Software Size Estimation in Incremental Software Development based on Improved Pairwise Comparison Matrices

Peter Ochieng
Jomo Kenyatta University of Agriculture and Technology
P.O BOX 62000 NAIROBI

Waweru Mwangi
Jomo Kenyatta University of Agriculture and Technology(JKUAT)
P.O BOX 62000 NAIROBI

Solomon Mwanjele Mwgha
Taita Taveta Unversity
P.O BOX 635 VOI

ABSTRACT
Software sizing is a crucial activity among the task of software management. Work planning and subsequent estimation of effort required is based on the estimate of the software size required. Software developers are realizing the need to speed up the development process to respond to customers' needs .This has resulted in adoption of rapid development methods and adoption of agile methodologies.

Incremental method of software development has been adopted as one of the methods to speed up software development. Unfortunately there is little work that has been done to develop a clear framework to estimate software size and cost in incremental software development environment.

This research work proposes the use of Pairwise Comparison matrices framework to estimate size and cost in incremental software development and evaluate the pairwise comparison framework against Putman's size estimation model to determine if it produces more accurate results in terms of estimation of size relative to actual size.

Keywords
Pairwise; judgment matrix; Incremental Estimation

1. INTRODUCTION
Before development of any software, it is vital for proper planning to be conducted. Projected size of the software to be developed is an important variable that is needed by software project managers to estimate the cost of the software, number of people to allocate to the development of the software and the number of months or duration the development lifecycle will take. Since software size estimate prior to development is non-existence or abstract , it needs experienced human judgment to estimate the size of the software prior to development .The idea of using human experience and judgment fits well in the field of Human-centered computing (HCC) .In these stages most of the required information is not available. To help them in this difficult task, prediction models and the experience of past projects are fundamental. Software size metrics play a significant role to the success of this task. Unfortunately the existing software size estimation models still produce size estimates which have been blamed for software development failures. The popular computing literature is awash with stories of software development failures and their adverse impacts on individuals, organizations, and societal infrastructure. Indeed, contemporary software development practice is regularly characterized by runaway projects, late delivery, exceeded budgets and reduced functionality and questionable quality that often translate to cancellations, reduced scope and significant rework circles [1].

The net result is an accumulation of waste typically measured in financial terms (always billions of dollars) [2]. The Standish Group makes a distinction between failed projects and challenged projects. Failed projects are cancelled before completion, never implemented, or scrapped following installation. Challenged projects are completed and approved projects that are over budget, late, and with fewer features and functions than initially specified. Most organizations are constantly searching for ways to improve their project management practice and reduce the likelihood of failures and the degree to which their projects are challenged. Typical projects entail a balancing act between the triple constraints of budget, schedule, and scope. Tradeoffs and adjustments are therefore made by restricting, adding to, or adjusting the cost, time, and scope associated with a project. Indeed the traditional triangle in project management is said to be concerned with finding a balance between cost, time, and scope as show in Figure1.

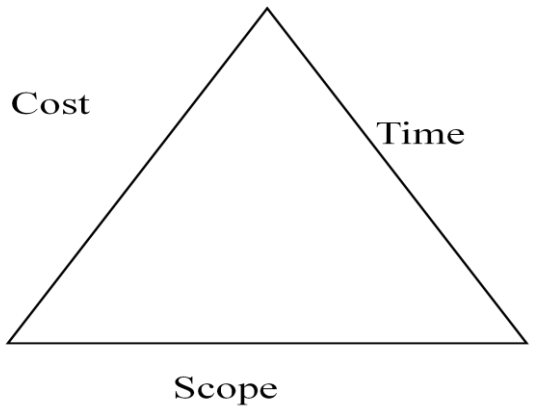

Figure 1: Traditional triangle in project management

For example, the more that is requested in terms of scope (or arguably even the performance or the quality), the more it is likely to cost and the longer the expected duration. If the client needs to have a certain performance delivered very rapidly, this will increase the cost due to the need to work faster and have more people involved in the development. The

more features expected from a system, the higher the cost and the longer the expected duration. Conversely, if the costs need to be kept to a minimum, one may need to consider the essential performance, or the overall scope, and compromise there. Many managers quickly discover that the triangle is not flexible. In order to address the challenge. In order to address lack of clear framework of software size estimation this paper proposes to develop software size estimation framework for incremental development environment using pairwise comparison matrix.

