Software Size Estimation Using Function Point Analysis – A Case Study for a Mobile Application

Volkan Tunalı

Department of Software Engineering
Maltepe University
Marmara Egitim Koyu, 34758, Istanbul-Turkey
volkantunali@maltepe.edu.tr

***Abstract* - A good planning is essential for a software project to be successful, and a good estimation of the size of the software to be developed is critical for a good planning. There are many methods used for size estimation of software projects. One of the most widely known and used methods is the Function Point Analysis (FPA). In this study, size of a mobile software project was estimated using FPA method. The estimate was compared to the actual size of the project after development, and results were presented.**

Index Terms – software size estimation, function poins analysis, mobile software.

I. Introduction

A key factor for the success of projects including software projects is the accurate estimation and planning of activities to be performed, anticipating time, budget, and quality constraints. Estimation and planning in software projects can be carried out by first measuring the size of the deliverables to be created or developed [1].

One of the most widely used method for measuring the size of the estimated software system is the Function Point Analysis (FPA). FPA was first introduced by Allan Albrecht in 1979, and now it is kept updated by the International Function Point User Group (IFPUG) [2, 3]. FPA is based on the amount of functionality in a software project and a set of individual project factors. FPA is a standard method for measuring software development from the user's point of view. In the literature, there are other several widely recognized methods used for estimating the size of software projects based on the function point approach. Some of them are Mark II FPA [4], COSMIC FFP [5], NESMA [6], and FiSMA [7]. These methods are also recognized ISO standards for functionally sizing software along with the IFPUG method as of 2012.

In this study, FPA was used to estimate the size of a mobile software project developed by the author. In Section II, FPA method is explained in some detail. Section III describes the properties of the application developed, and the estimation of the project size using the FPA method. The estimated size is compared with the actual project size. Finally, Section IV contains some conclusions and future work.

II. Function Point Analysis

FPA is a method for measuring the size of a software system in terms of the amount of functionality and complexity in the software system from the user's perspective as seen in Fig. 1.

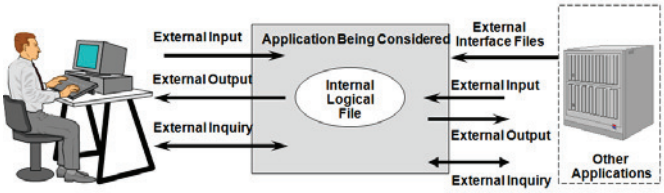


Fig. 1 Functionality as viewed from the user's perspective [3].

Established in 1986, IFPUG is a nonprofit organization which promotes FPA as its standard methodology for software size estimation. IFPUG also maintains the Function Point Counting Practices Manual (CPM) which is the recognized industry standard for FPA [8]. The standard counting procedure described in CPM is shown in Fig. 2.

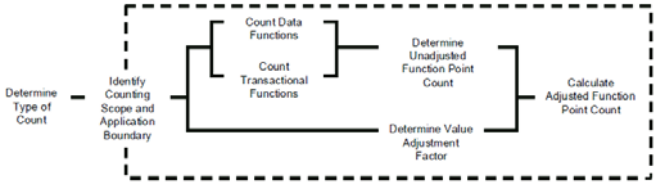


Fig. 2 IFPUG function point counting procedure diagram [8].

A. Identification of User Function Types

In FPA, a software system consists of five major components or types that provide information processing functionality to the users. In the FPA procedure, these five user function types are identified as below [3, 9]:

- *External Input (EI)*: user data or user control input type that enters the external boundary of the software system being measured
- *External Output (EO)*: user data or control output type that leaves the external boundary of the software system being measured
- *Internal Logical File (ILF)*: logical file types that are generated, used, or maintained by the software system
- *External Interface File (EIF)*: file types passed or shared between software systems
- *External Inquiry (EQ)*: input-output combination, where input causes and generates an immediate output.

B. Complexity Classification of User Function Types

After the identification of the instances of each user function type in the software system being measured, the instances are classified according to their complexity levels as either high, average, or low complexity. The counts of user function type in each complexity category are then multiplied by the weights specified in Table I.

