Using 'USDP Functional Model' in Function Point Analysis

Iwona Dubielewicz

Bogumiła Hnatkowska
Informatics Department, Technical University of Wrocław, Wrocław, Poland

Abstract

This paper demonstrates the function points (FP) counting for the information system model using Albrecht's and MKII FP analysis, and supports FP finding with a brief synopsis of the Unified Software Development Process (USDP) presentation. Also included is a FP case study performed for the USDP "functional model" using the simple computerized system of an electrical kiln for firing clay pots.

1. Introduction

It is a well-known fact that the adage, "time is money" holds true within our corporate culture. However, it is those key factors, time and money, that are always under estimated in determining the viability of developing software. They are the basis upon which all requirements are mapped. The more complex the system, the more these parameters are affected. Precision in making time/cost estimates influences the overall sizing and accuracy when mapping the requirements to the project plan.

In 1979, A. Albrecht proposed function points (FP) as the measure for sizing the development of software. At that time, his approach, called Function Point Analysis (FPA), did not get wide acceptance. Today, its benefits are highly recognized, the most important being the technique in which it quantifies the functions (contained within the software) from the user point of view. It is easy to use as it considers only data structures that are the foundation on which system

functions work. Moreover, FP measure is versatile, and can be applied to assessing efforts in other business events, such as productivity.

FP analysis is independent of the technology used for developing any application. Additionally, it can be used early in the software life cycle. It is recommended to count these points once the requirements have been finalized. This is advantageous because the sooner it is used to quantify time and cost, the sooner the project can be controlled.

Thus far, FPA has been applied mainly to the business system developed with structured approaches. However, now that the Object Oriented (O-O) approach has become more popular, a new reality must be proposed on how to conduct the FP analysis. Therefore, the objective of this paper is to show how function point approach can be used to size software functionality during the modeling stage in the USDP.

The first part of this analysis provides a short presentation of FP techniques and USDP methodology. The notion of the functional model is described, along with its interpretation and construction in the USDP environment. The process of creating a functional model according to USDP guidance is illustrated using a simple example of a "soft" real-time computerized system.

The second part of this analysis contains the case study. This study presents applying sizing methods for a system, using standard (Albrecht's) FP counting and MKII FP counting. Additionally, the possibility of functional sizing as proposed by COSMIConsortium [5], is briefly discussed.

The conclusion summarizes the obtained results.

2. FP analysis characteristic

The FP analysis uses the algorithmic model based on counting the 'function points'.

The starting point of construction of this model in Albrecht's approach is to determine a number of items (also named 'functions') occurring in the system:

Data function (types)

External files (EIF) – pertain to all machine readable interfaces to other systems

Internal files (ILF) – the master (logical) files in the system

Transaction function (types)

External inputs (EI) – the inputs from the user that provide distinct application oriented data; they change the values of system data

External outputs (EO) – the data directed to the user (messages, reports)

User inquiries (EQ) – interactive inputs requiring the response; they do not change any data in the system

Additionally, to accommodate the various level of complexity of each function it has been introduced the subjective complexity three-level rating: simple, average, complex; these levels are mapped on numerical scale as is given in Table 1.

Table 1. Rating for function of FP analysis

Function	**Simple**	**Average**	**Complex**
External interface files (EIF)	7	10	15
Internal logical files (ILF)	5	7	10
External inputs (EI)	3	4	6
External outputs (EO)	4	5	7
User (external) inquiries (EQ)	3	4	6

In the original Albrecht's approach the Unadjusted Function Count (UFP) is treated as a size measure of software functionality and is determined as:

$$UFC = \Sigma function_i * w_i \quad (1)$$

where w_i is the score taken from the Table 1 for each function (after fixing the level rate for the function).

To fix the level rate the other tables [2] should be used. There are various tables for each function category, which gives the rate that depends on the numbers of data (element) types (DET) and record types (REC) – for ILF, EIF – or on the numbers of record types and file referenced (REF) – for EI, EO, EQ functions.

Albrecht proposed also to count the (adjusted) Function Points FP, defined as *FP=UFC*TFC*, where *TFC* is the so-called technical complexity factor. This metrics considers the nonfunctional user requirements and its computation is done using the experimentally derived formula which includes the 14 technical factors contributing to the overall notion of complexity. This counting is often criticized as being very subjective. The same opinion is presented in [4].

