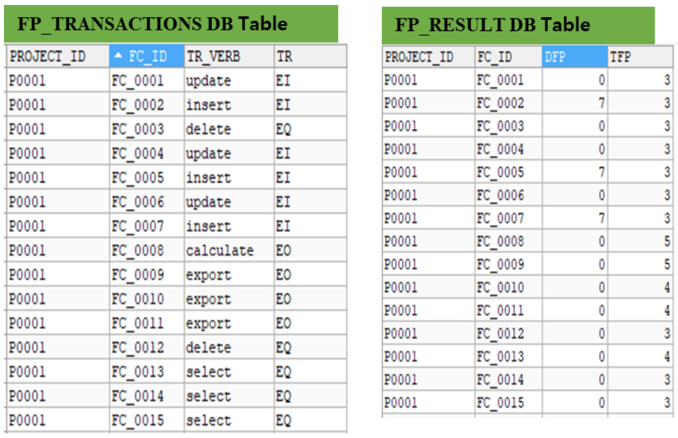
PROJECT_ID	FC_ID	DETS	Table_name
P0001	FC_0001	car_type, phone_number	customertable
P0001	FC_0002	customer_name, phone_number, car_type, car_number	customertable
P0001	FC_0003	customer_name, car_number, car_type, phone_number	saletable
P0001	FC_0004	customer_name, phone_number, install_name, install_date, install_cont	saletable
P0001	FC_0005	install_name, install_date, install_content, install_model, component	saletable
P0001	FC_0006	sale_name, standard, quantity, unit, unit_price, total_price, stock_d	stocktable
P0001	FC_0007	sale_name, standard, quantity, unit, unitprice, total_price, stock_da	stocktable
P0001	FC_0008	date_From, date_To, date, export_total_price, func_calculation1, into	saletable, stocktable
P0001	FC_0009	date_From, date_To, date, export_total_price, func_calculation1, into	saletable, stocktable
P0001	FC_0010	customer_name, phone_number, install_name, install_date, install_cont	saletable
P0001	FC_0011	sale_name, standard, quantity, unit, unit_price, total_price, stock_d	stocktable
P0001	FC_0012	customer_name, car_number, car_type, phone_number	customertable
P0001	FC_0013	customer_name, car_number, car_type, phone_number, sale_name, price,	customertable, saletable
P0001	FC_0014	user_id, user_password	customertable
P0001	FC_0015	customer_name, phone_number, install_name, install_date, install_cont	saletable

Figure 4. Function Point DB Table

Figure 4 is contents of the FP_FUNCTION_NAMES table for managing only function names. Functional requirements ID FC_0008 of project ID P0001 store earn_spend corresponding to the function name.

6) Function Point Calculation

Figure 5 is contents of the FP_TRANSACTIONS table that contains the contents of the transaction function. Functional requirements ID FC_0001 use the verb update, and the transaction function type is EI (External Input). Functional requirement ID FC_0009 uses the verb export, and the transaction function type is EO (External Output). Functional requirement ID FC_0015 uses the verb select, and the transaction function type is EQ (External Query). Figure 57 is the content of the FP_RESULT table, which stores the results for data functions and transaction functions by function ID using the function score complexity formula. The data function score is 7+7+7 = 21, and the transaction function score is 3+3+3+3+3+3+3+5+5+4+3+4+4+3+3=52. Therefore, DFP + TFP = 73. If the original requirement was from FC_0001 to FC_0012, the data function score is 7+7+7 = 21, and the transaction function score is 3+3+3+3+3+3+3+3+5+4+4+3=42. DFP + TFP = 63.



