Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 4; April-June, 2015 pp. 334-337

http://www.krishisanskriti.org/acsit.html

Software Metrics estimation using Object Oriented Approach

Deep Chandra Andola[1] and Harendra Pratap Singh[2]

[1,2]Amrapali Institute of Technology & Sciences, Haldwani
E-mail: [1]deep.andola@gmail.com, [2]harendra.engineer@yahoo.co.in

Abstract—*Development of a software system using Object - oriented programming is widely used approach, there are several techniques specified by different researchers and analysts for estimating the size of object-oriented software system. Unified modeling language (UML) is also used to implement an object oriented software system, for mapping UML to function point analysis different types of approaches are proposed by software practitioners, here UML diagrams are used to map UML to function point analysis. Several rules were proposed in past, those rules can be applied on UML for estimating function points. Here UML class diagram are used for data function analysis and UML sequence diagram for transaction function analysis than applied transformation rules and guidelines to estimates function point is done. A tool based estimation technique for object oriented software metrics is developed which extricate the required information from UML diagrams and its logical view is reported to estimate the size of software, then using COCOMO II rest of the software metrics like effort, development time, productivity and cost will be calculated.*

1. INTRODUCTION

As we all are well aware that in present time there is abundance of Object oriented software e.g. banking applications, military applications and in several other fields. Developing a quality, cost- effective software within the boundary of given time period is still a challenging task. In this sequence, it is required to manage the whole software development processes on the operative project plan. Therefore software development process must be incorporated with accurate estimation of varied software metrics like size, devoted effort, devoted time to complete, quality, risks factors and several resources of software. It is proved from research that size estimation should be done particularly in the early phase of the software development life cycle that is on the transformation model. There are several models for estimation efforts, cost and quality used by software practitioners and analysts. A varied variety of effort models are suggested and used and software size as a very important parameter in most of them.LOC (lines of codes) is often adopted in the above proposed effort models However, size estimation through LOC has difficulties one of them is that, the definition of LOC is imprecise and LOC is directly proportional to the programming language. Function point is a measure of software size that uses logical and functional terms, professionals and users more readily understand. The Unified Modeling Language (UML) was introduced to produce a simple language for object oriented modeling. Which was built to be extensible in a hierarchy to fulfill ample variation of needs and was also determined to be flexible of certain programming language and development techniques? In this research we have projected our efforts that is to automate this course of action is entirely in initial stages of development life cycles then COCOM II estimation techniques will be incorporated to calculates rest of the software metrics like effort, cost, development time etc.

2. BASIC SOFTWARE METRICES

Size or the bulkiness of a software system is considered as the basic metrics in software metrics model. Size can be estimated in Lines of code or function point. LOC cannot be measure properly in advance of completion of software because it varies due to programming language complexities of different programming language. In this span Function points are technologically individualistic, uniform, and repeatable, help normalize data, allow similarity and set project range and client presumption and expectation. So in such situation size is estimated regarding function point.

2.1 FUNCTION POINT

Function points measure the knowledge processing components of software systems. Function points measure the size of an application from the customer's perspective. The aspects of a software system that can be measured accurately are these:

- Inputs to the application.
- Outputs from the application.
- Inspection by the end users.
- Data files updated by the application.
- The interface to other similar type of applications.

2.2 FPA PROCESS ANALYSIS

The FPA process involves:

1. Diagnosing the function point estimating boundary. A boundary specifies the limit into the interval in which the software system being measured and the external application or the user domain. A boundary adjudges which functions are involved in the function point count.
2. Determining the unadjusted function point count (UFPC). The unadjusted function point count illustrates the exact measurable functionality produced to the user by the application.

2.3 INTERNAL LOGICAL FILES

An internal logical file (ILF) is a user recognizable cluster of associated data arranged within the limits of the application. An ILF is a cluster of data that is arranged within the application and meets the required user specification. Data stores that were developed for technical issues or for storage of transitional values are not counted or included. Additional ability automatically accommodated are not counted except the customer especially demands from the client.

2.4 EXTERNAL INTERFACE FILES

An External Interface File (EIF) is a user recognizable cluster of logically associated data arranged outside the limits of the application. An example of an EIF is a file or table consisting names of codes read by the system being counted but maintained by some other application. The cluster of data is logical and user recognizable and meets required user specification, laid by the application, not maintained by the application, is also an ILF in some other application.

2.5 EXTERNAL INPUTS

An external input (EI) computes the data that is extracted from outside the application limit. An external input is the ease given to the customer to insert, update, and delete records of an ILF. One or more ILFs can be maintained. For example, an external input may uphold department and employee data and facts. The data and facts inserted will be saved in some or more ILFs. Another example may be the maintenance of system parameters, which will be used by the processes of the software system which is being developed. Data and facts are received from outside the application boundary or limit, input is the basic business affair as seen by the user, elaborated and self contained.

