The Development and Achievements of Software Size Measurement

Tharwon Arnuphaptrairong

***Abstract*—Software size measurement is crucial for the software development process. It is used for project planning and control purposes during the project execution. It is required for productivity measurement after the project finished. Software size is also an important driver of software effort and cost estimation. This paper analyzed software sizing articles reviewed from the literature and presents the development, and achievements of software size measurement. Comments on the findings and future trends and challenges of the software size estimation models are also given.**

***Index Terms*— Software Size, Software Sizing Software size measurement.**

I. Introduction

Software size is crucial for the software development process. It is used for project planning and control purposes during the project execution. It is required for productivity measurement after the project finished. Software size is also an important driver of software effort and costs estimation. Most of the software effort estimation methods require software size. Boehm asserted in his list that "the biggest difficulty in using today's algorithm software cost models is the problem of providing sound sizing estimates [1]."

Lines of Code (LOC) and Function Points (FP) are two well known measures for software sizing. Over the years, there have been a lot of researches contributed to this area. This intrigues the question on how software sizing research has been developed and achieved. Hence, the objective of this paper is to review and analyze the software sizing articles from the literature and present the general picture the development and update the status of research in the field of software size measurement.

The remaining of this paper is organized as follows: Section II gives a review of the software size. Section III concludes and discusses the findings. Future trends of software sizing are also discussed in section IV.

II. Software Size Measurement Review

Review from the literature has shown that software size can be measured in either lines of codes, or function points and its variants. Only few other measures were mentioned.

T. Arnuphaptrairong is with the Department of Statistics, Chulalongkorn Business School, Chulalongkorn University, Bangkok 10250 Thailand (e-mail: Tharwon@chula.ac.th).

A. Lines of Code

Measuring software size in Lines of Code (LOC) can be achieved by either "counting" or "estimating / approximating".

Lines of Code Counting

Lines of Code (LOC) or Source lines of code (SLOC) is probably the oldest software matrix used to measure software size, since the first program was typed on cards one instruction per line per card. It is simply counting the number of lines in the text of the program's source codes.

Counting for SLOC is feasible when the program is completed. The counting is helpful for productivity and performance measurement and evaluation. However, for planning of software effort, duration and cost purpose, one cannot wait until the software is finished. Estimating or approximating the LOC of the program to be developed is then necessary before it is actually built.

Lines of Code Estimation

There are two methods in order to estimate the lines of code (LOC), either by using a model with some parameters or without any parameters. To estimate the LOC using a model with some parameters is sometimes called model based method or derived method. To estimate the LOC without any parameters is also known as non-model based method or direct method [2]. The direct methods or direct estimation methods include: expert judgment, Delphi, Wideband Delphi (expert consensus), analogy or case-based reasoning (CBR), thumb's rule, standard component, and three points estimation. The derive methods comprises of methods such as, extrapolative counts, components based method and backfiring. The following section gives more details for these methods.

Direct estimation method

Expert judgment [1], [3], [4], sometimes called heuristic method, refers to the estimation method based on the expertise, skill and experience of one or more experts. This method needs someone who already familiar with the domain of the software to be built.

Delphi or Shang technique [2]-[4] was originally developed by Rand Corporation in 1944. This method may be considered as a subset of expert judgment method, where group of experts are asked for the estimates until the consensus is reached.

Wideband Delphi or expert consensus [3], [4] is a subset of Delphi method proposed by Rand Corporation and

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

later modified by Boehm [5]. Original Delphi technique avoids discussions. Wideband Delphi accommodates group discussion to achieve the consensus.

Analogy or Case-Based reasoning (CBR) [4], [6], [7] involves the retrieval of the similar projects from the repository and use of the knowledge learned from those projects for the estimation of the new project. Accessing the similarity may use variety of techniques including -- Euclidian distance, nearest distance, manually guided induction, template retrieval, goal directed preference, specialty preference, frequency preference, recency preference, and fuzzy similarity.

Thumb's Rule [3], [8], the estimation is made according to a rough and ready practical rule or way of guessing, not based on science or exact measurement.

Standard component or three point estimation [2] **or PERT technique** [4], this is rather a technique than a method to accompany with other direct estimation method in order to improve the estimation. It involves the expert judgment of three possible estimates – the highest (Max), the most likely and the lowest possible (Min). The estimate of the LOC is computed as: LOC = (Max + 4 Most likely + Min) / 6.

Machine Learning (ML), a subfield of artificial intelligence, has been applied in the area [9, 10]. Machine learning method is to automatically inducing knowledge in the forms such as models, functions, rules and patterns, from historical project data. Regolin et al. [10] demonstrated how two machine learning algorithms --genetic programming (GP) and neural networks (NN) can be used to predict lines of code from function points (FP), or number of components (NOC) [11]. Machine learning algorithms such as genetic programming (GP) and neural networks are considered as "black boxes" and therefore it is not easy to explain how it works to the users [12].

Derived method

This LOC estimation method uses a model with some known (at the point of estimation) software attributes as parameters or drivers to estimate the LOC.

