American Journal of Applied Science 2 (8): 1218-1221, 2005
ISSN 1546-9239

Measuring the Function Points for Migration Project: A Case Study

[1]K. Mustafa, [2]K. Gowthaman and [2]R.A. Khan
[1]Department of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan
[2]Department of Computer Science, Jamia Millia Islamia (A Central University), New Delhi, India

Abstract: In this study, we extend the study of Function Point Analysis (FPA) for the measurement of source code based migration project using the migration tooling options. FPA is an objective and structured technique to measure software size by quantifying its functionality provided to the user, based on the requirements and logical design. It is noticed that the procedure for the above measurement is not fully described in FPA method. Hence, the same estimation concept has been applied to an in-house project on an experimental basis at a leading software development organization. The yielded results are compared with FPA for the re-engineering project.

Key words: Function point analysis, source code migration, software estimation, re-engineering

INTRODUCTION

One of the most important activities in the early stages of software development is estimation. The size of the software[1], be it Function Points or Lines of Code, plays a pivotal role in this process and forms the base for deriving a number of metrics to measure various aspects of the software throughout the development cycle. Hence, measuring the size of the software becomes critical though many other sizing measures such as objects, classes, modules, screens, programs and so on. Accurately predicting the size of the software has always troubled the software industry over the years. Function Points are becoming widely accepted as the standard metric for measuring software size.

We sketch briefly for existing software systems[2] and in particular legacy systems, where the functional documentation is often missing or obsolete. Hence, the standard Function Point Analysis (FPA) is not applicable for the sizing of enhancement projects (the implementation of change requests) as back firing is not applicable either it only refers to the complete software system and its precision is not sufficient to size individual enhancement projects. A method to perform FPA based on the source code is proposed. This method is instantiated for COBOL and JCL (Job Control Language). It can be integrated into the maintenance process such that each change request is defined in terms of the objects from the Function Point (FP) conceptual model. In this way, the sizing of a change request is obtained for free.

It is demonstrated that simplified way of the IFPUG (International Function Point Users Group) function points based on the simplification ideas suggested by NESMA (Netherlands Software Metrics Association) to estimate the size of management information systems[3]. He analyzed nearly twenty web applications using the simplified method, whose result was very close to the ones using the IFPUG detailed method. The simplified method is based on assigned low complexity to all the data and transactional functions. Thus, when data and transactional functions are identified, their complexity is determined automatically for Web based applications[4].

Function points analysis: Function Point Analysis is a well-known method to estimate the size of software systems and software projects[5]. This technique breaks the system into smaller components so that they can be better understood and analyzed. The function point count can be applied to development projects, enhancement projects and existing applications as well. Function Point Analysis is expected to obtain the following objectives[6]:

* Determine the type of Function Point count.
* Determine the application boundary
* Identify and rate transactional function types to calculate their contribution to the Unadjusted Function Point count (UFP).
* Identify and rate the data function types to calculate their contribution to the UFP.
* Determine the Value Adjustment Factor (VAF) by using General System Characteristics (GSCs).
* Finally, calculate the adjusted Function Point count.

There are 5 major components of Function Point Analysis which capture the functionality of the application. These are External Inputs (EIs), External Outputs (EOs), External Inquiries (EQs), Internal Logical Files (ILFs) and External Interface Files (EIFs). First three are treated as Transactional Function Types and the last two are called Data Function Types. Each

Corresponding Author: K. Mustafa, Department of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan

of the components of Function Point Analysis is explained in brief in the following sub-sections[6-8].

External input (EI): External Input is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application. The data may be used to maintain one or more internal logical file. The data can be either control information or business information.

External output (EO): External Output is an elementary process in which derived data passes through the boundary from inside to outside. Additionally, an EO may update an internal logical file. The data creates reports or output files sent to other applications. These reports and files are created from information contained in one or more internal logical files and external interface files. The derived data is processed beyond direct retrieval and editing of information from internal logical files or external interface files.

External inquiry (EQ): External Inquiry is an elementary process with both input and output components that results in data retrieval from one or more internal logical files and external interface files. The input process does not update or maintain any FTRs (Internal Logical Files or External Interface Files) and the output side does not contain derived data.

Internal logical file (ILF): Internal Logical File is a user identifiable group of logically related data that resides entirely within the application boundary and is maintained through External Inputs. Even though it is not a rule, at least one external output and/or external inquiry should include the ILF as an FTR.

External interface file (EIF): External Interface File is a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application boundary and is maintained by external inputs of another application. In other words, the external interface file is an internal logical file in another application. At least one transaction, external input, external output or external inquiry should include the EIF as a File Type Referenced.

