Function Point Measurement Tool for UML Design Specification

Takuya UEMURA[†], Shinji KUSUMOTO[†], and Katsuro INOUE[†], [‡]
[†] Graduate School of Engineering Science, Osaka University,
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Tel: +81-6-6850-6571, Fax: +81-6-6850-6574
E-mail: {uemura, kusumoto, inoue}@ics.es.osaka-u.ac.jp
[‡] Graduate School of Information Science, Nara Institute of Science and Technology

Abstract

Function point analysis(FPA) was proposed to help measure the size of a computerized business information system. It is widely used in the actual software development. However, it has been reported that since function point counting involves judgment on the part of the counter, some difference for the same product would be caused even in the same organization. In this paper, we propose detailed FPA measurement rules for the design specifications based on the UML(Unified Modeling Language) and develop the function point measurement tool, whose input products are design specifications on Rational Rose. We have also applied the tool to the actual design specification and examine the difference between the values by the tool and one by the specialist of FPA. The results show the applicability of our tool.

1. Introduction

As the size and the complexity of software increase, it becomes important to develop high-quality software cost-effectively within a specified period. In order to attain it, it is necessary to manage the entire software development processes based on the effective project plan.

In order to construct the distinct project plan, it is essential to estimate various phenomena happened in the project and take measures to meet them in advance. A subject of estimation about software development is size, effort invested, development time, technology used and quality. Especially development effort is the most important subject.

There have been proposed a lot of effort models and most of them include software size as an important parameter. In the models, LOC (lines of codes) is often adopted. However, using LOC as the software size has difficulties because the definition of LOC is very vagueness and LOC depends on the programming language.

Function point is a measure of software size that uses logical functional terms business owners and users more readily understand [1]. Since it measures the software requirements or business models, the measured size stays constant despite the programming language, design technology, or development skills involved. Also, it is available early in the development process, making its use opportune for planning the design and development projects. Up to the present, various FPA versions based on the Albrecht's version have been proposed. IFPUG (International Function Point Users Group) version [2] and MarkII version [13] are frequently used in software organizations.

However, it has been reported that since function point counting involves judgment on the part of the counter, some difference for the same product would be caused even in the same organization [6]. For example, G. Low and R. Jeffery reported that a 30-percent variance was caused within an organization and more than 30-percent variance was caused across organizations[8]. In order to get consistent value, it is important to automate the function point measurement[1]. In the past, function point measurement could not be completely automated because there were no machine readable requirements/design specifications. Recently, CASE tools have been providing machine readable ones. Now, it is easy to introduce the function point measurement tools for the requirements/design specifications produced by the specific CASE tool.

On the other hand, recently, many companies have started to introduce object-oriented technology into their software development environments and the object-oriented development technologies are often used in software organizations. However, it has created new challenges for companies and researchers which use software metrics as a tool for managing the software and the process [4]. That is, it is necessary to establish the method to calculate the software metrics from the object-oriented software.

This paper proposes detailed FPA measurement rules for

the design specifications using the UML (Unified Modeling Language) and develop the function point measurement tool. With respect to the development using the UML, several CASE tools have been developed. Among them, Rational Rose by Rational Software is the most prevalent one and widely used in software development organizations. Thus, as the input of the tool, we address the design specifications on Rational Rose. Finally, We apply the tool to the actual design specification and examine the difference between the value by the system and one by the specialist of FPA. Then, we examine the difference between them and discuss the applicability of our tool.

Section 2 describes the overview of function point analysis, the IFPUG version and the several diagrams by the UML. Next, Section 3 proposes detailed rules to count the function point from the diagrams by UML. Section 4 describes a function point measurement system based on the proposed rules and case study that evaluates the applicability of our system. Finally, Section 5 concludes this paper.

2. Preliminaries

2.1. Function Point Analysis

Function point measures the functionality provided by software. It can be determined from the requirements specification, design specification and program code. Unlike LOC, since function point measures functionality, it should be independent of the technology and language used for the software implementation.