MKII FP proposed by Symones, 1991 [5], is another approach, which has gained much popularity in counting function points. Symones has defined the functional size of the application as the sum of sizes of each Logical Transaction understood as the lowest level of self-consistent process. Functional size of a Logical Transaction is the weighted sum of its three components i.e. input, processing and output, and the formula used for function point counting is as follows:

$$FP\ Index = w_o * OutDET + w_i * InDET + w_e * EnR \quad (2)$$

where w_o, w_i, w_e are the industry-calibrated weights equal to 0.58 (for per Input Data Element Type – *InDET*), 1.66 (for Entity Type Referenced – *EnR*), 0.26 (for Output Data Element Type – *OutDET*).

Both, Albrecht's as well as Symones propositions are often criticized because they do not accommodate other features of software e.g. complexity of algorithms, processing of control signals, etc. To pass through these drawbacks some practitioners have proposed the other measures (feature points, 3D, etc.), but they have resulted in inconsistencies among various sizing methods (N.B. none of them has gained much popularity).

To overcome these different interpretations of the original idea, the new concept called Functional Size Measurement (FSM) was introduced and some fundamentals of it has been defined in ISO 14143 [3]. This concept should be treated as some kind of generalization of the aforementioned approaches. The ISO standard introduces the notion Base Function Component which is understood as elementary unit of functional user requirements defined by and used by an FSM method. The newest proposal of Functional Size measurement based on this conception has been formulated by COSMIC team [6]. They proposed to measure functional size at the level of "Data Movement Type (DMT)"and this size expressed in DMT is as follows:

$$DMT = N * (no.\ DET's) \tag{3}$$

where N=1 and *no. DET's* is the number of unique Data Element Types on Data Entry or Read, which have to be processed, or on Data Exit or Write – of the obtained results. This counting is done for each transaction. Identification of the logical movement types as Entries, Exits (in/from process), Read and Write (from/ to data files) is relatively simple and unambiguous in functional user requi–rements.

As one can easy notice all described methods have used (with subtle differences) three basic notions in the sense like these:

(logical) transaction – describes user interaction with a system and allows user input data or control information, which can: change data stores in the system, retrieve data stored in the system or send control information to the outside world;

logical files / entities – represent data stores the contents of which can be access using the proper transactions (store or change or retrieve);

data element type – uniquely processed item of information that is indivisible to the transaction.

3. Methodology characteristic

Unified System Development Process (USDP) is a methodology recommended for the object-oriented approach to the information system development. It has been elaborated as the response to the demands which could be used for the specification and designing of the software-intensive systems. It has been defined

by Rambaugh, Jacobson and Booch, so it is based on concepts, ideas and notations used in their very popular object-oriented methodologies.

USDP uses 7 kinds of models to describe modeled system from different point of view and at a given level of abstraction, i.e. use-case model, analysis model, design model, deployment model, component model, implementation model and test model.

The first two models are the most important as they constitute (form) the 'functional environment' necessary for counting the function points. The use-case model contains actors and use-cases. It represents external view of a modeled system. The analysis model is a "conceptual realization" of the use-case model and it is described in terms of collaboration groups. Each collaboration group is presented by set of interaction diagrams. The other models are out of our interests as they are the results of later workflows of software development.

Analysis model and use-case model include all the elements required for functional sizing, i.e. transaction description, and entity classes with domain attributes. It appears that these two models are detailed enough to support the basis for 'first-cut' estimation of application functional size. These two models are assumed to be the "USDP functional model".

4. Counting function points in USDP – case study

In the USDP, the modeling process begins with the requirements specification expressed by the use case model (based on the problem description in natural language as presented below) [4]. This specification is often accompanied with the vocabulary containing the definitions of the main notions and objects encountered in the problem (part of which is presented in the domain model).

4.1. Problem specification

The computerized system is needed to support the managing of an electrical kiln for clay pots firing. The kiln consists of n heaters, ventilator and automatically moved conveyor belt on which pots are set. The temperature of the kiln is controlled by a programmable regulator and read by *k* temperature sensors. When the kiln starts or ends the heating, belt is pull by an electrical engine [1].

There are several types of firing, each with a fixed duration and an expected temperature profile. During a firing the temperature of the kiln is automatically recorded at regular intervals and the temperature readings are displayed graphically. The temperature readings associated with earlier firing may be dis-

played at any time. During a firing, if the actual temperature differs from the expected temperature by more than a pre-set amount an audible alarm will sound (by the regulator) to alert the operator.