Extrapolative counts, this method extrapolates the LOC from the countable components using statistical method or other theoretical basis, for example, work of Tan, Zhao and Yuan [13]. Zhao, Yuan and Zhang [13] used conceptual data model to estimate the LOC. By using regression techniques they found the relations between KLOC (Kilo Source Lines of Code), C (number of class), R (Number of relation) and $\bar{A}$ (Average number of attributes per class). The relation can be expressed as:

$$KLOC = \beta_0 + \beta_1 C + \beta_2 R + \beta_3 \bar{A}$$

From the dataset they analyzed, the Java-Based system, the relation is as:

$$KLOC = -10.729 + 1.342 C + 1.254 R + 0.889 \bar{A}$$

Components based method, Verner and Tate [11] discussed a method that is bottom up approach by sizing the individual software components e.g., menus, screen, report components, relations and updates. Then sum up all of the components sizes to obtain the software size.

Backfiring method, this method came from the research of Caper Jones [14]. This method estimates the lines of codes needed to implement a Function Point value (Albrecth's Function Points) of a program or a piece of software for different languages. Details of this method are discussed in the next section –Function Points Estimation.

B. Function Point

Function Point (FP) was originated in 1979 and widely accepted with a lot of variants, from both academics and practitioner [15]. The research in this area is also known as **Function Point Analysis (FPA) or Function Size Measurement (FSM)**. The FP measurement could be classified into FP counting and estimation [2].

Function Point Counting

Function Point was introduced by Albrecht [16], the concept is based on the idea that the functionality of the software delivered is the driver of the size of the software (LOC). In other words, the more the functions delivered, the more the LOC. The functionality size is measured in terms of Function Points (FP).

FPA assumes that a software program comprises of functions or processes. In turn each function or process consists of five unique components or function types as shown in Figure 1. The five function types are External Input (EI), External Output (EO), External Query (EQ), Internal Interface File (ILF), and External Interface File (EIF).

Each of these five function types is individually assessed for complexity and given a Function Point value which varies from 3 (for simple external inputs) to 15 (for complex internal files). The Function Point values are based the complexity of the feature being counted.

The low, average and high complexity level of ILF and EIF are based on the number of Record Element Type (RET) and Data Element Type (DET). A Record Element Type (RET) is a subgroup of the data element (record) of an ILF or ELF. A data element type is a unique non-repeated data field.

The complexity level of EI and EO and EQ are based on the number of File Type Referenced (FTR) and Data Element Type (DET). A File Type Referenced (FTR) is an ILF or EIF.

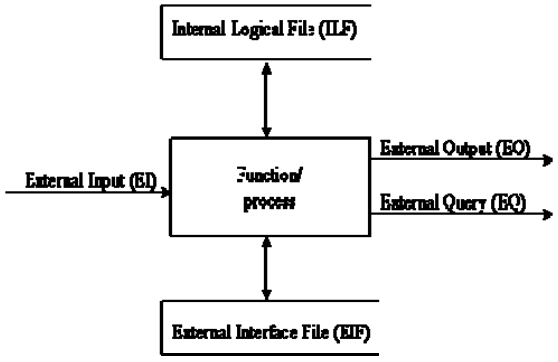

Fig. 1. The Albrecht five function types

The Unadjusted Function Points (UFP) or Unadjusted Function Points Counts (UFC) is calculated as follows [4]:

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The sum of all the occurrences is computed by multiplying each function count (N) with a Function Point weighting (W), and then the UFP is attained by adding up all the values as follows:

$$UFP = \sum_{i=1}^{5}\sum_{j=1}^{3} N_{ij} W_{ij}$$

Where Nij is the number of the occurrences of each function type i of the five types and Wij is the corresponding complexity function point weighting value j of the 3 levels –low, average and high.

The Function Point values obtained can be used directly for estimating the software project schedule or software costs. But in some cases, it may need further adjustments with the software development environment factors.

In order to find adjusted FP, UFP is multiplied by technical complexity factors (TCF) which can be calculated by the formula:

$$TCF = 0.65 + (\text{sum of factors}) / 100$$

There are 14 technical complexity factors --data communications, performance, heavily used configuration, transaction rate, online data entry, end user efficiency, online update, complex processing, reusability, installation ease, operations ease, multiple sites, facilitate change, distributed functions. Each complexity factor is rated on the basis of its degree of influence from no influence (0) to very influential (5). The adjusted Function Points (FP) or Function Point Counts (FC) is then derived as follows:

$$FP = UFP \times TCF$$

Evolution of the FPA method

The International Function Point User Group (IFPUG) is the organization establishes the standards for the Function Point Size Measurement to ensure that function points counting are the same and comparable across organizations**.** The counting manual can be found at http://www.ifpug.otg.

The International Standard Organization (ISO), in 1996, established the common standard, in order to support the consistency and promote the use of this Function Size Measurement (FSM). The updated versions are maintained. Besides the IFPUG FPA, three other FPA variants are also certified methods by ISO --Mk II, NESMA, and COSMIC FFP. These methods are reviewed in the following sections.

Function Points Estimation

In some situations when counting was not possible because of the lack of detailed information needed, many surrogate methods were suggested. This can either be direct estimation method or derived method [2].

Direct estimation method

The direct estimation methods are the same techniques already described in direct method for LOC estimation. These include expert judgment, Delphi or Shang techniques and its variants –Wideband Delphi (expert consensus), three points or standard component, analogy or case-based reasoning (CBR), thumb's rule and machine learning method. Expert judgment, Delphi or Shang techniques and Wideband Delphi (expert consensus) are also called expert opinion method [2].

Machine Learning Method, as reviewed in LOC estimation, Regolin [10] demonstrated how two machine learning algorithms –genetic programming and neural networks can be used to predict lines of code (LOC) from Function Points (FP), or number of components (NOC) [11]. Conversely, this implies that Function Points can be estimated from lines of code.