2. LITERATURE REVIEW

In general size estimates of an application is presented as lines of codes (LOC) or function points (FP).There are techniques that can be applied to convert function points to lines of code for specific language, and vice versa. One of the best techniques used to do this is called backfiring technique[10].This paper will focus on the methods using LOC since our new methodology also uses LOC ,this will make comparison between the methods easy.

2. 1 Lawrence H. Putnam LOC Estimation

A SLOC estimate of a software system can be obtained from breaking down the system into smaller pieces and estimating the SLOC of each piece. [9] Putnam suggests that for each piece, three distinct estimates should be made:

Smallest possible SLOC – *a*

Most likely SLOC – *m*

Largest possible SLOC – *b*

Putnam suggested that three to four experts make estimate of a, b, c for each function. Then the expected SLOC for piece $E_i$can be computed by applying beta distribution as shown in equation (1)

$$E_i=\frac{a+4m+b}{6} \quad (1)$$

The expected SLOC for the entire software system *E* is simply the sum of the expected SLOC of each function as shown in equation (2)

$$E=\sum_{i=1}^{n} E_i \quad (2)$$

where *n* is the total number of pieces.

An estimate of the standard deviation of each of the estimates $E_i$can be obtained by getting the range in which 99% of the estimated values are likely to occur and dividing by 6 as shown in equation (3).

$$SD_i=\frac{|b-a|}{6} \quad (3)$$

standard deviation of the expected SLOC for the entire software system *SD* is calculated by taking the square root of the sum of the squares of standard deviations of each estimate SD_i as shown in equation(4)

$$SD=\sqrt{\sum_{i=n}^{n} SD_I} \quad (4)$$

where *n* is the total number of pieces.

Therefore the total software size is expected to lie in the range expressed in equation (5)

$$E \pm SD \quad (5)$$

Where E is the estimated size of the entire system computed as shown in equation (5)

With real experience this method can yield accurate results [7], though it also faces criticism of the allowed range of the size estimate as shown in equation (5) being large hence the estimate cannot be narrowed down to given value.

2.2 Developer opinion and previous project experience

Software cost estimates are typically required in the early stages of the life-cycle when requirements and design specifications are immature. Under these Conditions, the production of an accurate cost estimate requires extensive use of expert judgment and the quantification of significant estimation uncertainty. Research has shown that under the right conditions, expert judgment can yield relatively "accurate" estimates [9]. Unfortunately, most expert judgment-based estimates do not meet these conditions and frequently degenerate into outright guessing. At its best, expert judgment is a disciplined combination of a 'best guess' and historical analogies. Developer opinion is otherwise known as guessing. If you are an experienced developer, you can likely make good estimates due to familiarity with the type of software being developed. How well this estimates are, depend on the expertise of the person giving the estimate hence it cannot be empirically proved but only to trust the estimates. But incase of good experience by the developer it can yield good estimates.

Looking at previous project experience serves as a more educated guess. By using the data stored in the metrics database for similar projects, you can more accurately predict the size of the new project. If possible, the system is broken into components, and estimated independently.

2.3 Count Function Blocks

The technique of counting function blocks relies on the fact that most software systems decompose into roughly the same number of "levels"[5]. Using the information obtained about the proposed system, follow these steps:

1. Decompose the system until the major functional components have been identified

(Call this a function block, or software component).

2. Multiply the number of function blocks by the expected size of a function block to get a size estimate.

3. Decompose each function block into sub functions.

4. Multiply the number of sub functions by the expected size of a sub function to get a second size estimate.

5. Compare the two size estimates for consistency.

Compute the expected size of a function block and/or a sub function with data from previous projects that use similar technologies and are of similar scope.

If there are no previous developments on which to base expected sizes, use the values 41.6 KSLOC and 4.16 KSLOC for the expected size of function blocks and sub functions respectively. These values were presented by Britchner and Gaffney (1985) [9] as reasonable sizes for aerospace projects (real-time command and control systems).It has the disadvantage that it requires that one is well knowledgeable

on the software to be developed in order to decompose it to blocks and sub functions

2.4 Function Point Analysis

Function points allow the measurement of software size in standard units, independent of the underlying language in which the software is developed. Instead of counting the lines of code that make up a system, count the number of externals (inputs, outputs, inquiries, and interfaces) that make up the system.

