Automatic Measurement of Function Points from Java Applications

Heon Ki Lee
Department of Software Engineering, Korea University
Seoul, Republic of KOREA
leee@korea.ac.kr

ABSTRACT

Function points have been widely used to measure the functional size and productivity of software by various approaches: structured or object-oriented software methods. But none of these methods have yet been fully automated. All of them still require the involvement of an expert in order to be used correctly. To address this need, this paper describes a new kind of UML-to-FP mapping in order to automatically measure the FP from the combination of class and sequence diagrams based on the IFPUG 4.3.1. The mapping process of UML-to-FP mapping consists of the two main steps: the first identifies the data function types; the second identifies the transaction function types.

KEYWORDS

Function Points, IFPUG 4.3.1, UML-to-FP mapping, UML diagram, Reverse Engineering.

1 INTRODUCTION

Function points (FP) have been used for more than 30 years for measuring software functional size: development project, enhancement project and application project. Today, function points and Source Lines of Code (SLOC) are most widely used methods. However, SLOC approach model is less useful to measure functional size because SLOC depends on the programming languages [1]. First developed by Albrecht [2], it was an attempt to overcome difficulties associated with lines of code as a measure of software functional size, and to assist in developing a mechanism to predict effort associated with software development. Function points are a unit of measurement to express the amount of business functionality a product provides to a user. The method has been maintained and updated by the IFPUG 4.3.1 [3]. Function points are a precise measurement of software functional size designed to remove the ambiguity from consideration of the software being examined.

Since 1999, function points measurement based on Unified Modeling Language (UML) has proposed several methods to calculate the size of FP from the object-oriented software, such as [4], [5], [6], [7]. The feature of this approach considers UML diagrams such as use case, class and sequence diagrams. Thus, measuring FP based on UML diagrams has more advantages on the object-oriented software and also may give the stakeholders better understanding on the project. Unfortunately, several unsolved UML-to-FP mapping still remain. We have tried to automatically identify FP with UML class and sequence diagrams generated from Java programs.

This paper proposes an automatic function point measurement method for object-oriented software application using UML class and sequence diagrams only. In this approach, the class diagram determines the data function which relates to logical data, while the sequence diagram determines the transaction function which corresponds to elementary processes. It is our purpose to automate an object oriented function points measurement with detailed UML-to-FP mapping.
2 OVERVIEW OF UML-TO-FP MAPPING

A function point is a unit of measure for software functional size as defined within the IFPUG 4.3.1 [3]. The functional size enables the estimate of cost and resources required for software development and maintenance regardless of any nonfunctional constraints [8]. This paper aims at measuring the functional size of object-oriented software application based on IFPUG 4.3.1 using the combination of class and sequence diagrams, as in the figure 1. According to this step, the UML-to-FP mapping make use of class diagram for counting data functions whereas transaction functions are obtained from sequence diagram based on class diagram. The data functions relate to the logical data stored and available for update and retrieval [8]. The transaction functions perform the processes of storing, updating, retrieving, and displaying the logical data [8].

![Figure 1. UML-to-FP Mapping Step](image)

Table 1. Unadjusted Function Point Count Matrix

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Complexity</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Function</td>
<td>ILF</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>EIF</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Transaction</td>
<td>EI</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Function</td>
<td>EO</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>EQ</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

As described in IFPUG 4.3.1, FP is divided into two components: data function and transaction function [3]. The data function types are classified into internal logical file (ILF) and external interface file (EIF). The transaction function types are further classified into external input (EI), external output (EO) and external query (EQ). Therefore, there are five function types. Each type is furthermore classified into low, medium or high complexity, according to Table 1.

UML is designed to provide a standard way to visualize the design of a system [9]. UML diagrams represent two different views of a system model. First, static (or structural) view emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams as in figure 2 which correspond to the logical data directly. Finally, the dynamic (or behavioral) view emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams as in figure 3 which correspond to the transaction units as elementary process.

![Figure 2. Class Diagram describing Data Functions](image)
3 UML-TO-FP MAPPINGS

3.1 UML-to-FP mapping Steps

Based on the concepts presented in previous section, our purpose is to automatically count the unadjusted function point count based on IFPUG 3.4.1. We propose the following four steps (UML-to-FP mapping) to measure the software functional size of an existing Java application based on UML class and sequence diagrams only.

Step1. Extract UML diagrams from the source code of Java system.

Both UML class and sequence diagrams are generated from the Java programs by our reverse engineering tool with some limited relationship as in figure 2, 3:

- The components of class diagram tool are:
  - class, interface, package
  - relationship (except multiplicity): association, generalization, composition, dependency
- The components of sequence diagram tool are:
  - actor, object, message, lifeline, focus of control, object destructor

Step2. Identify the data function types.

Data functions are automatically identified based on classes in the class diagrams and data function types are determined by the boundary of objects specified as data function in an elementary process of sequence diagrams, according to the rules explained in Section 3.2.

Step3. Identify the transaction function types.

Transaction functions are automatically identified using elementary processes obtained from the sequence and the transaction function types are determined by both the boundary of elementary process and the boundary for objects specified as data functions in the elementary processes using sequence diagrams, according to the rules explained in Section 3.3.

Step4. Determine the unadjusted function point count.

As the result of Step 2 and Step 3, the counts for each function type are classified according to complexity and then weighted using the Table 1. Finally, the unadjusted function point count is the sum of the number of function points assigned to each of the functions.