Derived Method

The derived method, some surrogate variables or algorithms are suggested in order to obtain the Function Points. Literature in [2] includes methods such as, extrapolative counts, sample counts, average complexity estimation, catalogue of typical elements, Early Function Points Analysis (EFPA), backfiring and others variants.

Extrapolative counts, this method extrapolates the FP counts from the countable components (usually the Internal Logical File (ILF)) using statistical method (mostly regression analysis). This method, the size of the whole system is approximated with respect to some FP components (LIF, EIF, EI, EO, or EQ). A few examples – Mark II, NESMA's Indicative FP, Tichenor ILF Model, Prognosis by CNV AG, and ISBSG Benchmark are reviewed as follows:

Mark II FPA [17] or Mk II FP was originated by Charles Symon and published in 1988. Mk II is mainly used in the United Kingdom. Mk II FPA assumes that a software program comprises of functions or Logical transaction or Basic Function Component (BFC) which is elementary process. Each function or process consists of 3 components –input, process, and output. The function size (Mk II UFP)

= W_i * number of input data element types +
W_e * number of entity types referred +
W_o * number of output data element types

W_i, W_e, W_o are the function points weighting values. The industrial weighting values are 0.58 for W_i, 1.66 for W_e, and 0.28 W_o

NESMA's Indicative FP (Netherlands Software Metrics Association) or the Dutch method [18].

Indicative Size Unadjusted Function Point (UFP)
= 35 * no. of ILF + 15 * no. of EIF

Tichenor ILF Model is another example. The UFP is computed as the following:

UFP = No. of ILF * 11.01
FP = 1.0163 * (UFP * VAF) $^{1.0024}$

VAF is Value Adjusted Factors.

Prognosis by CNV AG uses the following model:
FP = 56 + 7.3 * #IO; Where #IO = number of EI + EO;

ISBSG Benchmark employs the following model:
UFP(only ILF) = 7.4 * #ILF

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$$UFP(Total) = UFP(only\ ILF) / 22 * 100$$

Asensio *et al.* [19] is an example of recent works of this category. Asensio [19] argued the need for estimates at the early stage of software development when the required document is not available yet. They, therefore, proposed a method called "Early Function Point Method (EFPM)". The FP can be found using the following regression equation:

$$FP = 130{,}327 + 15{,}902 * CILE$$
$$FP = 66{,}905 + 13{,}035 * CILEEIF$$
$$FP = 50{,}784 + 6{,}289 * CEIEOEQ$$

Where CILE is the counter of ILFs, CILEEIF is the counter of ILFs+EIFs, and CEIEOEQ is the counter of EIs + EIs + EQs.

Asensio *et al.* [19] also claimed that he included a greater number of sample projects in his work and supposed an advantage in comparison to Tichenor ILF Model or Function Point prognosis CNV AG mentioned above.

Sample counts, this method counts Function Points from parts of the system and the rest or the system is then estimated based on these counts. With this method, only some portions of the system are investigated but with respect to all FP components (LIF, EIF, EI, EO, or EQ) [2].

Average complexity estimation --Instead of following the IFPUG complexity by classifying the components for low, average, or high complexity, One may use the average complexity for all components (EI, EO, EQ, ILF, and EIF) [2]. For example, the estimated UFP of ISBSG using average complexity values is as follows:

$$UFP = EI * 4.3 + EO * 5.4 + EQ * 3.8 + ILF * 7.4 + EIF * 5.5$$

Tichenor and NESMA also follow this approach and proposed their own average complexity weights.

Catalogue of typical elements, this method catalogues or lists the typical functionalities or processes, for example, create, add, delete and update. Then identifies the number of UFP needed for those processes. The UFP values are ready for the estimators to use when they come across these functions [2].

Backfiring method, this method came from the research of Capers Jones [14]. This method derives the Function Point values of a program or a piece of software from it number of lines of codes (LOC). Jones gives a very detailed table for converting between LOC and FP for different programming languages.

C. Function Point variants

To overcome the shortcoming of Albrecth's FPA, a number of refinement methods were proposed. Major contributions are Function Bang Metric [20], Feature Points [14], 3D FP [21], EFP [22], FFP [23] and COSMIC FFP [24], [25], and FP estimation from Structured Analysis (SA) [26]-[31], for more details see [15]. Some prominent measures will be reviewed below.

Function Bang Metric [20], [32], DeMarco categorized systems into three groups: function-strong, data-strong and hybrid systems. The categorization based on the calculated ratio of RE/FP where RE stands for the number of the relationships in the retained data model and FP means the number of the function primitives –bottom level process of the data flow diagram (DED). The system is considered function-strong, if the ration RE/FP is less than 0.7.The system is considered data-strong, if the ration RE/FP is greater than 1.5. The system is hybrid system if otherwise. For function strong systems, function bang (FB) metric is calculated as follows:

$$FB = \sum wi * C\ F\ P\ I\ i$$

Where the term C F P I i is derived from the Halstead's model for counting a program size and is calculated as follows:

$$C\ F\ P\ I\ i = (TC\ i * \log 2\ (TC\ i)) /2$$

The term TCi represents the number of data tokens around the boundary of the i function primitive in a DFD (Dataflow Diagram). The data tokens are data items that need not to be subdivided with in the function primitive.