The system may be used to pre-set a program of future firings. The schedule of the firings may be run or stopped at any time (a current firing must be finished). An Administrator is a special kind of operator and can modify firing types, test the kiln and the engine.

Therefore, the functional model in USDP, from the perspective of the Operator user has the features presented in Fig. 1 and Fig. 3 The functions the operator can perform are Schedule Firing and Manage Schedule (Fig.1). Figure 2 with a class diagram shows static aspect of a system.

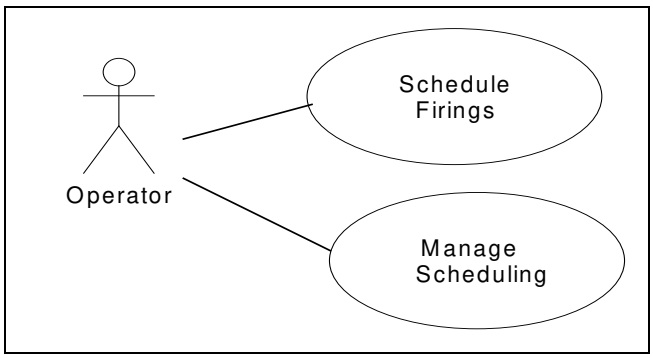

Figure 1: Fragment of a use-case model

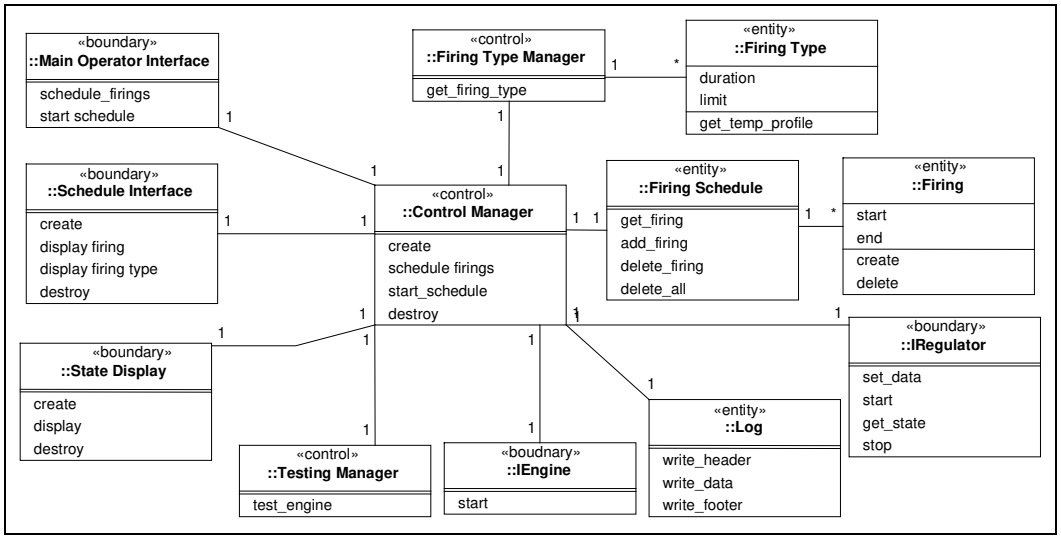

Figure 2: Operator view class diagram

Only the realization of the Schedule Firing function is presented (Fig. 3); it has been chosen as the functionality of this use-case is dominated by the operations of data storage and retrieval.