3.2 Data Function Identification Rules

As referred to in IFPUG 4.3.1, a data function represents functionality provided to the user to meet internal and external data storage requirements [3]. A data function is either an Internal Logical File (ILF) or an External Interface File (EIF), where the term file does not mean physical file or class, but refers to a logically related group of data. ILF is maintained within the boundary of the application being measured, and EIF is read or referenced only within the boundary of the application being measured but are maintained within a different application boundary [8].

Based on the definitions of the data function [3], we propose the following stepwise rules to extract the data functions from the class and sequence diagrams, as in the figure 4.
Step 1. Search candidates of data functions.
From the class diagrams (figure 2), the candidate data functions are regarded as the concrete class (except interface) if they satisfy the following condition:
- A class that has attributes (data) and methods (transactions)

Step 2. Identify data functions.
Among candidates for data functions, all concrete classes if they satisfy the following condition are identified as the data function:
- A concrete class that has no children

The class diagram in Figure 2 illustrates the concrete classes that correspond to data function directly.

Step 3. Classify the data function types.
The data function types are determined by the boundary of objects specified as data function in an elementary process of sequence diagrams (figure 3). However, if some data function are not maintained or referenced by the elementary process, the data function also is excluded. A data function is classified as ILF if elementary processes in the sequence diagrams maintain (create, update, or delete) objects specified as the concrete class. Other classes are regarded as EIF.

Step 4. Identify complexity of data functions.
The complexity of ILF and EIF is determined by the data element type (DET) and the record element type (RET) (Table 1). Since the DET for an ILF or EIF is a unique user recognizable, non-repeated attribute [3], DET is the number of attributes of the corresponding concrete class. If the concrete class is derived from other class, the number of attributes of the base class is also added. Unlike the DET, the RET is a user-recognizable subgroup of DET within an ILF or EIF. However, since the RET cannot be counted from the class diagrams, the RET would be considered as one for each data function (concrete class) by default.

3.3 Transaction Function Identification Rules
As referred to in IFPUG 4.3.1, a transaction function is an elementary process that provides functionality to the user to process data [3]. A transaction function is an external input (EI), external output (EO), or external inquiry (EQ). An EI is an elementary process that processes data or control information sent from outside the boundary. An EO is an elementary process that sends data or control information outside the application’s boundary. An EQ is an elementary process that sends data or control information outside the boundary. An elementary process is the smallest unit of activity that is meaningful to the user, and constitutes a complete transaction.
Based on the definitions of the transaction Function [3], we propose the following stepwise rules to extract the transaction functions from the sequence and class diagrams, and the data functions of class diagrams, as in the figure 5.
Step1. Search candidates of elementary processes.

From the concrete classes of the class diagrams (figure 2), the candidate elementary processes are regarded as the concrete methods if they satisfy all of the following conditions:

- The methods must always be declared as public.
- The fan-in metric for each method must always be zero.
- The method must always be written as the argument and/or return.

Step2. Identify transaction functions.

Among candidates for elementary processes, all elementary processes if they satisfy the following condition are identified as the transaction function:

- The elementary processes obtained from the sequence diagram have to maintain and/or reference objects specified as the data functions in sequence diagrams.

The sequence diagram in Figure 3 illustrates the elementary process through which a transaction is performed.

Step3. Classify the transaction function types.

The transaction function types are determined by both the boundary of elementary process and the boundary for objects specified as ILFs within the elementary process using sequence diagrams (figure 3): external input (EI), external output (EO), and external inquiry (EQ).

First, the boundary of elementary process indicates the border between the actor and the elementary process. The actor sends a message with arguments to the elementary process and/or receives a message with return value from the elementary process. However, if no arguments and no return value are given, the elementary process is not considered as a transaction function, such as provided in Table 2.

Next, the boundary for objects indicates the border between the elementary process and the ILF. The ILF is maintained and/or referenced by each elementary process. However, if ILF is not maintained and referenced by each elementary process, the elementary process is not considered as a transaction function, such as provided in Table 2.

The transaction function types are determined with a two-dimensional index using the boundary of elementary process (EP) and the boundary for objects in accordance with Table 2.

There are therefore 16 cases to consider as in Table 2. Of the 16 possibilities,

- EI is classified as three patterns
- EO is classified as three patterns
- EQ is classified as only one pattern
- The others are N/A
Step 4. Identify complexity of transaction functions.

The complexity of EI, EO and EQ is determined by the data element type (DET) and the File Types Referenced (FTR) (Table 1). Since the DET is a unique user recognizable, non-repeated attribute [3], DET for EI is the total number of the argument of the methods called by the boundary of elementary process and DET for EO or EQ is the total number of the return of the methods called by the boundary of elementary process. If the return value is a class, then the number of attributes of the class is also counted.

Unlike the DET, the FTR is the number of unique data function read and/or maintained by a transaction function.

### 4 CONCLUSION

Function point analysis permits us to measure the functional size of a software development project and a software application. It can also be used to present the functional size of changes to a software application. Knowing the functional size allows many other useful metrics to be determined: productivity, quality, financial and maintenance.

In this paper, we introduced a new kind of UML-to-FP mapping in order to automatically measure the FP from the combination of class and sequence diagrams only generated from Java programs, such as shown in the Figure 1, 4, 5 and Table 2. This work is a first step towards the automatic improvement of the counting of the function points from UML class and sequence diagrams only based on the IFPUG 4.3.1.

The next step to be done would involve measuring in the large software Java applications and relating these measures to calculate financial and maintenance using the functional size.

### REFERENCES