TABLE I
COMPLEXITY WEIGHTS IN FPA [9]

User Function Type	Complexity Weight		
	Low	Average	High
External Input (EI)	3	4	6
External Output (EO)	4	5	7
Internal Logical File (ILF)	7	10	15
External Interface File (EIF)	5	7	10
External Inquiry (EQ)	3	4	6

With the original FPA as defined by Albrecht, classification of user function types according to their complexities was intuitive and subjective. In order to make the complexity evaluation less subjective, IFPUG developed complexity matrices for each feature that considers the number of file types, record types, and/or data element types. For example, Table II shows the matrix dealing with the complexity of internal logical files. Similar tables exist for all other user function types.

TABLE II
IFPUG INTERNAL LOGICAL FILE TYPE COMPLEXITY MATRIX [9]

Number of Record Types	Number of Data Types		
	< 20	20-50	>50
1	Low	Low	Average
2 to 5	Low	Average	High
> 5	Average	High	High

C. Calculation of Unadjusted Function Point

Once the complexity level of each user function type is identified, it is possible to compute the Unadjusted Function Point (UFP) value using the formula in (1).

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} x_{ij} \times w_{ij} \quad (1)$$

The UFP is calculated as the weighted total of the number of user function types where x_{ij} is the number of user function type i with complexity level j, and w_{ij} is the weight value for type i with complexity level j.

D. Calculation of Technical Complexity Adjustment

In FPA, the effort required to implement a software system is related not just to the number and complexity of the features provided to the user but also to the operational environment of the system [1]. Thus, there are 14 factors identified which can influence the degree of difficulty associated with implementing the system. These factors are called *value adjustments*, and they are listed in Table III [9].

Each factor is assigned a value ranging from 0 to 5 (0 if the factor has no impact at all on the application; 5 if the factor has a strong and pervasive impact). When all of the 14 factors are considered and scores assigned individually, Total Degree of Influence (TDI) is simply the sum of 14 factors. Then, TDI is converted into a final Technical Complexity Adjustment using the formula in (2).

$$TCA = 0.65 + 0.01 \times TDI \quad (2)$$

TABLE III
VALUE ADJUSTMENT FACTORS [9]

ID	System Characteristic
C1	Data communications
C2	Distributed functions
C3	Performance objectives
C4	Heavily used configuration
C5	Transaction rate
C6	On-line data entry
C7	End-user efficiency
C8	On-line update
C9	Complex processing
C10	Reusability
C11	Installation ease
C12	Operational ease
C13	Multiple sites
C14	Facilitate change

E. Calculation of Function Point and Size Estimate

Function Point (FP) value of the software system is calculated by multiplying the UFP by the TCA as in (3).

$$FP = UFP \times TCA \quad (3)$$

In order to obtain the size estimate of the software system in terms of Lines of Code (LOC), the FP value is multiplied by the productivity factor of the programming language that will be used for implementing the software. Productivity factor is the number of logical code statements per function point, and it varies among the programming languages. For example, productivity factor of Java language is 53 [9].

III. CASE STUDY

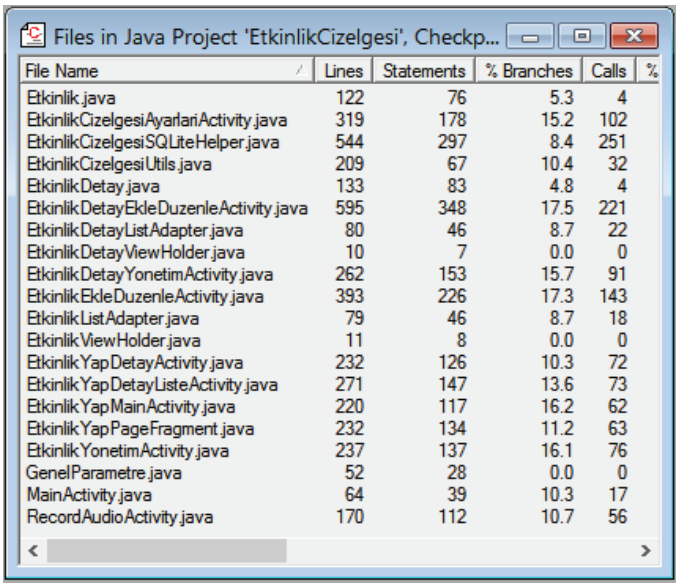
A. The Mobile Application: Activity Schedule

In this study, a mobile application developed by the author was addressed as a case study. The Activity Schedule application was designed and developed specifically to run on tablet computers for teaching children with autism to use activity schedules. Activity schedules are very important visual support systems for children with autism to develop skills like getting organized, acting independently, and making selections.