There are five types of externals to count:

1. External inputs- data or control inputs (input files, tables, forms, screens, messages, etc.) to the system

2. External outputs- data or control outputs from the system

3. external inquiries-- I/O queries which require a response (prompts, interrupts, calls, etc.)

4. External interfaces- libraries or programs which are passed into and out of the system (I/O routines, sorting procedures, math libraries, run-time libraries, etc.)

5. Internal data files - groupings of data stored internally in the system (entities, internal control files, directories)

Apply these steps to calculate the size of a project:

1. Count or estimate all the occurrences of each type of external.

2. Assign each occurrence a complexity weight

3. Multiply each occurrence by its complexity weight, and total the results to obtain a function count.

Table 1: Complexity weights are

Description	Low	medium	High
external inputs	3	4	6
external outputs	4	5	7
external inquiries	3	4	6
external interfaces	5	7	10
internal data files	7	10	15

4 Multiply the function count by a value adjustment multiplier (VAM) to obtain the function point count.

$$\text{VAM}=\sum_{i=1}^{14} v_i \times 0.01 + \ 0.065 \qquad (6)$$

Where V_I is a rating of 0 to 5 for each of the following fourteen factors (i). The rating

reflects how each factor affects the software size.

1. Data communications

2. Distributed functions

3. Performance

4. Heavily used operational configuration

5. Transaction rate

6. On-line data entry

7. Design for end user efficiency

8. On-line update of logical internal files

9. Complex processing

10. Reusability of system code

11. Installation ease

12. Operational ease

13. Multiple sites

14. Ease of change

Assign the rating of 0 to 5 according to these values:

0 - factor not present or has no influence

1 - Insignificant influence

2 - Moderate influence

3 - Average influence

4 - Significant influence

5 - Strong influence

Function point analysis is extremely useful for the transaction processing systems that make up the majority of MIS projects. However, it does not provide an accurate estimate when dealing with command and control software, switching software, systems software or embedded systems.

3. Pairwise comparison matrix size estimation framework

In 1977, Saaty [3] argued that like a physical measurement scale with a zero and a unit to apply to objects, we can also derive accurate and reliable relative scales that do not have a zero or a unit by using our understanding and judgments that are the most fundamental determinants of why we want to measure anything [3]. He showed that AHP (Analytic Hierarchy Process) can be used to solve the Multi Criteria Decision Making (MCDM) problem. MCDM is referring to making decision in the presence of multiple criteria. Zahedi [4] summarizes the original AHP procedure by Saaty [3] into four phases:

1. Break the decision problem into a hierarchy of interrelated problems.
2. Provide the matrix data for pair wise comparison of the decision elements.
3. Using Eigenvector Method (EV) as a prioritization method.
4. Aggregate the relative weights of the decision

The most integral part of Analytical Hierarchical Process (AHP) is a pairwise comparison matrix. A pairwise comparison matrix provides a formal, systematic means of extracting, combining, and capturing expert judgments and their relationship to analogous reference data (historical data) [5]. Bozoki gives a more detailed description of his approach in his paper [6]. Using a pairwise comparison matrix to estimate software size in incremental development requires an expert's judgment on increments' relative size compared to one another. The effectiveness of this approach is supported by experiments that indicate that the human mind is better at identifying relative differences than at estimating absolute values [83]. Many adjustments have been made to the original AHP procedure that was proposed by Saaty [3] in 1977 by different authors. This research work therefore intends to use some of these modifications and further adapt it in order to fit

it in the estimation of software size and cost in incremental development. Analytical Hierarchy Process (AHP) therefore

3.1: Step1: Rank the increments

Rank the increments from the largest to the smallest. An expert ranks the increments to be developed starting with the one he perceives to be the largest to the smallest. Though this step is not mandatory it lessens work during comparison.