Allan Albrecht first proposed original function point analysis [1]. Albrecht's function point is computed by counting the following software characteristics:(1)External inputs and outputs, (2)User interactions, (3)External interfaces and (4)Files used by the system. Each of them is then individually assessed for complexity and given a weighting value which varies from 3 (simple) to 15 (complex).

Albrecht's function point has been widely used but it has some weakness. Thus, many kinds of function point, such as IFPUG version[2], 3D Function Points version[5], Feature Points version[5] and MarkII version[13], have been proposed. In this paper, we address the IFPUG version since it provides the detail procedures and rules for function point counting compared to other versions.

2.2. IFPUG version

IFPUG version is a modified-version of the Albrecht's function point. In the modification, the evaluation of the complexity of the software was objectively established and the rules of the counting procedures were also described minutely and precisely.

In the IFPUG version, the counting procedure of function point consists of the following seven steps[2].

Step1 (Determine the Type of Function Point Count): Select the type of function point from the following three ones:(1) Development project function point count, (2)Enhancement project function point count and (3)Application function point count.

Step2 (Identify the Counting Boundary): A boundary indicates the border between the application or project being measured and the external applications or the user domain. A boundary establishes which functions are included in the function point count.

Step3 (Count Data Function Types): Data function types represent the functionality provided to the user to meet internal and external data requirements. Data function types are classified into the following two types: Internal logical file(ILF) and External interface file(EIF).

The definition of data functions are described as follows:

Internal Logical File(ILF): (1)The group of data is user identifiable group of data. (2)The group of data is maintained within the application boundary. (3)The group of data identified has not been counted as an EIF for the application.

External Interface File(EIF): (1)The group of data is user identifiable group of data. (2)The group of data is not maintained by the application being counted. (3)The group of data identified has not been counted as an ILF for the application.

Here, the term "file" refers to a logically related group of data and not to the physical implementation of those group of data.

Then, assign each identified ILF or EIF a functional complexity based on the number of data element types (DETs) and record element types (RETs) associated with the ILF or EIF using the RET/DET complexity matrix(See Table 1). A data element type (DET) is a unique user recognizable, nonrecursive field on the ILF or EIF. A record element type(RET) is a user recognizable subgroup of data elements within an ILF or EIF.

Table 1. RET/DET complexity matrix

RET\DET	1-19	20-50	51-
1	Low	Low	Average
2-5	Low	Average	High
6-	Average	High	High

Step4 (Count Transactional Function Types):

Transactional function types represent the functionality provided to the user for the processing of data by an application. They are defined as the following three types: External input(EI), External output(EO) and External inquiry(EQ). The definition of transactional functions are described as follows:

External input(EI): An external input processes data or control information that comes from outside the application's boundary. The external input itself is an elementary process.

External output(EO): An external output is an elementary process that generates data or control information sent outside the application's boundary.

External inquiry(EQ): An external inquiry is an elementary process made up of an input-output combination that results in data retrieval. The output side contains no derived data. Here, derived data is data that requires processing other than direct retrieval and editing of information from internal logical files and/or external interface files. No internal logical file is maintained during processing.

Then, assign each identified EI or EO a functional complexity based on the number of file types referenced (FTRs) and data element types (DETs).A file type referenced is ,(1) An internal logical file read or maintained by a function type, or (2) An external interface file read by a function type. Also, assign each EQ a functional complexity based on the number of file types referenced (FTRs) and data element types (DETs) for each input and output component. Use the higher of the two functional complexities for either the input or output side of the inquiry to translate the external inquiry to unadjusted function points. For each of EI, EO and EQ, there is a FTR/DET complexity matrix. Table 2 shows the FTR/DET complexity matrix for EI.

Table 2. FTR/DET complexity matrix of EI

FTR\DET	1-4	4-15	16-
0-1	Low	Low	Average
2	Low	Average	High
3-	Average	High	High

Step5 (Determine the Unadjusted Function Point Count):

As the result of Step3 and Step4, the counts for each function type are classified according to complexity and then weighted using the Table 4. The total of all the function types is the unadjusted function point count.