FP_TRANSACTIONS DB Table

PROJECT_ID	FC_ID	TR_VERB	TR
P0001	FC_0001	update	EI
P0001	FC_0002	insert	EI
P0001	FC_0003	delete	EQ
P0001	FC_0004	update	EI
P0001	FC_0005	insert	EI
P0001	FC_0006	update	EI
P0001	FC_0007	insert	EI
P0001	FC_0008	calculate	EO
P0001	FC_0009	export	EO
P0001	FC_0010	export	EO
P0001	FC_0011	export	EO
P0001	FC_0012	delete	EQ
P0001	FC_0013	select	EQ
P0001	FC_0014	select	EQ
P0001	FC_0015	select	EQ

FP_RESULT DB Table

PROJECT_ID	FC_ID	DFP	TFP
P0001	FC_0001	0	3
P0001	FC_0002	7	3
P0001	FC_0003	0	3
P0001	FC_0004	0	3
P0001	FC_0005	7	3
P0001	FC_0006	0	3
P0001	FC_0007	7	3
P0001	FC_0008	0	5
P0001	FC_0009	0	5
P0001	FC_0010	0	4
P0001	FC_0011	0	4
P0001	FC_0012	0	3
P0001	FC_0013	0	4
P0001	FC_0014	0	3
P0001	FC_0015	0	3

Figure 5. FP_TRANSACTIONS and FP_RESULT DB Table

4. Cost Validation based on Reverse Engineering

Public institutions do not trust the accuracy of software cost estimates. In the past, large companies did not ask for additional costs or recalculate costs in unconditional acceptance, even if requirements changed due to software development. In this situation, small and medium-sized companies continue to accept changes in requirements, affecting software quality and increasing the burden on companies. To solve this problem, we propose an automatic software cost validation method. Among the cost estimation models, Albrecht's function point is the most used [14] [15]. In general, function points are used to estimate the size, schedule, and cost of requirements or projects through requirements analysis. In the case of Korea, more development time is required than the initial requirements, and more costs are added due to underestimation of software development costs or acceptance of excessive changes in requirements. However, the proposed approach is essential because no one discusses whether the initial cost estimate was appropriate. Therefore, verifying whether the function point is calculated based on the implemented source code and whether it is valid when compared with the actual cost is essential. We propose an automated approach for verifying cost prediction based on reverse engineering.

1) Cost Validation Code Analyzer

A code analyzer parses the source code. The result of syntactic analysis is expressed as an abstract syntax tree (AST) and syntax tree [16]. In AST, variable definitions, loop statements, and conditional statements are all classified as statements. The For-Statement is represented by a node called For-Statement. The abstract syntax tree is extracted through the Java parser, and the code is analyzed to review the elements inside the code in the form of a tree. Internal nodes include packages, compilation units, binary classes, types, methods, and fields. Therefore, AST structure analysis is required for function point extraction. Figure 6 is a diagram of a Java-based abstract syntax tree.