3.2 Step 2: Create a pairwise matrix

Pairwise matrix begins with creating a judgment matrix to solve the sizing problem in incremental software development. Creating a judgment matrix involves creating an *n* x *n* matrix ($\boldsymbol{A^{n\times n} = a_{ij}}$), where *n* is the number of increments to be developed in order to deliver the whole software. Note that in incremental software development the total software is delivered in a series of increments and demonstrated in equation (7)

Total software=$\sum_{i=1}^{n} increments_i$ (7)

(The assumption in equation (7) is that all the increments are independent and the gluing code i.e. extra code for integration has been factored in. This assumption will be addressed fully in this thesis)

Element, a_{ij} in the matrix is an estimate of the relative size of increment i with respect to increment *j*, i.e.

$$a_{ij} = {size_i}/{size_j} \quad (8)$$

Judgment matrix should have two critical properties

1. $a_{ij} = {1}/{a_{ij}}$ which means that increment i is a_{ij} times bigger than increment *j*, then increment *j* is ${1}/{a_{ij}}$ times smaller than entity i;
2. An increment is same size as itself meaning that all diagonal elements a_{ii} = 1.

The implication of these properties is that an expert is only required to fill the upper or lower triangle of the judgment matrix. For example, see Table 2, which is a judgment matrix with estimates of the relative software size of four increments. The values in Table 2 row1 indicate that increment I is a_{ij} as big as increment, a_{ik} bigger than increment K, and a_{il} bigger than increment L. The remaining entries are interpreted in the same manner.

Taking an example of four increments that are to be developed to deliver the whole software namely increment I, increment J, increment K, increment L we create a pairwise comparison matrix of n=4 (the matrix is depicted here as a table with the sole purpose of increasing understanding and making comparison concept clear to all).

Create matrix consisting of n increments in this case four increments and an expert fills the elements according to his or her perceived relative size of one increment in respect to the other.

Table 2: Partially filled pairwise comparison matrix showing relative sizes of four increments

	Increment I	Increment J	Increment K	Increment L
Increment I		a_{ij}	a_{ik}	a_{il}
Increment J			a_{jk}	a_{jl}
Increment K				a_{kl}
Increment L				

Table 2 represents a partially filled matrix $\begin{bmatrix} \boldsymbol{a_{ij}} & \boldsymbol{a_{ik}} & \boldsymbol{a_{il}} \\ & \boldsymbol{a_{jk}} & \boldsymbol{a_{jl}} \\ & & \boldsymbol{a_{kl}} \end{bmatrix}$

Note that the expert when filling the judgment matrix is guided by the scale developed by Saaty [3] as shown in Table 3. The explanation and definition column of Table 3 are adapted to fit this work as shown in Table 3

Table 3: Modified Saaty scale

Intensity of importance	Definition	Explanation
1	Equal size between the increments	Two increments are equal in size
2	Weak or slight size advantage of one increment over the other	judgment slightly favor one increment over another
3	Moderate size advantage of one increment over the other	judgment moderately favor one increment over another
4	Moderate plus	
5	Strong size advantage of one increment over the other	judgment strongly favor one increment over another
6	Strong plus	
7	Very strong or demonstrated importance	An increment is favored very strongly over another
8	Very, very strong	
9	Extreme importance	The evidence favoring one increment over another is of the highest possible order
1.1–1.9	When increments are very close a decimal is added to 1 to show their difference as appropriate	

Applying condition two of the judgment matrix i.e. an increment compared to itself is the same in terms of size. The diagonal of the matrix represented in Table 4 is filled making the upper part of the matrix fully filled leaving only the lower part (Table 4).

Table 4: Partially filled matrix where condition2 is applied

	Increment I	Increment J	Increment K	Increment L
Increment I	1	a_{ij}	a_{ik}	a_{il}
Increment J		1	a_{jk}	a_{jl}
Increment K			1	a_{kl}
Increment L				1

Table 4 represents partially filled matrix $\begin{bmatrix} 1 & a_{ij} & a_{ik} & a_{il} \\ & 1 & a_{jk} & a_{jl} \\ & & 1 & a_k \\ & & & 1 \end{bmatrix}$

Using the first condition of the judgment matrix i.e. $a_{ij} = 1/a_{ij}$ which means that entity i is a_{ij} times bigger than entity j, then entity j is $1/a_{ij}$ times smaller than entity i;

We can now fully fill the matrix i.e. the lower part of the matrix is filled by considering the reciprocal condition.