Table 3. General System Characteristics

Article	GSC
1	Data communications
2	Distributed data processing
3	Performance
4	Heavily used configuration
5	Transaction rate
6	Online data entry
7	End-user efficiency
8	Online update
9	Complex processing
10	Reusability
11	Installation ease
12	Operational ease
13	Multiple sites
14	Facilitate change

Table 4. Unadjusted Function Point Calculation Table

	Complexity			Total
	Low	Average	High	
ILF	□ × 7= □	□ ×10= □	□ ×15= □	
EIF	□ × 5= □	□ × 7= □	□ ×10= □	
EI	□ × 3= □	□ × 4= □	□ × 6= □	
EO	□ × 4= □	□ × 5= □	□ × 7= □	
EQ	□ × 3= □	□ × 4= □	□ × 6= □	
		Unadjusted Function Point		

Step6 (Determine the Value Adjustment Factor):

The value adjustment factor (VAF) indicates the general functionality provided to the user of the application. VAF is comprised of 14 general system characteristics that assess the general functionality of the application. Each characteristic has associated descriptions that help determine the degree of influence of the characteristic. The degree of influence ranges on a scale of 0 to 5, from no influence to strong influence. (See Table 3.)

Finally, VAF is calculated based on the following formula, $\text{VAF} = 0.65 + \frac{\text{total}}{100}$.

Step7 (Calculate the Final Adjusted Function Point Count):

The final adjusted function point count is calculated using a specific formula for development project, en-

hancement project or application based on the result of Step1.

IFPUG version has been widely used in software organizations and also will be included into the ISO standard.

2.3. Unified Modeling Language

The Unified Modeling Language (called UML) [14] was developed to provide a common language for object-oriented modeling. It was designed to be extensible in order to satisfy a wide variety of needs and was also intended to be independent of particular programming language and development methods.

The concrete syntax of the UML is dominated by a graphical notation. The UML defines a large number of different diagrams. They are divided into following three categories: Static structure diagrams, Behavior diagrams and Implementation diagrams.

Static structure diagrams describe the structure of the system and include class diagrams and object diagrams. Behavior diagrams describe the behavior/dynamic perspective of the system and include use-case diagrams, interaction diagrams, sequence diagrams, collaboration diagrams, state diagrams and activity diagrams. Implementation diagrams provide the information of actual source code and includes component diagrams and deployment diagrams.

In order to calculate the function point from the above diagrams, we use the sequence diagrams and class diagrams. Because these diagrams include the information about all functions and data manipulated in the system. In subsections 2.4 and 2.5, we briefly explain the class diagrams[10] and sequence diagrams[10].

2.4. Class diagrams

Class diagrams describe the static structure of the model, that is objects, classes and the relations between these entities including generalization and aggregation. They also represent the attributes and operations of the classes.

For example, Figure 1 shows the class diagrams for Person, Student and Teacher that are used in the office system in a University. In Figure 1, the class Person has two attributes (Name and Address) and four operations (SetName, SetAddress, GetName and GetAddress). Other classes, Student and Teacher, inherit the information in the parent class Person.

2.5. Sequence diagrams

A sequence diagram shows an interaction arranged in time sequence. In particular, it shows the objects participating in the interaction by their “lifelines” and the messages that they exchange arranged in time sequence. It does not show the associations among the objects. Sequence diagrams show the explicit sequence of messages and are better for real-time specifications and for complex scenarios.

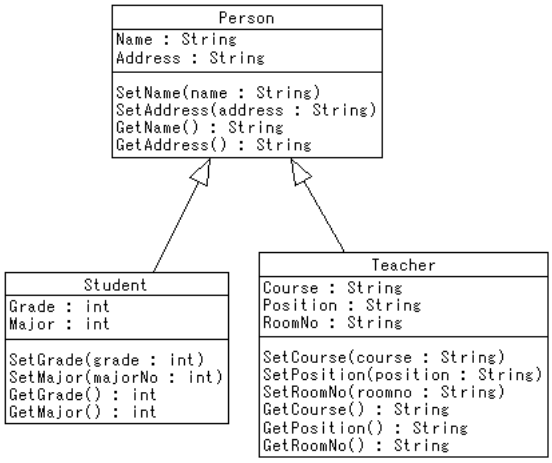

Figure 1. Example of the Class diagram

The sequence diagram represents an interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result.