The application was developed with Java programming language using the Eclipse ADT development environment, and it runs on mobile devices with Android operating system.

Actual total size of the project considering only the Java source code files is 4235 lines in 20 files. Comment lines

inside the code are intentionally included in this count because meaningful comments are as importans as the actual code lines regarding the management of complexity in software development. Fig. 3 displays some metrics of the code files, collected with the Source Monitor software [10].



File Name	Lines	Statements	% Branches	Calls	%
Etkinlik.java	122	76	5.3	4	
EtkinlikCizelgesiAyarlariActivity.java	319	178	15.2	102	
EtkinlikCizelgesiSQLiteHelper.java	544	297	8.4	251	
EtkinlikCizelgesiUtils.java	209	67	10.4	32	
EtkinlikDetay.java	133	83	4.8	4	
EtkinlikDetayEkleDuzenleActivity.java	595	348	17.5	221	
EtkinlikDetayListAdapter.java	80	46	8.7	22	
EtkinlikDetayViewHolder.java	10	7	0.0	0	
EtkinlikDetayYonetimActivity.java	262	153	15.7	91	
EtkinlikEkleDuzenleActivity.java	393	226	17.3	143	
EtkinlikListAdapter.java	79	46	8.7	18	
EtkinlikViewHolder.java	11	8	0.0	0	
EtkinlikYapDetayActivity.java	232	126	10.3	72	
EtkinlikYapDetayListeActivity.java	271	147	13.6	73	
EtkinlikYapMainActivity.java	220	117	16.2	62	
EtkinlikYapPageFragment.java	232	134	11.2	63	
EtkinlikYonetimActivity.java	237	137	16.1	76	
GenelParametre.java	52	28	0.0	0	
MainActivity.java	64	39	10.3	17	
RecordAudioActivity.java	170	112	10.7	56	

Fig. 3 Some metrics collected from the code files of the project.

B. *Application of FPA to the Project*

First of all, user function types of the project were identified and classified according to their complexity levels. There are 15 External Inputs, 3 Internal Logical Files, and 2 External Interface Files identified in this project. These are listed with their complexity levels in Tables IV, V, and VI respectively.

TABLE IV
EXTERNAL INPUTS

No	Description	Complexity
1	Main app screen	Low
2	Activity management main screen	Low
3	Activity steps management main screen	Low
4	Doing activity main screen	Low
5	General app configuration screen	Average
6	Taking activity picture feature	Average
7	Choosing picture from gallery for activity	Average
8	Taking activity step picture feature	Average
9	Choosing picture from gallery for activity step	Average
10	Recording activity step audio feature	Average
11	Choosing audio from gallery for activity step	Average
12	Add/edit activity screen	Average
13	Add/edit activity step screen	Average
14	Doing activity screen with picture and sliding	Average
15	Doing activity screen with step list only	Average

TABLE V
INTERNAL LOGICAL FILES

No	Description	Complexity
1	Database table ETKINLIK	Low
2	Database table ETKINLIK_DETAY	Low
3	Database table GENEL_PARAMETRE	Low

TABLE VI
EXTERNAL INTERFACE FILES

No	Description	Complexity
1	Database, picture, and audio file export feature	Low
2	Database, picture, and audio file import feature	Low

Using the user function type count and complexity data, the UFP value was calculated as 87. Calculation of the UFP value can be seen in Table VII.