Table 5: Fully filled matrix where the two conditions have been applied

	Increment I	Increment J	Increment K	Increment L
Increment I	1	a_{ij}	a_{ik}	a_{il}
Increment J	$1/a_{ij}$	1	a_{jk}	a_{jl}
Increment K	$1/a_{ik}$	$1/a_{jk}$	1	a_{kl}
Increment L	$1/a_{il}$	$1/a_{jl}$	$1/a_{jk}$	1

Table 5 represents the matrix $\begin{bmatrix} 1 & a_{ij} & a_{ik} & a_{il} \\ \frac{1}{a_{ij}} & 1 & a_{jk} & a_{jl} \\ \frac{1}{a_{ik}} & \frac{1}{a_{jk}} & 1 & a_k \\ \frac{1}{a_{il}} & \frac{1}{a_{jl}} & \frac{1}{a_{jk}} & 1 \end{bmatrix}$

3.3 Step 3: Extract ranking vectors

Now we have a fully filled matrix that gives us relative sizes of the increments that we want to develop in order to deliver the entire software. The judgment matrix is interpreted that each column yields a different ranking vector for the purpose of determining the relative size of the four increments. Each vector is normalized such that the increment that corresponds to itself (the diagonal elements) is always 1, and it is the reference increment against which all comparisons in the same column are made. Therefore, column 1 indicates that increment J is $1/a_{ij}$ as big as increment I; increment K is $1/a_{ik}$ of the size of increment I and increment L is $1/a_{il}$ of increment I. Each column can be interpreted in this manner. In this case the ranking vectors are $\begin{bmatrix} 1 \\ \frac{1}{a_{ij}} \\ \frac{1}{a_{ik}} \\ \frac{1}{a_{il}} \end{bmatrix}$ this is the ranking vector from column one the respective ranking vector generated by column two, three and four are $\begin{bmatrix} a_{ij} \\ 1 \\ \frac{1}{a_{ik}} \\ \frac{1}{a_{jl}} \end{bmatrix}$ $\begin{bmatrix} a_{ik} \\ a_{jk} \\ 1 \\ \frac{1}{a_{jk}} \end{bmatrix}$ $\begin{bmatrix} a_{il} \\ a_{jl} \\ a_{kl} \\ 1 \end{bmatrix}$ respectively. A matrix of n increments will yield four n ranking vectors. If the ranking vectors are different which is always the case then it means that there is more estimation uncertainty and extra estimation needs to be done in order to come up with one ranking vector called Priority vector. A special case exists when a judgment matrix is perfectly consistent. This occurs when $a_{ij} \times a_{jk} = a_{ik}$ all for i, j, k

If judgment matrix is not consistent then we need to go to step 4 otherwise we skip to step 5

3.4 Step 4: Compute priority vector

There are some methodologies that have been proposed by different authors to tackle this problem. The methods are reviewed as follows without looking at the strengths and weaknesses of each.

3.4.1 Eigen value methodology

Let A be a $n \times n$ matrix. A number λ is known as Eigen value of A if there exists a non-zero vector v such that

$$Av=\lambda v \quad (9)$$

In this case, vector v is called an eigenvector of vector A corresponding to Eigen value λ Eigen values and eigenvectors are defined only for square matrices i.e. the number of rows must be equal to the number of columns in the matrix i.e. $n \times n$ matrix hence works well with the comparison matrix which must be a square matrix. For a $n \times n$ matrix there are n Eigen values for the matrix. In order to solve for priority vector using this method we must calculate the Eigenvector corresponding to highest Eigen value of the judgment matrix that is

$$Av=\lambda_{max} v \quad (10)$$

Where λ_{max} is the highest Eigen value of the judgment matrix

The next step is to test if the pair wise comparison matrix is consistent this is achieved by calculating consistency index (C.I).

$$C.I=\frac{\lambda_{max}-n}{n-1} \quad (11)$$

Where λ_{max} is the highest Eigen value of the judgment matrix and n is the number of rows or column.

In accordance with Saaty [3] defined the matrix as consistent when C.I<0.1 If C.I.>0.1.If C.I falls outside this range he suggested an algorithm about repeating questions to correct the matrix until it became consistent.