The sequence diagram has two dimensions: the vertical dimension which represents time and the horizontal dimension which represents different objects. Normally time proceeds down the page. There is no significance to the horizontal ordering of the objects. Objects can be grouped into “swimlanes” on a diagram.

Figure 2 shows an example of a sequence diagram of the office system in a University. In Figure 2, there are nine messages and the set of these messages correspond to the following processing: (1)a student hands his/her registration sheet of the lectures, that he/she wants to attend, to the receptionist in the office of the University, (2)the receptionist inputs the data to the database through the Registration system and (3)the student receives the result of the registration.

3. Proposed Rules for Function Point

3.1. Overview

We aim to calculate the Unadjusted Function Point. We propose the following five steps to apply IFPUG version to the requirements/design specifications (class diagrams and sequence diagrams) based on the UML.

Step1 (Determine the Type of Function Point Count):

We handle only the development project function point.

Step2 (Identify the Counting Boundary):

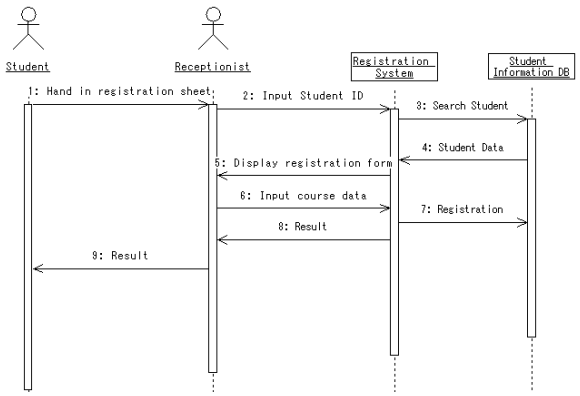

Figure 2. Example of the Sequence diagram

The counting boundary is determined by the type of objects which appear in the sequence diagrams. That is, actor objects are outside the boundary and other objects are inside the boundary.

Step3 (Count Data Function Types):

Data function types are automatically decided based on the information of the class and sequence diagrams according to the rules explained in subsection 3.2.

Step4 (Count Transactional Function Types):

Transactional function types are automatically decided based on the information of the class and sequence diagrams according to the rules explained in subsection 3.3.

Step5 (Determine the Unadjusted Function Point Count):

As the result of Step3 and Step4, the counts for each function type are automatically classified according to complexity and then weighted. The total for all function types is the unadjusted function point count.

3.2. Rule of counting data function types

Based on the definitions of data function, we propose following rules to extract the data functions from class and sequence diagrams. Here, we classify the objects into actor and non-actor object. Since actor objects exist outside of the application, they are not regarded as the data function.

Step1: Select candidates of data functions:

We select the objects, that have some attributes and exchange data with not-actor objects, as the candidates of data functions.

Step2: Determine function type:

For each of the candidates selected in Step1, we determine the function type. Objects that have operations which change the attributes of other objects in exchanging the data are regarded as ILF. Others are regarded as EIF.

Step3: Judge complexity of data function

Complexity of ILF and EIF is determined by the data element type(DET) and the record element type(RET). Since the DET is a unique user recognizable, nonrecursive field on the ILF or ELF, we count the number of attributes of the corresponding class. If the class is derived from other class, the number of attributes of the base class is also added.

On the other hand, the RET cannot be counted from the class and sequence diagrams. However, from our previous experience, the RET is almost one in requirements/design specification. So, we consider that the RET is one.

Finally, the functional complexity is rated based on the RET/DET complexity matrix (See Table 1).

3.3. Rule of counting transactional function types

In accordance with IFPUG rules, we regard each of the messages, which is exchanged by the object specified as data function in sequence diagrams, as the candidate of transactional function. Then, if a message has no arguments, it means that it doesn't exchange data and so we determine that it is not a transactional function.