TABLE VII
CALCULATION OF THE UFP VALUE

User Function Type	Complexity Weight x Counts			
	Low	Average	High	Total
External Input	3x4=12	4x11=44	6x0=0	56
External Output	4x0=0	5x0=0	7x0=0	0
Internal Logical File	7x3=21	10x0=0	15x0=0	21
External Interface File	5x2=10	7x0=0	10x0=0	10
External Inquiry	3x0=0	4x0=0	6x0=0	0
			Unadjusted Function Point	87

After calculating the UFP, first the TDI value was calculated as 28 using the value adjustment factors as seen in Table VIII.

TABLE VIII
CALCULATION OF THE TDI VALUE

ID	System Characteristic	DI
C1	Data communications	0
C2	Distributed functions	0
C3	Performance objectives	0
C4	Heavily used configuration	3
C5	Transaction rate	0
C6	On-line data entry	4
C7	End-user efficiency	5
C8	On-line update	5
C9	Complex processing	2
C10	Reusability	0
C11	Installation ease	0
C12	Operational ease	4
C13	Multiple sites	0
C14	Facilitate change	5
	Total Degree of Influence	28

Calculated TDI value was substituted in (2), and then the TCA value was calculated as 0.93. Finally, the FP of the system was calculated by multiplying the UFP by TCA as in (4).

$$FP = 87 \times 0.93 = 80.91 \quad (4)$$

In order to obtain the size estimate of the system in terms of source code lines, the FP value was multiplied by 53 which is the productivity factor of Java programming language [9] as in (5).

$$Size\_in\_LOC = 80.91 \times 53 \cong 4288 \quad (5)$$

Size of the project was estimated about 4288 lines using the FPA method. After the completion of the project, the actual size of the project was 4235 lines. When the estimated size is compared to the actual size, the deviation is about 1.2%. This was a highly successful estimation. However, there could be a bigger difference between the estimate and the actual size because correct identification of the function types was a difficult process, and complexity classification of the identified function types was a subjective issue although IFPUG provided general guidelines for complexity classification. Therefore, it would be good approach to anticipate a difference between the estimate and the actual size to some extent.

IV. Conclusion and Future Work

In this study, Function Point Analysis software size estimation method was explained in some detail. The method was applied on a mobile tablet computer application developed by the author, and results were evaluated. Applications developed for mobile devices like tablets and smart phones are usually small in size and functionality when compared to conventional software projects like information systems. It is seen that the FPA method produces a very accurate estimate even for such a small scale mobile application. As a future work, in addition to function point based approaches, estimation accuracy of size estimation methods based on class count and use case count like Class Point [11] and Use Case Point [12] can be investigated on mobile software projects.

References

[1] B. Hughes and M. Cotterell, *Software Project Management*. London: McGraw-Hill Higher Education, 2009.

[2] A. J. Albrecht, "Measuring Application Development Productivity," in *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, Monterey, California, 1979, pp. 83-92.

[3] (2014, February). *IFPUG - International Function Point Users Group*. Available: http://www.ifpug.org

[4] C. R. Symons, *Software Sizing and Estimating: Mark II FPA*. New York, NY, USA: John Wiley and Sons, 1991.

[5] A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre, and C. Symons, *COSMIC-FFP Measurement Manual, Version 2.1*. Université du Québec, 2001.

[6] (2014, February). *NESMA - Netherlands Software Metrics Users Association*. Available: http://www.nesma.nl

[7] (2014, February). *FiSMA - Finnish Software Measurement Association*. Available: http://www.fisma.fi

[8] IFPUG, *Function Point Counting Practices Manual Release 4.1.1*, 2000.

[9] C. Jones, *Applied Software Measurement, Global Analysis of Productivity and Quality*, 3rd ed. New York, NY, USA: McGraw-Hill, 2008.

[10] (2014, February). *SourceMonitor Version 3.4*. Available: http://www.campwoodsw.com/sourcemonitor.html

[11] G. Costagliola and G. Tortora, "Class Point: An Approach for the Size Estimation of Object-Oriented Systems," *IEEE Transactions on Software Engineering,* vol. 21, pp. 52-74, 2005.

[12] K. Periyasamy and A. Ghode, "Cost Estimation Using Extended Use Case Point (e-UCP) Model," in *International Conference on Computational Intelligence and Software Engineering*, Wuhan, China, 2009, pp. 1-5.