The term C F P I i needs to be adjusted with the complexity of each function primitive by 16 weighting factors (wi) suggested by DeMarco.

Feature Points [14], the original Function Points method was developed with the primary aim for management information systems which is data intensive. In 1986, Feature Points was therefore developed by Software Productivity Research (SPR) in order to anticipate the real time system and algorithm intensive systems. Feature points method adds a new parameter –an algorithm to the five function point parameters with a default weight of 3. Feature points method also reduces the ILF weight from 10 to 7.

Early Function Points (EFP) and Extended Function Points (XFP) were proposed by Meli [22], to anticipate for the need of software size estimate at the early stage of the development life cycle. The method requires the estimator to put in knowledge at different detail levels of a particular application. Functionalities are classified as: Macrofunction, Function, Microfunction, and Functional Primitive. Each type of functionality is assigned a set of FP value (minimum, average, and maximum).

COSMIC FFP [23]-[25], The Full Function Points (FFP) method was originated in Canada. The purpose is to extend the IFPUG FPA accuracy of the real time systems estimation. The research groups were later formed as Common Software Measurement Consortium (COSMIC). The FFP method was then modified and referred to as COSMIC Full Function Points (COSMIC FFP). In the COSMIC FFP method, the Functional User Requirements (FURs) of the software is broken down into "functional process type"

The elementary process comprises of a unique cohesive and independently executable set of data movement types

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

which are –Entries (E), Exits (X), Reads (R), and Writes (W).

The CSOMIC FFP method breaks down the software architecture into software layers. The software layers can receive requests from the layers above and can request for services from the layers below. For the software components in the same layer, peer to peer communication can also be employed.

Each data movement –E, X, R and W, is assigned a size of one Cfsu (COSMIC function size unit). The sum of all data movements of all functional processes will give the size of a piece of software.

FP Estimation and Structured Analysis (SA), DFD and ERD

Functionality is the heart of FPA. One stream of research proposed that functionalities can be retrieved using Structured Analysis (SA) which expressed in the form of Dataflow Diagram (DFD) for process modeling and Entity Relationship Diagram (ERD) for data modeling FPA.

DFD was proposed as the estimator for FPA by a number of papers using either DFD alone or together with ERD [26]-[31].

Rask [26, 27] introduced the algorithm for counting the Function Points using specification from DFD and ERD data model. The automated system was also built.

O'brien and Jones [28] proposed a set of counting rules to incorporate Structured Analysis and Design Method (SSADM) into Function Points Analysis. DFD, together with I/O structure diagram, Enquiring Access Path (EAP) and Effect Correspondence Diagram (ECD) were applied to the counting rules for the Mark II FPA.

Shoval and Feldman [29] applied Mark II Function Points with Architectural Design of Information System Based on structural Analysis (ADISSA). The proposed method counts the attributes of all inputs and outputs from the Dataflow Diagram (DFD) of the system to be built and all of the relations in the database from the database design process, and then plugs in all the numbers in the Mark II model.

DFD was found also proposed to be used together with ERD in [30]. Lamma et al. [31] to solve the problem of counting error, a system for automating the counting is built and called FUN (FUNction points measurement). The system used the specification of a software system from Entity Relationship Diagram and Dataflow diagram to estimate software Function Points. Later, the system was automated by Grammantieri *et al* [32].

D. Other Function Points "Like" Measures

With the advent of new technologies, platform and languages, for example, Object Oriented technologies and web technologies, researchers have incorporated the information obtained from object-oriented analysis and design approach into the object-oriented software size estimation. Literature had shown a number of proposed new measures to handle these advancements [33], [34]. To cover this view point, the following section explores some of these frequent mentioned measures **--Use Case Points [35], Predictive Object Points (POPs) [36], Class Points [37], Object-Oriented Function Points [38], Object Oriented Design Function Points [39], Web Points [40], OOmFP [41], UML Points [42], and Pattern Points (PP) [43].**

Use Case Points (UCP) method [35], [44] was introduced by Karner's 1993 M.Sc. thesis under supervision of Ivar Jacobson written while Karner worked at Objectory AB, (now Rational Software). UCP utilized the information available in use case diagrams to estimate the software size. The use case diagrams contain information about the behavior of the software available from the requirement analysis. UCP uses 2 drivers –actors and use case. The procedure to calculate the Use Case points is as follows:

1. Actors are classified as simple (for example, API), average (for example TCP/IP), and complex (for example GUI web page) with the weight of 1, 2 and 3 respectively. The total unadjusted actor weight (UAW) is calculated as:

$$UAW = \sum actor_i \text{ x } weight_i$$

2. Use cases are classified into $\leq$ 3, 4-7, and $>$ 7 transactions with the weight of 5, 10 and 15 respectively. The Unadjusted Use Case Weights (UUCW) and the Unadjusted Use Case Points (UUPC) are calculated as:

$$UUCW = \sum UC_i \text{ x } weight_i$$

and $UUCP = UAW + UUCW$

3. Calculate the Technical Complexity Factor (TCF) and Environmental Factor (EF) from the following equations

$$TCF = 0.6 + (0.1 * TFactor)$$

Where TFactor is calculated by assigning value 0-5 for each factor, from factor 1 to 13, and then multiply by its weight. Whereas,

$$EF = 1.4 + (-0.03 * EFactor)$$

Where EFactor is calculated by assigning value 0-5 for each factor from, factor 1 to 8, and then multiply by its weight.