At first, in order to count the transactional function from the class and sequence diagrams, we assume the following restrictions:

- Assume that the sender object sends a message to other object (receiver object) in the sequence diagram. If the receiver object returns meaningful message to the sender one, the reply message must be precisely described in the sequence diagram (According to the definition of the sequence diagrams of the UML, it is not necessary to describe it).
- Data exchange must be written as the arguments of the messages in the sequence diagram.
- When an argument of the message whose name is the same as the sender object's attribute, we recognize the data stored in the argument is simply sent from the sender object.
- If a message is repeatedly appeared in the sequence diagrams, the arguments and message names must be the same.

For each actor object in the sequence diagrams, we apply the following two steps to count the transactional function types. These steps are based on the fact that function types

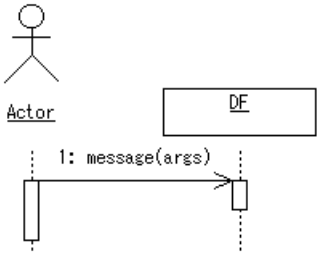

Figure 3. Pattern 1

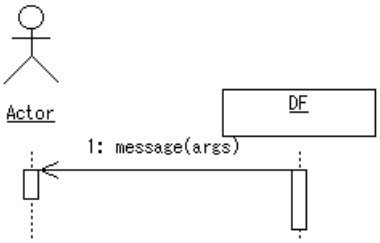

Figure 4. Pattern 2

of the transactional function can be determined by renewal or reference of data functions or comparison of data elements outputted.

Step1: Select candidates of transactional function: List the sequence of messages that the first message is sent by the actor object and the last message is received by the actor object or non-actor object in the sequence diagram.

Step2: Determine the type of transactional function: For each sequence listed in Step1, using the following five patterns, we determine the type of the transactional function and the complexity (DET and FTR) of them.

Pattern 1: An actor object sends message to a DF(Data Function) (See Figure 3)

We regard this pattern as External Input. If no arguments are given on the message, we don't regard it as EI. DET is the number of arguments of the message. FTR is 1 since there is one DF.

Pattern 2: A DF sends message to an actor object (See Figure 4)

If all the arguments of the message are the same as the attribute of the DF, we regard it as External Inquiry. Otherwise, it means that the message contains derived data. Then, we regard it as External Output. DET is the number of arguments of the message. FTR is 1 since there is one DF.

Pattern 3: An actor object sends message to a DF, and returns message from the DF to the actor object(See Figure 5)

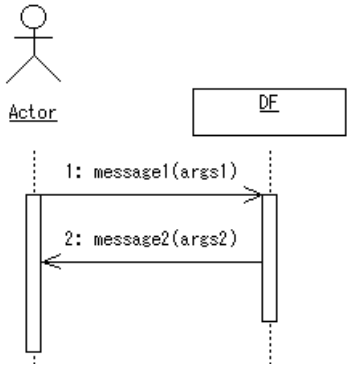

Figure 5. Pattern 3

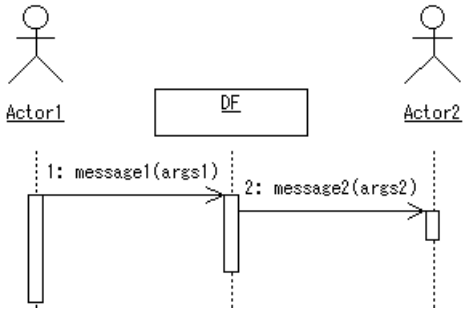

Figure 6. Pattern 4

In Figure 5, pay attention to a "message2", if all the arguments of the "message2" is the same as the attributes of the DF, we regard two messages (message1 and message2) as one External Inquiry. Otherwise, it means that the "message2" contains elementary process, we regard two messages as one External Output. In other words, we consider that the "message1" from the actor is the input of the key for data retrieval.

DET is the number of arguments of outputted message (message2). FTR is 1 since there is one DF.