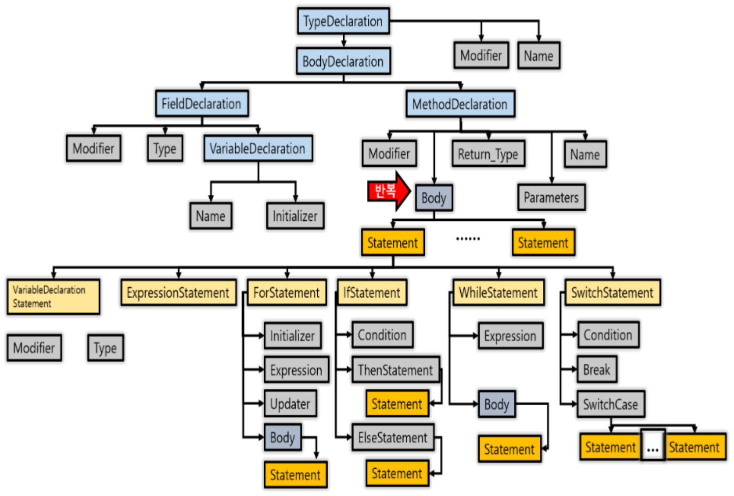


Figure 6. Abstract Syntax Tree for JAVA

Classes are classified from TypeDeclaration to BodyDeclaration to FieldDeclaration and MethodDeclaration. FieldDeclaration includes information about class variables, and Modifier has public, private, and protected accessor information. ‘Type’ has information on whether it is a basic data type provided by Java or a class data type. VariableDeclaration includes Name with variable name information and Initializer information with variable initialization value Similarly, MethodDeclaration has information about Modifier and Name, Return_Type has method return type information, and Parameters has method parameter information. If you look closely here, Body is composed of Statements, and Statements are composed of VariableStatement, ExpressionStatement, ForStatement, IfStatement, WhileStatement, SwitchStatement, and TryStatement. To analyze Java sentences, you need to implement code by analyzing the structures in Figure 6 and extracting them individually. In this paper, a code analyzer was developed to find attribute object binding, local object binding, parameter object binding, return object binding, inheritance binding, and interface binding between classes. Static analyzers of commercial tools or open-source SW do not include a function to extract function points. Therefore, for cost validation, a cost validation code analyzer was developed by finding the code corresponding to the function point in the source code and conducting a type of analysis.

2) Validation Result

Table 4 is the result of deriving function points based on reverse engineering. The sum of data function points is 21 points, the sum of transaction function points is 58 points, and the total function point is 79 points. In the results predicted based on requirements, there were no requirements for EI_wearDel (stock_delete, FC_0016) and EQ_wearSearch (stock_search, FC_0017). The difference in function points is due to the addition of two requirements. Also, as mentioned in Section 5.2, if the cost is predicted even when the initial requirements were from FC_0001 to FC_0012, the function score is 63 points.

Table 4. Reverse engineering-based function points

Method Name	Data Function Point	Transaction Function Point
EI_customerReg	7	3
EI_customerDelete	0	3
EQ_customerSearch	0	4
EQ_login	0	3
EI_CustomerMod	0	3

EI_sellReg	7	3
EI_sellMod	0	3
EI_sellDel	0	3
EQ_sellSearch	0	3
EI_wearReg	7	3
EI_wearMod	0	3
EI_wearDel	0	3
EQ_wearSearch	0	3
EO_exportExcelWear	0	4
EO_earnSpend	0	5
EO_exportExcelSell	0	4
EO_exportExcelEarnSpend	0	5
Total	21	58

Cost validation for requirements-based development cost and reverse engineering-based development cost compares the two values and calculates whether they are included in the error range.

$$C=\frac{ECS-ECR}{ECS}\times 100 \qquad \cdots\cdots\cdots\cdots\ \text{(1) formula}$$

$$-10\% < C < 10\% \qquad \cdots\cdots\cdots\cdots\ \text{(2) tolerance range}$$

$$C=\frac{31,993,286-29,563,417}{31,993,286}\times 100 = 7.59\% \qquad \cdots\cdots\cdots\cdots\ (3)$$

$$-10\% < C(7.59\%) < 10\% \qquad \cdots\cdots\cdots\cdots\ (4)$$

As a result of the calculation by substituting the value into formula (1) for case I, C is 7.59% in (3), so it is included within the tolerance range of (4). If included within this range, there is no additional request for cost. Conversely, if the cost calculated based on the code is lower than the cost predicted based on the requirements, it can be checked whether requirements have been changed or deleted, or there are requirements that have not been implemented. The following is a comparison for Case II.

$$C=\frac{ECS-ECR}{ECS}\times 100 \qquad \cdots\cdots\cdots\cdots\ \text{(1) formula}$$

$$-10\% < C < 10\% \qquad \cdots\cdots\cdots\cdots\ \text{(2) tolerance range}$$

$$C=\frac{31,993,286-25,513,634}{31,993,286}\times 100 = 20.25\% \qquad \cdots\cdots\cdots\cdots\ (3)$$

$$-10\% < C(20.25\%) < 10\% \qquad \cdots\cdots\cdots\cdots\ (4)$$

As a result of calculating by substituting the value into formula (1) for case II, in (3), C is 20.25%, which is outside the tolerance range of (4). As it is outside this scope, the proposed method provides a basis for requesting additional costs.