2.6 EXTERNAL OUTPUTS

An external output (EO) is a phenomenon that produces data sent outside the application boundary or limit, for e.g., the external output the customer views in the form of reports, messages, etc. An external output also procures the files the application produced to be used as transaction by another application. One or more ILFs or EIFs can be treated as an external output. Data and facts are sent outside the application limits for another ILFs or EIFs. The output is consequential to the customer's business perspective, elaborated and self contained. Data in the ILF or EIF is not changed by the external output. Count only unique external output.

3. GENERAL SOFTWARE METRICES

Effort, Development time, cost and productivity are considered as a general software metrics. COCOMO II model is adopted for estimating these metrics. COCOMO II requires software size in terms of LOC. In first layer we estimate size in unadjusted function point.

3.1 COCOMO II

COCOMO II is tuned to modern software life cycles. The original COCOMO model has been very successful, but it doesn't apply to newer software development methodologies as well as it does to traditional methodologies. COCOMO II targets the software projects of the 1990s and 2000s, and will continue to evolve over the next few years.
The primary objectives of the COCOMO II effort are:

- To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's.
- To develop software cost database and tool support capabilities for continuous model improvement.
- To provide a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.

3.2 EFFORT ESTIMATION

In COCOMO II effort is expressed as Person-Months (PM). A person month is the amount of time one person spends working on the software development project for one month. This number excludes time typically devoted to holidays, vacations, and weekend time off. The number of person-months is different from the time it will take the project to complete; this is called the development schedule or Time to Develop, TDEV. For example, a project may be estimated to require 50 PM of effort but have a schedule of 11 months.

$$PM = A \times (Size)^{E} \times \prod_{i=1}^{n} EM_i$$

Where A= 2.94

Scale Factor

The exponent E in equation is an aggregation of five scale drivers that account for the relative economies or diseconomies of scale encountered for software projects of different sizes. If E < 1.0 the project exhibits economies of scale. If the product's size is doubled, the project effort is less than doubled. For small projects, fixed start-up costs such as tool tailoring and setup of standards and administrative reports are often a source of economies of scale. If E = 1.0 the economies and diseconomies of scale are in balance. This

linear model is often used for cost estimation of small projects. If E > 1. 0 the project exhibits diseconomies of scale.

Table 1: Scale Factors (E) COCOMO II estimation model

Scale drivers	Very low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00
	or the estimated Process Maturity Level (EMPL)					

Early Design Model Cost Drivers

COCOMO II uses a set of effort multipliers to adjust the nominal person-month estimate obtained from the project's size and exponent drivers. This model is used in the early stages of a software project when very little may be known about the size of the product to be developed, the nature of the target platform, the nature of the personnel to be involved in the project, or the detailed specifics of the process to be used. This model could be employed in Application Generator, System Integration, or Infrastructure development sectors. The Early Design model uses KSLOC or unadjusted function points (UFP) for size. UFPs are converted to the equivalent SLOC and then to KSLOC. The application of project scale drivers is the same for Early Design and the Post-Architecture models. In the Early Design model a reduced set of cost drivers is used as shown in Table given below. The Early Design cost drivers are obtained by combining the Post-Architecture model cost drivers.

Table 2: Post-Architecture

Early Design	Post-Architecture Cost Drivers
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED

3.2 SCHEDULE ESTIMATION

Nominal-Schedule Estimation Equations

Both the Post-Architecture and Early Design models use the same functional form to estimate the amount of effort and calendar time it will take to develop a software project. These nominal-schedule (NS) formulas exclude the cost driver for Required Development Schedule, SCED. The amount of effort in person-months, PMNS, is estimated by the formula:

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$

$$\text{where } E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

$TDEV_{NS}$, it will take to develop the product is estimated by the formula:

$$TDEV_{NS} = C \times (PM_{NS})^{F}$$

$$\text{Where} F = D + 0.2 \times 0.01 \times \sum_{j=1}^{5} SF_j$$

The value of n is 16 for the Post-Architecture model effort multipliers, EM_i, and 6 for the Early Design model. The values of A, B, C, D, SF_1… and SF_5 for the Early Design model are the same as those for the Post-Architecture model. The values of EM_1… and EM_6 for the Early Design model are obtained by combining the values of their 16 Post-Architecture counterparts.

The subscript NS applied to PM and TDEV indicates that these are the nominal-schedule estimates of effort and calendar time. The effects of schedule compression or stretch-out are covered by an additional cost driver, Required Development Schedule. Size is expressed as thousands of source lines of code (SLOC) or as unadjusted function points (UFP). Development labor cost is obtained by multiplying effort in PM by the average labor cost per PM.