Pattern 4: An actor object sends message to a DF and the DF sends message to another actor object (See Figure 6)

We divide it into two transactional functions. That is, Pattern 4 is the combination of the Pattern 1 and the Pattern 2.

Pattern 5: An actor object sends message to a DF and finally the actor object receives the reply message through several DFs (See Figure7)

Pattern 5 handles a sequence of messages. That is the case that when an actor object sends a message, the reply message comes through several DFs. For example, in Figure 7, a sequence of

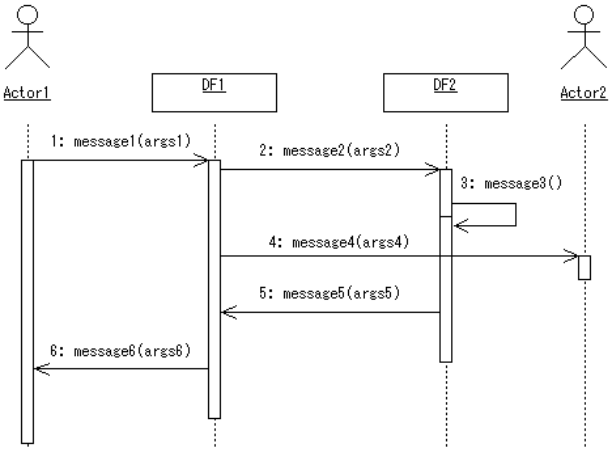

Figure 7. Example of Pattern 5

five messages (message1,2,3,5,6) is a candidate of transactional function. Then, we apply Pattern3 to message1 and message6. In Figure 7, since message4 is not included in the sequence, we apply other pattern (in this case, Pattern2) separately to the message.

DET is the number of arguments of outputted message (message6). FTR is 2 since there are two DFs in the sequence.

4. Function Point Measurement Tool and its application

4.1. Outline of system

We have designed and implemented the tool that can compute function points of the design specification developed by Rational Rose version 4.0 using Visual C++ language on Windows98. The inputs for the tool are sequence diagrams and class diagrams of the Rational Rose and the output includes the values of function points, transactional functions, data functions and objects which may be related to the function points calculation.

The system structure is shown in Figure 8. The system consists mainly of three units and two database: Analysis unit, Analysis database, Counting unit, Counting database and Interface unit.

Analysis unit executes syntax analysis to the input files (that is, sequence diagrams and class diagrams of the Rational Rose), extracts data functions and stores them into the Analysis database. Counting unit calculates the value of function points using the data in the Analysis database and stores them into the Counting database. Interface unit shows the measurement results, that includes the value of function point, candidates for the data functions and transactional functions.

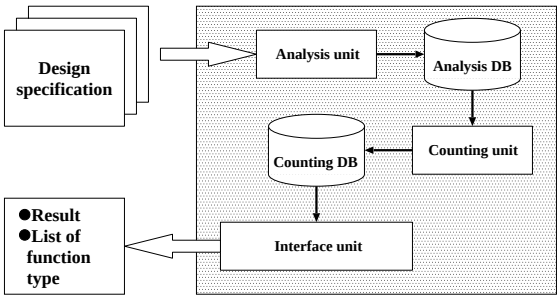

Figure 8. Outline of the system

4.2. Case study

We applied the function measurement tool to the actual design specifications for typical business application system developed by the Rational Rose as follows:

Purchase processing system: It makes out several documents to purchase office equipments.

Order processing system: It manages to make up an estimate for the office equipments and decide the shop to which they are ordered.

Stock Control system: It controls the stock of the office equipments and makes the production plans.

Figure 9 shows the part of the design specification for the Purchase processing system.

We calculated the function points from the specifications by our tool. Also, a function point analysis specialist (not an author of this paper) of a software organization calculates the function points manually from the same specifications. The measurement results are shown in Table 5.

With respect to the data function, the values measured by the tool are quite similar to the ones by the specialist. There are no differences among them.

On the other hand, with respect to the transactional function, there are some differences among them. These differences are caused by that in the several sequence diagrams, there is only one External inquiry, but the specialist considered that this transactional function would consist of two transactional functions and counted two External inquiries. If the detailed descriptions existed in the sequence diagrams, our tool could counted the transactional functions correctly.