This FPA model, developed by Albrecht[9,10] is widely used in the software industry. The model estimates software size by counting "function points". This is done using three steps[11]:

Step 1: Computing an Unadjusted Function Count (UFC): Using the above five types of components (EI, EO, EQ, ILF and EIF), the number of items in the system is counted and the level of complexity is determined (distinguishing between simple, medium and complex). Thus the total number of items are 5 components multiply with 3 levels of complexity. Each level has a weight (provided by the model). The UFC is computed as follows[9-11]:

$$UFC = \sum_{i=1}^{15} (\text{No.of items of types i}) * (\text{weight i})$$

Step 2: Value Adjustment Factor (VAF): The value adjustment factor (VAF) is calculated based on 14 General System Characteristics (GSC) that rate the general functionality of the application being counted. The 14 General System Characteristics are: Data communications, Distributed data processing, Performance, Heavily used configuration, Transaction rate, On-line data entry, End-user efficiency, On-line update, Complex processing, Reusability, Installation ease, Operational ease, multiple sites and Facilitate change. The degree of influence of each characteristic has to be determined as a rating on a scale of 0 to 5 as defined below.

0: Not present, or no influence
1: Incidental influence
2: Moderate influence
3: Average influence
4: Significant influence
5: Strong influence throughout

Once all the GSCs have been rated, Total Degrees of Influence (TDI) are obtained by summing up all the ratings[11].

$$TDI = \sum_{I=1}^{14} F_i$$

where, F_i is the weight of attribute I

Now, Value Adjustment Factor is calculated using the formula[10]:

$$VAF = 0.65 + TDI/100$$

Step 3: After determining the Unadjusted Function Point count (UFP) out of transactions and data function types and calculating the Value Adjustment Factor (VAF) by rating the general system characteristics, the final Function Point count can be calculated using the formula[7,8,10]:

$$FP = UFC * VAF$$

To use this model, the software development organization has to maintain a database of its projects,

including duration, cost, manpower effort and function points (FE). Based on this database, the cost (in terms of time and money) of one FP can be computed. Henceforth, the FP of any new project can be computed as described above and the cost estimates derived.

Re-engineering case study: To measure the Function Point for re-engineering application, we apply the FPA method for the in-house re-engineering project at one of the leading software organizations. Identity is not disclosed honoring the industry sentiments.

About the project: The Project is an application namely Resource Requirement Form (RRF) which is basically developed for associates who are working for the organization. The Project has the following features:

* Each request form has a unique ID, which is used for tracking and status viewing.
* It automates all functions of the workflow pertaining to hardware and software installation processing.
* Generates automatic mail between users to draw their attention.
* Approving Authority's name, time and date are endorsed from the terminal.
* When the form is accepted /submitted by higher authorities, the associate receives feedback on e-mail with date and time of the approval of the request.
* E-mail notification is sent at every transition of the flow of the process.

Table 1: FP count for normal re-engineering process (without using migration tool)

Function Type	Complexity Functional	Function Total	Type Total
ILFs	2 Low x7	14	49
	2 Avg x10	20	
	1high x15	15	
EIFs	2 Low x5	10	51
	3 Avg x7	21	
	1 high x10	10	
EIs	10 Low x3	30	52
	4 Avg x4	16	
	1 high x6	6	
Eos	5 Low x4	20	44
	2 Avg x5	10	
	1 high x7	14	
EQs	6 Low x3	18	36
	3 Avg x4	12	
	1 high x6	6	
Total UFC			232
Total TDI			53
VAF			1.18
Total Adjusted FP			273
Source code based FP count using MM^{LC}			

Table 2: Migration tool based FP count using MM^{LC}

Function Type	Complexity Functional	Function Total	Type Total
ILFs	3 Low x 7	21	56
	2 Avg x10	20	
	1high x15	15	
EIFs	3 Low x5	15	32
	1 Avg x7	7	
	1 high x10	10	
Eis	6 Low x3	18	44
	5 Avg x4	20	
	1 high x6	6	
Eos	5 Low x4	20	51
	2 Avg x5	10	
	3 high x7	21	
EQs	3 Low x3	9	27
	3 Avg x4	12	
	1 high x6	6	
Total UFC			210
Total TDI			58
VAF			1.23
Total Adjusted FP			258.3

The Project was developed in VB, ASP with SQL server environment and is being used successfully by the associates. Over the period, the higher authorities decided to redesign and redevelop this in .Net environment with some additional requirements. We measured the FP as per given requirements documents. Detail is given in Table 1.

We also estimate the FP count for the same project which was developed using Migration Model for Legacy Source (MM^{LC})[12]. Using this model, the legacy source codes are loaded into. Net migration tool. The tool converts 45 % of source code into the target environment successfully. A migration tool cannot resolve the remaining compatibility issue and the developer need to fix the issues[13]. The developer needs to pay attention to the following activities in this regard:

1. The development team will have to rewrite the code before using the migration tool as and when the target environment does not support certain features[14].
2. The duplicate code removal, implementing coding standards, upgrading the problematic syntax and controls should be done[15].
3. Once the migration tool is loaded, the development team needs to take care of certain editing like syntax changes, changes in control object models, unsupported functions requiring complete redesign, etc.[16].
4. Another important factor is knowledge transfer related to Focus on Future Re-engineering (FFR). In this stage, the developer, should share knowledge and train the co-developers for transparency leading to be better understanding and an accurate and fast achievement of the desired objectives of the project.