3.4.2 Normalization of the Row Sum (NRS)

NRS sums up the elements in each row and normalizes by dividing each sum by the total of all the sums, thus the results now add up to unity. NRS has the form:

$$\boldsymbol{a_i'}=\sum_{j=1}^{n} \boldsymbol{a_{ij}}\ i = 1,2 \dots n \qquad (12)$$

$$\boldsymbol{w_i}=\frac{\boldsymbol{a_i'}}{\sum_{i=1}^{n} \boldsymbol{a_i'}}\ i = 1,2 \dots n \qquad (13)$$

3.4.3 Arithmetic Mean of Normalized Columns (AMNC)

AMNC was also called the Additive Normalization method in [10]. The new name is relatively clear, in that it describes its calculation process. Each element in *A* is divided by the sum of each column in A, and then the mean of each row is taken as the priority.

$$a'_{ij}=\frac{a_{ij}}{\sum_{i=1}^{n} a_{ij}}\ j, i = 1,2 \dots n \qquad (14)$$

$$w_i=\frac{1}{n}\sum_{i=1}^{n} a_j'\ i = 1,2 \dots n \qquad (15)$$

3.4.4 Normalization of Reciprocals of Column Sum (NRCS)

NRCS takes the sum of the elements in each column, forms the reciprocals of these sums, and then normalizes so that these numbers add up to unity, e.g. to divide each reciprocal by the sum of the reciprocals. It is in this form:

$$a_j^i=\frac{1}{\sum_{j=1}^{n} a_{ij}}\ 1,2 \dots n \qquad (16)$$

$$w_i=\frac{a_i'}{\sum_{i=1}^{n} a_i'}\ i = 1,2..n \qquad (17)$$

Without looking at the strength and weaknesses of each methodology that is beyond the scope of this research the thesis chooses to use Geometric Mean because of its simplicity.

3.4.5 Geometric mean method

The geometric mean is computed as

$$x_i=\prod_{j=1}^{n} a_{ij}^{\frac{1}{n}} \qquad (18)$$

Where $n = number\ of\ increments$ and Element, a_{ij} in the matrix is an estimate of the relative size of increment i with respect to increment j

This when computed for n rows for a matrix of $n \times n$ will yield vector $x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}$ for the matrix in Table 5 we will compute the values of vector x as shown in equation (19-22)

$$x_i=\left(1 \times a_{ij} \times a_{ik} \times a_{il}\right)^{\frac{1}{4}} \qquad (19)$$

$$x_j=\left(\frac{1}{a_{ij}} \times 1 \times a_{jk} \times a_{jl}\right)^{\frac{1}{4}} \qquad (20)$$

$$x_k=\left(\frac{1}{a_{ik}} \times \frac{1}{a_{jk}} \times 1 \times a_{kl}\right)^{\frac{1}{4}} \qquad (21)$$

$$x_l=\left(\frac{1}{a_{il}} \times \frac{1}{a_{jl}} \times \frac{1}{a_{jk}} \times 1\right)^{\frac{1}{4}} \qquad (22)$$

This will yield a priority vector of $x = \begin{bmatrix} x_i \\ x_j \\ x_k \\ x_l \end{bmatrix}$

3.5 Step 5: Factor in Historical analogy

It is at this point this research work proposes two adaptations to the methodology to apply to the problem of software sizing in incremental software development.

1. Case where there is only one historical analogy (reference).

2. Case where more than two historical analogies exist.

3.5.1 Case 1: One Historical Analogy

This is applicable when only one reference analogy exists i.e. only one reference historical analogy for a given increment can be found. The first step in this case is to calculate m (multiplier) as shown

$$m=\frac{size_{ref}}{x_{ref}} \qquad (23)$$

where ref is the increment which a historical reference exists

The next step is to calculate size of all increments using m according to equation (24)

$$\begin{bmatrix} size_i \\ size_j \\ size_k \\ size_l \end{bmatrix}=m \times x = \begin{bmatrix} m \times x_i \\ m \times x_j \\ m \times x_k \\ m \times x_l \end{bmatrix} \qquad (24)$$

3.5.2 Case 2: This occurs where more than two historical analogies exist

When only one historical analogy exists it becomes easy to compute the respective sizes of all increments. A complication therefore arises when more than one historical

reference exists because any reference picked will yield different estimate for a given increment .This work therefore proposes the use of Beta distribution to solve this problem

In order to use Beta distribution we will need to have the following values

1. Optimum estimate(OE)
2. Least likely estimate(LLE)
3. Expected estimate(ES)

Optimum estimate(OE)= the largest estimate for given increment generated by a given historical analogy (reference) this essentially means that the actual size of this increment is expected not to exceed this estimate.