4. The Use Case Points (UCP) is then calculated by:

$$UCP = UUCP * TCP * EF.$$

Predictive Object Points (POPs) method [36] was proposed by Minkiewicz for measuring object-oriented software size. Regression was performed on the data set used and settled with the following equation:

POPs (WMC, NOC, DIT, TLC) = WMC * $f1$ (TLC, NOC, DIT) * $f2$ (NOC, DIT)

Where TLC = the number of top level classes
DIT = average depth of inheritance tree
NOC = average number of children per base class
WMC = average number of weighted methods per class

$f1$ attempts to size the overall system and $f2$ applies the effects of reused through inheritance. To find the WMC, the

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

author, followed Booch [45], classified 5 method types – constructors, destructors, modifiers, selectors, and iterators with corresponding weights.

Class points method [37] was introduced by Costagilola *et al.* in 1998. The class points counting process consists of 4 steps:

1) Identify and classify of user classes.

System components are classified into 4 types --problem domain type (PDT), human interaction type (HIT), data management type (DMT), and task management type (TMT). The classes are identified and classified from these system components.

2) Evaluate the complexity of the classes

Two measures CP1 and CP2 are suggested. As for CP1 the complexity level of a class is assigned based on the Number of Services Requested (NSR) and the Number of External Methods (NEM) where as for CP2, the Number of Attributes (NOA) is also taken into account together with NSR and NEM. The measure CP1 is helpful for the initial size estimation at the early stage whereas the CP2 measure is suitable when more information is available at the later stage.

3) The Total Unadjusted Class Points (TUCP) is then calculated as:

$$TUCP = \sum_{i=1}^{4}\sum_{j=1}^{3} W_{ij}X_{ij}$$

Where Xij is the number of classes of system component type i (PDT, HIT, DMT, TMT) with the complexity level j (Low, Average, High) and Wij is the weight for the type i and complexity level j.

4) Adjusting the TCUP with Technical Complexity Factors (TCF)

The Technical Complexity Factors (TCF) is determined by giving the value of 0 to 5 depending on the degree of influence of the 18 technical system characteristics have on the application. The sum of the influence values is called Total Degree of Influence (TDI). The Technical Complexity Factors (TCF) is then computed as:

$$TCF = 0.55 + (0.01 * TDI)$$

The adjusted Class Points (CP) is computed as:

$$CP = TUCP * TCF$$

Object-Oriented Function Points (OOFPs), there are a number of proposals to adapt the traditional Function Points Analysis method with Object-oriented technologies. OOFPs method was proposed in works of Caldiera, *et al*. [46], and Antoniol, *et al*. [47].

According to Antoniol *et al.* [47] there are two streams of research in adapting FP approach to object-oriented software estimation. The first group are for example, Whitmire [48], Schoonevendt [49], IFPUG 1995 [50], and Fetcke *et al.* [51]. While keeping the same concept and counting method of traditional FP they introduced the method to get around when dealing with the counting method from classes diagram, to ILF and ELF. The second stream invented new measures to exploit the additional information gained from the OO method. For example, Sneed [52] who proposed Object Points, Minkiewizcz [36], Mehler and Minkiewizcz [53], presented the Predictive Object Points (POPs), and Graham (54) introduced Task Points.

Antoniol *et al*. [47] claimed that OOFPs share the characteristics of both groups. OOFPs keeps the same function points weights as in IFPUG. OOFPs maps classes to Internal Logical File (ILF) and External Interface Files (ELF), and methods to transactions. FP transactions –EI, EO, and EQ are treated as generic service requests (SRs). Counting OOFPs can be reached by:

$$OOFP = OOFP_{ILF} + OOFP_{EIF} + OOFP_{SR}$$

Where $OOFP_{ILF}$ = sum of the weights for each ILF objects, classified by the their DETs and RETs,

$OOFP_{EIF}$ = sum of the weights for each EIF objects classified by the their DETs and RETs,

$OOFP_{SR}$ = sum of the weights for each SRs, classified by the their DETs and FTRs .

The latest development and refinement of OOFP could be found in Zivkovic *et al.* [55].

Pattern Points method **(PP)** was proposed by Adelkile [43] in 2010. The model gives attention to UML sequence diagram for the object interactions. The method is based on size of each of the 23 object oriented design patterns defined in the book of *Gamma et al*. [56] entitled "Design patterns: Elements of Reusable Object-Oriented Software". Each of the pattern is sized based on a pattern ranking and an implementation ranking. The pattern ranking is a function of the degree of difficulty and the structural complexity of the design pattern, and the implementation ranking is a function of the ease of applicability of the pattern to the problem type.

Web Objects [40], Web Objects were introduced in 2000, as a measure appropriate for web applications sizing. Ruhe *et. al.* [57] proposed to add four new web-related components or web objects (WO) to the five functions types of FP approach for web applications development. The four web objects are multimedia files, web building block, scripts, and links.

Each instance of the web objects and the five function types are counted and classified in term of its complexity level: low, average and high and then multiply with the correspondence weight to the counted components. The sum of those values represents the function size of the web application in web objects

E. Other measures

Jones Very Early Size Predictor

Jones Very Early Size Predictor was developed by Capers Jones [14]. The method classifies software projects into different scopes, classes and types. There are altogether 10 scopes, 15 classes and 20 types. The size of the software to be developed is then computed by the following formula:

$$Size = (Scope + Class + Type)^{2.35}$$

Software science (Halstead)

Literature also showed different view of software

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

measurement. Halstead [58] proposed to measure software size using code length and volume metrics. Code length is designed to measure program source code length which is defined as:

$$N = N1 + N2$$

Where N is code length, N1 is the total number of operator occurrences, and N2 is the total number of the operand occurrences. Whereas Volume is the storage space amount required and is defined as:

$$V = N \log (n1 + n2)$$

Where V = Volume, N = Code length, n1 is the number of distinct operators and n2 is the number of operand that appears in the program.