Thus, we can conclude that our proposed rules and measurement system are considerably useful for the practical software development.

5. Conclusions

In this paper, we have proposed detailed function point analysis rules for design specification developed based on

Table 5. The measurement results

	Function point analysis specialist			Our tool		
	Purchase	Order	Stock Control	Purchase	Order	Stock Control
Data Function	14	29	26	14	29	26
Transactional Function	18	63	36	16	50	33
Total	32	92	60	30	79	59

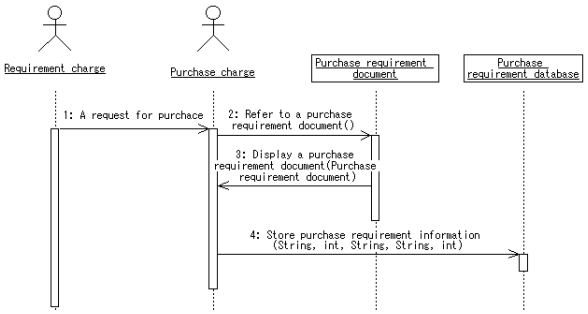

Figure 9. A part of the design specification

the UML. Then, based on the proposed rules, we have developed the function point measurement tool. We have also applied several design specifications for typical business application. The values calculated by the tool are considerably adequate.

Class and sequence diagrams may be available only considerably later during the development process than the time when function points are usually applied. However, since these diagrams can be easily constructed from use-case and if we have enough experience data about class and sequence diagrams, our system could be useful in the actual software development.

Future research works include the following:

1. The target products of our tool are UML Version 1.1, IFPUG Version 4.0, and Rational Rose Version 4.0. Now, UML Version 1.3, IFPUG Version 4.1 and Rational Rose 98 have just been released. We are going to modify our tool to cope with the new products.

2. In order to show the validity of our technique, we will apply our tool to many software development projects in actual software organizations.

3. Our proposed rules are based on only class and sequence diagrams. We must examine whether other diagrams can be applicable to measure function point more precisely.

Acknowledgments

We would like to thank Mr. Takashi Kasimoto, Mr. Michio Tsuda, Mr. Katuhiko Yuura, Ms. Ayane Suzuki, Mr. Kenji Hatsuta, Ms. Mari Morita and Mr. Makoto Kurashige of Hitachi, Ltd. for their discussions and advises in this paper.

References

[1] A. J. Albrecht. Function point analysis. *Encyclopedia of Software Engineering*, 1:John Wiley & Sons, 1994.

[2] A. J. Albrecht. *Function Point Counting Practices Manual, Release 4.0*. International Function Points Users Group, 1994.

[3] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, 9(6):639–648, 1983.

[4] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 20(22):751–761, 1996.

[5] C. Jones. *Applied Software Measurement*. McGraw-Hill, 1996.

[6] B. A. Kitchenham. The problem with function points. *IEEE Transactions on Software Engineering*, 14(2):29–31, 1997.

[7] L. Lian, S. Kusumoto, T. Kikuno, K. Matsumoto, and K. Torii. A new fault localizing method for program debugging process. *Information and Software Technology*, 39:271–284, 1997.

[8] G. C. Low and D. R. Jeffery. Function points in the estimation and evaluation of the software process. *IEEE Transactions on Software Engineering*, 16(1):64–71, 1990.

[9] T. Quatrani. *Visual Modeling with Rational Rose and UML*. Addison Wesley Longman, 1998.

[10] Rational. *UML 1.1 Notation Guide*. Rational Software, 1997.

[11] M. Saito, N. Onari, K. Yuura, and T. Kameda. Visualizing tool for required specifications. *The Hitachi Hyoron*, 77(12):15–18, 1995.

[12] I. Sommerville. *Software Engineering*. Addison-Wesley, 1995.

[13] C. Symons. *Software Sizing and Estimating*. John Wiley & Sons, 1991.

[14] S. Zamir. *Handbook of Object Technology*. CRC Press, 1999.