Least likely estimate=the smallest estimate for given increment generated by given historical analogy (reference) this implies that the actual size of this estimate is not expected to go below this estimate.

Expected estimate= average of the estimates for given increments lying between least likely estimate and most likely estimate it essentially means that the actual size of the increment has the highest probability of falling here.

In our earlier example lets now assume that references exist for all the increments it is therefore required that we calculate multiplier generated by each reference as shown in equation (25-28).

For increment I the multiplier generated by its reference is

$$m=\frac{size_{ref}}{x_{ref}} \quad (25)$$

For increment J the multiplier generated by its reference is $$m_j=\frac{size_{j_{ref}}}{x_j} \quad (26)$$

For increments K the multiplier generated by its reference is

$$m_k=\frac{size_{k_{ref}}}{x_k} \quad (27)$$

For increments L the multiplier generated by its reference is

$$m_l=\frac{size_{l_{ref}}}{x_l} \quad (28)$$

Each multiplier will yield different estimate for a given increment and so the challenge will be to know which estimate is the closest to the actual size of the increment. It is for this reason that this thesis proposes the use of Beta distribution to solve this problem. How to compute the estimates are shown in Table 6.

Table 6: Estimates of each increment

Multiplier used	Increment I estimates	Increment J estimates	Increment K estimates	Increment L estimates
Estimate 1	$m_i \times x_i$	$m_j \times x_i$	$m_k \times x_i$	$m_l \times x_i$
Estimate 2	$m_i \times x_j$	$m_j \times x_j$	$m_k \times x_j$	$m_l \times x_j$
Estimate 3	$m_i \times x_k$	$m_j \times x_k$	$m_k \times x_k$	$m_l \times x_k$
Estimate 4	$m_i \times x_l$	$m_j \times x_l$	$m_k \times x_l$	$m_k \times x_l$

For each increment its size estimate are in its column .For instance increment I its size estimates are $\begin{bmatrix} m_i \times x_i \\ m_i \times x_j \\ m_i \times x_k \\ m_i \times x_l \end{bmatrix}$ optimum estimate is picked as the highest estimate generated from the column and the lowest estimate is picked as least likely estimate, the expected estimate is taken as the average of the estimates lying between optimum estimate and least likely estimate because most weight tend to lie here. In the table above let's take Increment I as our example and picking $m_i \times x_i$ as the optimum estimate, $m_i \times x_l$ as the least likely estimate then the expected estimate will be

Expected estimate= $$\frac{sum\ of remaining\ \ estimates\ \ lying\ \ between\ \ OE\ and\ \ LLE}{number\ \ of\ \ estimates\ \ lying\ \ between\ \ LLE\ and\ \ OE} \quad (29)$$

In this case expected estimate is estimated as follows

$$ES=\frac{m_i \times x_j + m_i \times x_k}{2} \quad (30)$$

Where LLE=$m_i \times x_i$ and OE=$m_i \times x_l$

Increment I size is therefore estimated as

$$size_i = \frac{OE+4ES+LLE}{6} \quad (31)$$

The size estimates of other increments are computed in the same way. It is worth noting that the example used here has only four different estimates for given increment this could be more or less depending on number of increments to be developed to deliver a system and the number of Historical analogies (references available). A special case exist when only two references exist and hence results in two estimates for given Increment. If this scenario comes up the average of the two estimates is taken the estimate of the increment.

3.6 Step 6: Calculate the total software size

At this point we have size estimates for all the increments and the challenge therefore is to compute the size of the whole software i.e. total size denoted here as S_i.Because the total software is being developed incrementally there is substantial code that is written to glue the new increment to the already developed increment. The term used for this in the COCOMO models is breakage, because some of the existing code and design has to be mended to fit in a new increment. This

research will refer to gluing code as incremental breakage therefore the projected size of a given increment factoring in the incremental breakage is estimated according to equation (32)