The values of A, B, C, and D are:

A = 2.94 B = 0.91
C = 3.67 D = 0.28

The initial baseline schedule equation for the COCOMO II Early Design and Post-Architecture stages is:

$$TDEV = [C \times (PM_{NS})^{(D+0.2\times(E-B))}] \times \frac{SCED\%}{100}$$

$$\text{where } C = 3.67, D = 0.28, B = 0.91$$

In Equation, C is a TDEV coefficient that can be calibrated, PM_{NS} is the estimated PM *excluding* the SCED effort multiplier, D is a TDEV scaling base-exponent that can also be calibrated. E is the effort scaling exponent derived as the sum of project scale drivers and B as the calibrated scale driver base-exponent. SCED% is the compression / expansion percentage in the SCED effort multiplier rating scale.

4. TOOL ARCHITECTURE

In this section architecture of the instrument is assign which measures unarranged function point in early implementation of life cycle of object oriented software. Harput transformation rules are used to calculate function points. In this section architecture of tool is conferred which figure early design software metrics in layered way, in the first layer object oriented function points are computed based on UML design particularization on approaching Harput rules. These function points are changed into source line of codes (SLOC), primary input for COCOMO II to figure General Software Metrics,

which is done in layer two of the tool. The most basic calculation in the COCOMO II model is the use of the Effort Equation to calculate the number of Person-Months required enroots a project. Most of the other COCOMO II results are imitative from this quantity. In this model, some of the most crucial points are contributing to a project's endurance and costs are the Scale Drivers. By using COCOMO II we can calculate effort in person month and development time. Now other metrics can be changes to by means of the following techniques.

1. PM to Dollars – On the basis of hourly salary
2. Productivity = FP/PM
3. Productivity = KLOC/PM
4. Development Cost = $/FP
5. Development Cost = $/LOC
6. Documentation= pages-of-documentation/FP
7. Documentation = pages-of-documentation/KLOC

Our main consideration is to figure software metrics in early design phase.

DESIGN APPROACH

UML design (Class diagram, Sequence diagram) designed in Rational Rose, as an input resource. We used Rational Rose Class diagram to calculate Data function types and order diagram for Transactional function type calculation. Class diagram and order diagram by rational rose develop design specification in UML syntax which is observed by analysis unit by applying Harput transformation rules. Both analysis unit and counting unit follow the rules to calculate unadjusted function points (UFP).

5. CONCLUSION

In this paper, function point analysis rules for design specification developed based on the UML is applied. Tool estimates object oriented software metrics in early life cycle phase, based on information of software in early design Harput rules and some guidelines to estimate size metrics and then COCOMO II techniques is applied to calculate rest of the software metrics. Tool architecture and design is only for object oriented software. Harput transformation rules and Uemura approach is used to automate function point estimation but still fully automatic model transformation still seems to be out of reach. Compared with FPA, the estimation error range will decreased as we are accounting for the complexities of generalization, aggregation and association which are not considered in traditional function point measurement techniques. This approach easily estimate the effort for a software development project based on its size using FPA.

REFERENCES

[1] A.J. Albrecht, "*Measuring Application Development Productivity*", Proc. IBM Applications Development Symp., Monterey, Calif. ,Oct 14-17, 1979.

[2] Harput V,Kaindl H,Kramer S.,"Extending Function Point Analysis to Object-Oriented Requirements Specifications ",procceding on 11th IEEE International Software Metrics Symposium (METRICS 2005).

[3] D.J Ram, S.V.G.K Raju," *Object Oriented Design Function Points*", -7695-0825-1/00 2000 IEEE.

[4] International Function Point User Group (IFPUG), Function Point Counting Practices Manual, Release 4.0, IFPUG, Westerville, Ohio, April 1990.

[5] Symons,C.:"*Function-Point Analysis: Difficulties and Improvements*." *IEEE Transactions on Software Engineering*, Vol. 14, Nr. 1, January 1988, pp. 2-11.

[6] Poensgen, B. and Bock, B. *Function-Point An]alyse*, dpunkt.verlag, Heidelberg, 2005.

[7] G. Caldiera, *G.* Antoniol, R. Fiutem, and C. Lokan. "*Definition and experimental evaluation of function points for object-oriented systems*"In *Proc. of the 5'h International Symposium on Software Metrics*, pages 167-178, November 1998.

[8] Sneed, H.M.: "Estimating the Development Costs of Object-Oriented Software." *Proceedings of 7th European Software Control and Metrics Conference*, Wilmslow,UK,

[9] Reifer, D.: "Web Development: Estimating Quick-to-Market Software." *IEEE Software*, November/December 2000.

[10] Gupta D.,Kaushal S.,Sadiq M.,"software Estimation tool based on three layer model for software engineering metrics" , ICMIT 2008.

[11] Prof. Ellis, COCOMO II.2000.0 Horowitz University of southern California, Center for software engineering, 1995