III. Observation and Discussion

From the literature reviewed, the following observations are made:

1) Development of SLOC and FPA

Literature review has shown that software size can be measured in either lines of codes (LOC), or Function Points (FP) and Function Points variants. Only few other measures were mentioned. There is little research in LOC where as FP is widely accepted. Table I. shows the development of FPA method from 1979 to 2010.

Most of the researches are refinements of Function Points in order to address the problems of FPA. The developments can be classified into 4 streams.

It was argued that the FPA counting was complex and time consuming. From about 1979 to 1992, the first stream of research, therefore, tried to make the FPA counting simpler. Many surrogate methods were suggested in order to ease and speed up the software size estimation method, especially those extrapolation methods. To extrapolate, regression was found to be a very popular technique used to find the relations between software size and the software attributes in order to make the estimates. These include NESMA's Indicative FP, Tichenor ILF Model, Prognosis by CNV AG, ISBSG Benchmark, and Mark II.

Function Points method is good for business applications. From, 1979 to 1999, the second stream tried to cope with its weakness for scientific and real time applications. These Function Points extensions include Function Bang Metric, Feature Points, 3D FP, Full Function Points (FFP), and COSMIC FFP.

The third stream spreads from 1987 to 2006. Many researchers had arguing that functionalities can be retrieved using Structured Analysis (SA) which expressed in the form of Dataflow Diagram (DFD) for process modeling and Entity Relationship Diagram (ERD) for data modeling. These includes work of Rask [27, 28], O'brien and Jones [29], Shoval and Feldman [30] and Lamma et al. [31].

From 1993 to 2011, the object oriented technology has made a lot of impacts on the software size estimation methods. Traditional FPA is argued that it is not suitable with the object-oriented technologies. This latest stream includes Use Case Points (UCP), Class Points, Predictive Object Points (POP), Object-Oriented Function Points (OOFP) UML points, Web Object Points and Pattern Points.

Besides the 4 streams, in around 1997, there was a small group interest in solving the problems at the early stage software size estimation to satisfy the needs of estimation at the early stages of the software development life cycle, for example, Jones Very Early Size Predictor, Early Function Points Analysis (EFPA) and Meli's Early Function Points (EFP) and Extended Function Points (XFP) [20].

While there were many streams of interest for model based method, Artificial Intelligent (AI) techniques, such as simulation, machine learning, and neural network were the latest stream proposed for non-model techniques.

TABLE I
DEVELOPMENT OF FPA

Year	FPA Model
1979	Albreth FP
1980	
1981	
1982	Function Bang metric
1983	
1984	
1985	
1986	Feature Points
1987	
1988	Mark II FPA
1989	
1990	NESMA FP
1991	DFD (Rask's)
1992	3D FP
1993	UCP, SSADM (Obrien and Jones's)
1994	OO Metric
1995	
1996	ADISSA Object Points Task points
1997	Jones Very Early Size Predictor EFP FFP POP
1998	OOFP
1999	COSMIC FFP
2000	Object Oriented Design Function Points Web Object Points
2001	
2002	
2003	
2004	ER-DFD (Lamma's) OOmFP
2005	Class points (CP) OOFP(2)
2006	DFD (Gramantieri's) UML Points
2007	
2008	
2009	
2010	Pattern Points (PP)
2011	

2) The Needs for validation

Many software size estimation models and methods have been proposed but one of the questions is that which estimation model or method performs better. The answer is

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

probably there is no better one estimation method for every environment. [2]. A few research reported favor for different models for example, in [59] expert judgment seemed to perform better where as in [60], [61] favored the performance of neural network

The question is then which estimation method is the best under which circumstances or environment**.** More empirical evidence is therefore called for to answer this question.

IV. Future Trend and Software sizing

We can not reject the effects of changing technologies on the software size measurement. The literature has shown that technologies and techniques related to requirement gathering, and software analysis and design, such as, Structured Analysis and Design Method (SSADM), and Object-oriented Analysis and Design (OOAD), had impacted on the size measurement models. This is because they are directly related to the software functionality. Significant future challenges for software sizing is probably the sizing for new product forms which include requirement or architectural specifications, stories and component-based development [62]. Besides the new product forms, the new process forms, such as, extreme programming and agile methods is other aspect to look into.

References

[1] B.W. Boehm, "Software engineering economics," *IEEE Transaction of Software Engineering*, vol.10, no.1, pp. 4-21, 1984.

[2] R. Meli and L. Santillo, "Function point estimation methods: a comparative overview," in *FESMA '99 Conference proceedings,* Amsterdam, 4-8 October, 1999.

[3] M. Nasir, "A survey of software estimation techniques and project planning practices," In *Proceeding of the Seventh ACIS Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'06)*, 2006. pp. 305-310.