The FP count as per MM^{LC} is given in Table 2.

DISCUSSION

Variances of Functional Count of normal re-engineering process and in using migration model were very less. To be advantageous for the customer

Function Points can be used to help specify to a vendor the key deliverables, so as to ensure that appropriate levels of functionality will be delivered and to develop objective measures of cost-effectiveness and quality. They are most effectively used with fixed price contracts as a means of specifying exactly what will be delivered. From a vendor perspective, successful management of fixed price contracts depends absolutely on accurate representations of effort. Function Points offer a vast number of benefits by capturing the size of the software from its functionality standpoint. FPA does have some disadvantages as follows, Kitchenham[17].

Developers' experience: Implementation of a specific logic differs based on the level of experience of the developer. Hence, the number of lines of code differs from person to person. An experienced developer may implement certain functionality in fewer lines of code than another developer with relatively less experience does, though they use the same language.

Advent of GUI tools: With the advent of GUI-based languages/tools such as Visual Basic, much of the development work is done by drag-and-drops and a few mouse clicks, where most of the time the programmer virtually writes no piece of code. It is not possible to account for the code that is automatically generated in this case. This difference invites huge variations in productivity and other metrics with respect to different languages, making the lines of code more and more irrelevant in the context of GUI-based languages/tools, which are prominent in the present software development arena.

CONCLUSION

This study has taken the critical view of Function point estimation issues with a migration tool related to re-engineering software perspective. The estimated result is close to the normal FPA estimation process, but it specifies a particular migration tool. This method must be validated by applying a number of re-engineering applications with a known FPA model. The results have to be discussed with the team of software engineers for further enhancement.

REFERENCES

1. Hastings, T.E. and A.S.M. Sajeev, 2001. A vector based approach to software size measurement and effort estimation. IEEE Trans. Software Eng., pp: 337-350.
2. Steven Klusener, 2002. Source code based function point analysis for enhancement projects. Proc. Intl. Conf. Software Engg., pp: 001-004.
3. Edilson, J.D. Candido and Rosely Sanches, 2004. Estimating the size of web applications by using a simplified function point method. Proc. Web Media and Latin American Web Congress, pp: 98-105.
4. Ruhe, M., R. Jeffery and I. Wieczorek, 2003. Using web objects for estimating software development effort for web applications. Proc. 9th Intl. Software Metrics Symp., pp:30-39.
5. Roger S. Pressman, 2001. Software Engineering: A practitioner's Approach., pp: 89-94.
6. International Function Point Users Group, 1999. Function Point Counting Practices Manual. IFPUG, 4.1.1 Edn.
7. IFPUG., 2004. International function point users group.http://www.ifpug.org. Accessed on Dec., 2004.
8. NESMA., 2005. Netherlands software metric association. http://www.nesma.org/english/index.htm. Accessed on Jan., 2005.
9. Albrecht, A.J., 1979. Measuring application development productivity. Proc. IBM Applications Development Symp., pp: 83-92
10. Albrecht, A.J and J. E. Gaffney, Software Function: Source Lines of Code and Development Effort Prediction: A Software Science Validation, IEEE Trans. Software Engineering, Nov 1983, pp. 639-648.
11. Peretz Shoval and Ofer Feldman, 1996. Combining function points software estimation model with a Odessa methodology for systems analysis and design. Proc. 7th Israeli Conf. Computer based System and Software Engg., pp: 03-08.
12. Gowthaman, K., K. Mustafa and R.A. Khan, 2005. Source code migration to DOT NET Framework: A Re-engineering application perspective. Inform. Technol. J., Vol. 4.
13. Katre, D., P. Halari, N.R. Surapaneni, M. Gupta and M. Deshpande, 2002. Migrating to .NET: A Pragmatic Path to Visual Basic .NET, Visual C++ .NET and ASP.NET. Prentice Hall PTR., pp: 35-125.
14. MSDN Library, 2000. Preparing Your Visual Basic 6.0 Applications for the Upgrade to Visual Basic .NET. Microsoft Corp., http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvb600/html/vb6tovbdotnet.asp
15. Mens, K., B. Poll and S. Gonzalez, 2003. Using intentional source-code views to aid software maintenance. Intl. Conf. Software Maintenance, pp: 169-174.
16. Bergey, J., D. Smith and N. Weiderman, 1999. DOD Legacy System Migration Guidelines. Software Engineering Institute, Technical Report: CMU/SEI-99-TN-013, pp: 2- 25.
17. Kitchenham, B., 1997. The problem with function points. IEEE Software, pp: 29-31.