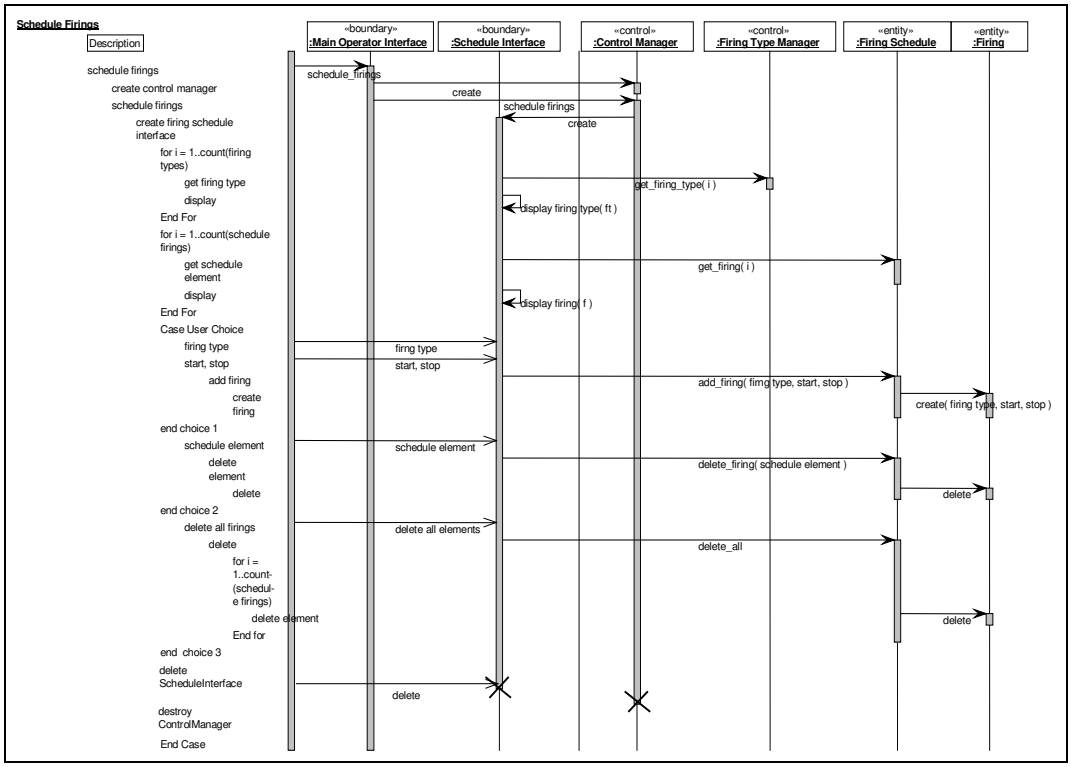

Figure 3: Sequence diagram for Schedule Firings use case

4.2. FP Counting

FP counting has different stages depending on the chosen approach. But to apply any of them, the transactions (business functions) should be identified

In the UML a use case is a specialization of classifier, so it can provide a set of operations. Analyzing scenario (Fig.3) for the Schedule Firings one can recognize that it consists of three operations: add firing, delete firing, delete all firings. They all can be treated as transactions.

But any of these transactions can have also internal structure, inner operations, hence some of them can also be treated as transactions:

add a firing

+ display firing type (*External Output*)

+ display data of firings schedule (*External Output*)

+ choose firing type (*External Inquiry*)

+ input stop and end time for the new firing (*External Input*)

+ create and schedule a new firing (*External Input*)

delete a given firing

+ display firing type (*External Output*)

+ display data of firings schedule (*External Output*)

+ choose a firing (*External Inquiry*)

+ delete a firing (*Internal Logical File*)

delete all firings

+ display firing type (*External Output*)

+ display data of firings schedule (*External Output*)

+ delete all previously scheduled firings (*Internal Logical File*)

4.2.1 Albrecht's approach to size software

To obtain FP index in Albrecht's approach the following activities are recommended:

1. Files (ILF and EIF) identification – using USDP analysis model it is enough to investigate class diagram. The classes with stereotype <<entity>> pretend to be Internal Logical Files, and some of those with <<boundary>> stereotype can be classified as External Interface Files; for our example ILF=2 (Firing Type and Firing) and EIF=0 (Regulator and Engine are rather the actors than data store).
2. Function (transaction – EQ, EI, EO) identification – in USDP analysis models they can be obtained by investigation of sequence diagrams for each use case; for our example the identified functions and its categories are shown in 4.2; there are 2 inquiry functions, two input functions and 5 output functions.
3. *Counting UFP* – Functional Points can be counted in two ways.

The first approach, rough, is as follows. Complexity rate is assumed for system as the whole – it means that one complexity rate is assigned to all system functions – and the system complexity is counted by multiplying value (taking from Table 1) by the number of the function of each category; for use cases described in 4.1. When the modeled system is treated as that of the average complexity, UFP is equal to:

$$UFP = 4*2_{EQ} + 4*2_{EI} + 5*2_{EO} + 10*0_{EIF} + 7*2_{ILF} = 40 \quad (4)$$

Such counting is done when no details concerning system structure and its data are given.