[4] H. Leong and Z. Fan, "Software cost estimation," [on line], 2002 Available: ftp://cs.pitt.edu/chang/handbook/42b.pdf

[5] B.W. Boehm, *Software engineering economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

[6] M. Sheppard and C. Schofield, "Estimating software project effort using analogies," *IEEE Transaction on Software Engineering,* vol. 23, no.11, pp.736-743; 1997.

[7] S. J. Delany, P. Cunningham and W. Wilke, "The limit of CBR in software project estimation," in *German Workshop on Case-Based Reasoning*, 1998.

[8] O. Demirors, and C. A. Gencel, "Comparison of size estimation techniques applied early in the life cycle," in *T. Dingsoyr (Ed), EuroSPI 2004, LNSC*, 2004, pp.184-194.

[9] D. Zhang and J.J. P.Tsai, "Machine learning and software engineering," in *Proceeding the 14th International Conference on Tools with Artificial Intelligence (ICTAI'02),* 2002.

[10] E. N. Regolin, G.A. de Souza, A.R. Pozo, and S.R. Vergilio, "Exploring machine learning techniques for software size estimation," in *Proceeding of the XXII International Conference of the Chilean Computer Science Society (SCCC'03),* 2003, pp.130-136.

[11] J. Verner and G. Tate, "A software size model," *IEEE Transactions on Software Engineering,* vol.19, no. 4, 1992.pp.265-278.

[12] A. Idril, T.M. Khoshogoftaar, and A. Abbran, "Can neural net works be easily interpreted in software cost estimation." in *Proceeding of the 2002 World Congress on Computational Intelligence,* Honolulu, Hawaii, May 12-17, 2002, pp.1162-1167.

[13] H.B.K. Tan, Z. Yuan and H. Zhang, "Estimating LOC for information systems from their conceptual data models," in *Proceeding of the International Conference on Software Engineering ICSE"06,* May 20-28, Shanghai, China. 2006.

[14] C. Jones, *Applied Software Measurement, Assuring Productivity and Quality,* 2ed, McGraw-Hill, 1997.

[15] C. Gencel and O. Demirors, "Functional size measurement revisited," *ACM Transaction on Software Engineering and methodology,* vol.17, no. 3, pp.15.1-15.36, June 2008.

[16] A. J. Albrecht, "Measuring application development productivity," in *Proceeding of the IBM Applications Development Symposium,* California, October 14-17, 1979, pp. 83-92.

[17] UKSMA, *Mark II Function Points Analysis Counting Practice Manual*, Version 1.3.1.

[18] NESMA, *Definitions and Counting Guidelines for the Application of Function Point Analysis*, Version 2.1, 2003.

[19] R. Asensio, F. Sanchis, F. Torre, V. Garcia and G. Uria, "A preliminary study for the development of an early method for the measurement in function points of a software product," ACM classes: D.4.8 [On Line], 2004 Available at: http://arxiv.org/abs/cs?papernum=0402015

[20] T. DeMarco, *Controlling Software Projects*, Yourdon Press, New York, 1982.

[21] S. A. Whitmire, "3D Function points: scientific and real-time extension to function points," in *Proceeding of the Pacific Northwest Software Quality Conferences,* 1992.

[22] R. Meli, "Early and extended function point: a new method for function points estimation," IFPUG Fall Conference, September, 15-19, Arizona, 1997.

[23] A. Abran, D. St. Peierre, M. Maya, and J. M. Desharnais, "Full function points for embedded and real-time software," in *Proceeding of the UKSMA Fall Conference, London,* 1998.

[24] COSMIC, *The COSMIC Functional Size Measurement Method*, Version 3.0, Common Software Measurement International Consortium, 2007.

[25] ISO, *ISO/IEC 19761: Software Engineering -- COSMIC FFP -- A Functional Size Measurement Method*, Version 2.2, 2003.

[26] R. Rask, "Algorithm for counting unadjusted function points from dataflow diagram" Technical report, University of Joensuu, 1991.

[27] R. Rask, "Counting function points from SA descriptions," *The Papers of the Third Annual Oregon Workshop on Software Metrics* (Ed. W. Harrision), Oregon, March 17-19, 1991.

[28] S. J. Obrien, and D. A. Jones, "Function points in SSADM," *Software Quality Journal,* vol. 2, no. 1, pp.1-11, 1993.

[29] P. Shoval, and O. Feldman, "Combining function points estimation model with ADISSA methodology for system analysis and design," in *Proceeding of ICCSSE"96,* 1996, pp.3-8.

[30] F. Gramantieri, E. Lamma, P. Mello, and F. Riguzzi, "A system for measuring function points from specification," Technical Report, Universitra di Bologna, 1997.

[31] E. Lamma, P. Mello and F. Riguzzi, "A system for measuring function points from an ER-DFD specification," *The Computer Journal*, vol. 47, no.3, pp.358-372, 2004.

[32] F. Gramantieri, E. Lamma, P. Mello, and F. Riguzzi, "A System for Measuring Function Points from Specifications," DEIS – Universita di Bologna, Bologna. and Dipartimento di Ingegneria, Ferrara, Tech. Rep DEIS-LIA-97-006, 1997.

[33] G. Antoniol, C. Lokan, G. Caldiera, and R. Fiutem, "A function point-like measure for object-oriented software," *Empirical Software Engineering,* vol. 4, no. 3, pp. 236-287, 1999.

[34] D. Card, K. El Eman, and B. Scalzo, "Measurement of object-oriented software development projects," Technical Report, Software Productivity Consortium, Herndon, VA, 2001.