5. Conclusion

In this paper, we propose a requirement framework of forward/reverse engineering-based cost estimation and validation for requirements before/after software development. Using the proposed mechanism, first, it is possible to avoid uncertainty in requirements and increase clarity. Second, since cost estimation can be made based on requirements, we can solve problems to calculate differently depending on the knowledge of experts. Third, requirements satisfaction with process activities and byproducts can be tracked in all aspects of forward engineering/reverse engineering, improving the quality of the software. Fourth, "cost estimation" can be automatically proved through reverse engineering-based code visualization.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2021R1I1A1A01044060, 2021R1I1A3050407) in 2023.

References

[1] Paulo Jose Azevedo Vianna Ferreira, Marcio de Oliveira Barros, “Traceability between Function Point and Source Code”, The 6th International Workshop on Traceability in Emerging Forms of Software Engineering, pp.10-16, 2011
DOI: https://doi.org/10.1145/1987856.1987860

[2] Steven Klusener, “Source Code Based Function Point Analysis for Enhancement Projects”, International Conference on Software Maintenance, 2003
DOI: https://doi.org/10.1109/ICSM.2003.1235445

[3] Heon Ki Lee, "Automatic Measurement of Function Points from Java Applications", SEMCMI2015, 2015

[4] Vinh T. Ho, Alain Abran, “A Framework for Automatic Function Point Counting From Source Code”, International Workshop on Software Measurement, pp.248-255, 1999

[5] Tharwon Arnuphaptrairong, “Early Stage Software Effort Estimation Using Function Point Analysis: An Empirical Validation”, international Journal of Design, Analysis and Tools for Integrated Circuits and Systems, Vol. 4, No. 1, pp.15-21, 2013

[6] G. Antoniol, R. Fiutem, C. Lokan, “Object-Oriented Function Points: An Empirical Validation”, Empirical Software Engineering, vol.8, pp.225-254, 2003

[7] Chris F, Kemerer, “An Empirical Validation of Software Cost Estimation Models”, Communication of the ACM, Volume 30, Number 5, pp.416-429, 1987
DOI: https://doi.org/10.1145/22899.22906

[8] T. Edagawa, T. Akaike, "Function point measurement from Web application source code basedon screen transitions and database accesses", The Journal of Systems and Software, pp.976-984, 2011
DOI: https://doi.org/10.1016/j.jss.2011.01.029

[9] G.C. Low, D.R. Jeffery, "Function Points in the estimation and evaluation of the software process", IEEE Transactions on Software Engineering, Vol. 16, Issue.1, pp.64-71, 1990
DOI: https://doi.org/10.1109/32.44364

[10] So Young Moon, “Requirement framework for cost estimation and automatic validation based on forward/reverse engineering”, Hongik University, 2019

[11] Wansik Kim, “Expert judgement as an estimating method”, Information and Software Technology, Volume 38, Issue 2, 1996, Pages 67-75, 1996
DOI: https://doi.org/10.1016/0950-5849(95)01045-9

[12] J. H. Hayes, A. Dekhtyar, and S. K. Soundaram, "Advancing candidate link generation for requirements tracing: The study of methods", IEEE Trans. Soft. Eng., vol. 32, No. 1, pp.4-19, Jan. 2006
DOI: https://doi.org/10.1109/TSE.2006.3

[13] Bernd Brugge, Allen H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, Prentice Hall, 2010

[14] Allan J. Albrecht, John E. Gaffney, “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE Transactions of Software Engineering, Vol. SE-9, No.6, 1983D
DOI: https://doi.org/10.1109/TSE.1983.235271

[15] A. J. Albrecht, "Measuring Application Development Productivity", Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, IBM Corporation, pp.83–92, 1979

[16] Kenneth C. Louden, "Compiler Construction Principles and Practice (1st ed.)", Course Technology, 1997