The second, more precise way of counting follows the standard procedure. FP is determined independently for each system function. Each function requires the identification of the data element types and record types it uses, and the files it referenced to. When using USDP this identification is easy to performe as the class attributes (DET), messages/operation parameters (DET, REC), and classes

(REF elements) are defined in the analysis model. Level rate, presented for each function in [2] allows to determine its complexity (low, medium, high). Obtained complexity is then transformed in scores given in Table 1. At the end the obtained results are summed – according to formula (1); for our example:

$$UFP = 3*2_{EQ} + 4*2_{EO} + 5*2_{EI} + 2*2_{ILF} = 28 \quad (5)$$

The calculation done in that way seems to be more precise then this resulting from totaling the elements. The latter approach requires much more detailed description of system models but it can be quite easy done when using USDP.

4.2.2 MKII FP approach to size software

To obtain FP index in Symones's approach the following steps are needed:

1. Determination of counting boundary. In USDP the counting boundary is very easy for determining (classes with stereotype <<boundary>> define it).
2. Identification of the logical transaction (defined as lowest level of self-consistent process). In the USDP analysis model all messages send to and from system boundary classes, or from an actor or other inner classes, pretend to be classified as logical transaction (it is shown in 4.2). It is worth to notice that two 'camouflage' transaction has been recovered ('display firing' and 'display firing type') due to precise description of the use-case realization (in sequence diagram form). Each transaction is driven by event (transaction of E type), or by user inquiry (transaction of Q type).
3. Calculation for each transaction its input and output DET's and Entity types it references to. Entity Types can be determined after analyzing the class diagram. Classes with stereotype <<entity>> are candidates for them. It can be also easy to determine the 'non-primary' (system) entities as they have few attributes changing very rare (in the example Log class is of this category).
4. Calculation for each identified transaction its FP index – according to formula (2).

The results obtained for the Schedule Firings use case sizing are presented in Table 2 – the numbers of Input and Output DET's (placed in the table) are derived from the operation signatures specified in class description.

Table 2. Calculation of FP index for Schedule Firings use case

Transaction name	**Transaction type**	**No of Input DET**	**Response**	**No of Output DET**	**Entity Referenced to**	**No of ER**	**MKII FP**
Add Firing	E	4	OK./Error	2	FiringType, Firing	2	6.42
Delete Firing	E	1	OK./Message	2	FiringType, Firing	2	4.68
Delete all Firings	E	1	OK./Message	2	Firing	1	3.18
Display Firing Type	Q	1	FiringType/ Message	4	FiringType	1	3.80
Display Firing	Q	1	Firing/Message	5	Firing	1	4.32
Display all Firings	Q	1	List/Message	7	Firing	2	7.54
Total							31.44

5. Conclusion

The function point technique is widely accepted technique as FP analysis provides an objective, comparative measure which can be effectively used in evaluation, management (e.g. planning, control) of software production.

The paper presents the calculation of this size measure following two different approaches, i.e. Albrecht's and Symones. FP values obtained in both of methods are very near as it has been expected (according to [5] for the simple systems these two values should be similar).

As it could be seen the FP calculation is independent of environment in which application will be situated as well as of technologies used to produce this software (e.g. CASE tools). It is also independent of the type of HC interface connected to the resulting software. It also seems that FP can be easy calculated for any additional functionality and changes introduced to application when using USDP – only calculation for the added use case should be performed and summed with the previous result.

Using USDP models in the way presented in section 4, the function point measure sizes the core of application (business logic) on the high abstract level so the obtained results should be treated as very rough estimation and recalculated after design workflow is finished.

References

[1]. I. Dubielewicz, B. Hnatkowska, Applying UML to information systems modeling, *Conf. Proceedings MOSIS'99*, Czech Republic, 1999.

[2]. IFPUG , Function Point Counting Practices Manual v.4.1, www.ifpug.com

[3]. ISO/IEC DIS 14143-1:1996, Information Technology - Software Measurement - Part 1: Definition of functional size measurement

[4]. I. Jacobson,G. Booch, J. Rambaugh, *Unified Software Development Process*, Addison-Wesley, 1999.

[5]. *MKII Function Point Analysis, Counting Practices Manual v.1.3.1*, September 1998 UKSMA Metrics Practices Committee

[6]. Ch. Symones, P. Rule, One size fits all "COSMIC' – Aims, Design Principles and Progress', www.cosmicon.com