[35] G. Karner, "Metrics for objectory," Diploma Thesis, University of Linkoping, Sweden, No. LiTH-IDA-Ex- 9344:21, December 1993.

[36] A. Minkiewicz, "Measuring object-oriented software with the predictive object points," in *Proceeding of 8th European Software Control and Metrics Conference,* Atlanta, 1997.

[37] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, "Class points: an approach for the size estimation of object-oriented systems," *IEEE Transaction on Software Engineering*, vol. 31, no.1, pp.52-74, January 2005.

[38] G. Caldiera, C. Lokan, G. Antoniol, R. Fiutem, S. Curtis, G. La Commare, and E. Mambella, "Estimating size and effort for object-oriented systems," In *Proceeding of 4th Australian Conference on Software Metrics*, 1997.

[39] D.J. Ram and S.V.G.K. Raju, "Object oriented design function points", in *Proceedings. First Asia-Pacific Conference on Quality Software*, October 30-31, 2000, pp121-126.

[40] D. Reifer, Web-Development: "Estimating quick-time-to-market software." *IEEE software,* vol. 17, no. 8, pp.57-64, November/December 2000.

[41] S. Abrahao, G. Poels, and O. Pastor, "A functional size measurement method for object oriented conceptual schemas: design and evaluation issues", Working paper, Faculty of economic and business administration, Ghent University, 2004.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[42] S. E. Kim, W. Lively, and D. Simmons, "An effort estimation by UML points in the early stage of software development," in *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006,* Las Vegas, Nevada, USA, Volume 1, June 26-29, 2006.

[43] O. Adekile, D.B. Simmons, W.M. Lively, "Object oriented software development effort prediction using design patterns from object interaction analysis." in *Proceeding of 2010 Fifth International Conference on Systems*, Menuires, April, 2010, pp.47-53.

[44] B. Anda, H. Dreiem, Dag I. K. Sjoberg and M. Jorgensen, "Estimating software development effort based on Use Cases --experiences from Industry, In G. Martin, and C. Kobryn (ed) *UML 2001: the unified modeling language: modeling languages, concepts, and tools*: 4th international conference, Toronto, Canada, October 1-5, 2001.

[45] G. Booch, *Object Oriented Analysis with Applications*, 2nd Edition. Benjamin/Cumming Publishing CO. Inc. Redwood City, CA. 1994.

[46] G. Caldiera, G. Antoniol, R. Fiutem, and C. Lokan, "Definition and experimental evaluation of function points for object-oriented systems," in *Proceeding of 5th International Symposium on Software Metrics, IEEE 1998*, pp. 167-178.

[47] Antoniol, G., Fiutem, R., and Lokan, C. "Object-oriented function points: an empirical validation," *Empirical Software Engineering*, vol. 8, no. 3, pp. 225-254, 2003.

[48] S. A. Whitmore. "Applying function points to object oriented software." in J. Keyes (ed), *Software Engineering Productivity Handbook*, Chapter 13, McGraw-Hill, 1993.

[49] M. Schoonneveldt. "Measuring the size of object oriented systems." in *Proc. 2nd Australian Conference on Software Metrics,* Australian Metrics Associations. 1995.

[50] IFPUG, *Function points counting practice: case study3 –object oriented analysis, object-oriented design* Draft, International Function Point Users Group, Westerville, Ohio, 1995.

[51] T. Fetcke, A. Abran, and T. H. Nguyen, "Mapping the OO-Jacobson approach to functions points analysis," in *Proceeding of IFPUG 1997 spring Conference,* pp.134-142. 1997

[52] H. Seed, "Estimating the development costs of object-oriented software," in *Proceeding of 7th European Software Control and Metrics Conference*, Wilmslow, UK., 1996.

[53] H. Mehler and A. Minkiewwicz, "Estimating size for object-oriented software," in *Proceeding of ASM'97 Application in Software Measurement,* Berlin, 1997.

[54] I. Graham, "Making Progress in Metrics," *Object Magazine*, vol. 6, no. 8, pp. 68-73, 1996.

[55] A. Zivkovic, I. Rozman, and M. Hericko, "Automated software size estimation based on function points using UML model" *Information & Software Technology*, vol. 47, pp. 881-890. 2005.

[56] E. Gamma, R. Helm, R. Johnson, and J.M. Vissides, *Design Pattern: Elements of reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

[57] M. Ruhe, R. Jeffery, and I. Wieczorek, "Using web objects for estimating software development effort for web application," in *Proceeding of the ninth International Software Metric Symposium (METRICS'03)*, 2003.pp.30-37.

[58] M.H. Halstead, *Elements of Software Sciences,* Elsevier, New York, 1977

[59] M. Jorgensen, "Forecasting of software development work effort: evidence on expert judgment and formal models," *International Journal of Forecasting*, vol.23, no. 3, pp:449-462, 2007.

[60] G. E. Witting, and G. R. Finnie, "Using artificial neural network and function points to estimate 4GL software development effort," Australian journal of information system, vol.15, no. 2, pp.87-94 2008.

[61] J. Kaur, S. Singh, K.S Kahlon, "Comparative analysis of the software effort estimation models," Proc. Of World Academy of Science, Engineering and Technology, vol.36, pp.485-487, December 2008.

[62] B. Boehm and R. Valerdi, "Achievement and challenges in COCOMO-based software cost estimation," Journal of IEEE Software, vol.25, no.5, pp.74-83, 2008.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)