$$s_i = size_i + csize_{i-1} \quad (32)$$

Where $csize_{i-1}$ is the increment size with breakage code factored in , $size_i$ is the initial increment size estimate computed from reference analogy according to equation (31) or beta distribution depending on the case that arises from step 5.The parameter *c* reflects the incremental *breakage* (or overhead) associated with the previous increment which is expressed in percentage. Kan asserted that 20% of the added code in staged and incremental releases of a product goes into changing the previous code [7]. Cusumano and Selby reported that features may change by over 30% as a direct result of learning during a single iteration [18]. In a recent paper the authors argued that the incremental integration breakage can be expected to lie in a range from 5% to 30%.. If *c* has a value of *0.15*, it corresponds to *15% breakage*. In order to simplify the discussion, it is assumed that all the code of $size_i$ is developed from scratch i.e. no code reuse is not taken into consideration the case of code reuse is beyond the scope of this research. Therefore the total system size T_s is computed according to equation (33)

$$T_s = \sum_{i=1}^{n} size_i + c \sum_{i=1}^{n} size_{n-1} \quad \text{for } n = 1,2 \ldots . n \quad \text{and} \quad i = 1,2 \ldots . n \quad (33)$$

Considering equation (32) equation (33) simplifies to

$$T_S = \sum_{i=i}^{n} s_i \quad (34)$$

Where T_s is the total software size,s_i is the net increment size as defined in equation (34)

Compared to the LOC and Function point methodology ,the pairwise framework has the advantage of combining both user judgment, experience and historical analogy to generate size estimates which are superior .The two methods only use user experience neglecting the importance of historical analogy which is useful in predicting size of future or current projects.

4.0 METHODOLOGY

4.1 Data and data collection form

The main focus of the research was to find out if the pairwise comparison matrix framework produces superior size estimates of a software compared to the currently popular existing Putman's Loc estimation [12] and to prove if there is any direct relationship between accuracy of the size estimates produced by the model used in the estimation procedure and the fact that software is delivered on time or not i.e. if software project meets deadline. In order to accomplish these objectives the following information was among the data gathered

1. Language used to develop a software
2. Start and end date of the development of the software
3. Number of developers allocated for the software project
4. The estimation model used to estimate size
5. Size estimate generated by the model
6. The actual size of the software upon completion

In order to capture all these and more information two questionnaires were designed (The choice of the questionnaire was appropriate because it allowed for the forms to be sent to the project managers earlier which enabled them to familiarize themselves with the content before an in person meeting was conducted to guide them as they filled out the forms. Second questionnaire was in form of a table to represent unfilled matrix. Experts' filled the form according to their judgment capturing relative sizes of different increments that were developed to deliver the whole software. The use of more than one expert was vital in checking consistency and have platform for comparison.

4.2 Data source

The source for the project data for this research was from JJpeople firm which is Software firm that develops wide range of softwares using JAVA and also serves training ground for young software developers who are interested in the same language. JJpeople has its offices in Vancouver, London and its African branch in Nairobi. Use of this firm's data conveyed several advantages to this research. First since one of the main aims of this project is to establish the size of the software the firm exclusive use of object oriented language(JAVA) made the counting of lines of codes (LOC) relatively easy. Secondly the firm develops wide range of application making the data diverse this broadened the level of interest in the results, as opposed to, say, a database composed of only one type of application. The firm also indicated that they have used incremental development in some of their application therefore fitting well with this research. The willingness of the management to provide data and advice on the framework was also good for this research.

4.3 Data-Collection Procedure and project attributes

For each of the projects, there was an in-person meeting with the project manager who filled out the forms. There were two main purposes to this labor intensive approach. The first goal was to discuss each of the questions to ensure that it is well understood and that each of the managers would answer consistently. The second purpose was to impress upon the managers the importance of their participation in this work.

Projects selected possessed two attributes: First, they were small to medium in size. The average project size in this study is just under 60 KSLOC .The project selected from the database were also fairly recent with the oldest developed in 2003.All projects except project number eight had not used any previous code i.e. there was no code reuse. Project number eight was just selected because it was exclusively developed incrementally therefore fitted well with this research

4.4 Data Analysis

The focus of this research was to check how close the size estimates generated by the pairwise size estimation framework were close to the actual size. Therefore an error analysis was done to check how the size estimates from the pairwise estimation framework deviates from the actual size. The focus therefore was on the degree to which the pairwise framework model's estimated size (MM_{EST} matches the actual size (MM_{ACT}). If the models were perfect, then for

every project $MM_{EST} = MM_{ACT}$ clearly, this was rarely, if ever, the case. A simple analysis approach was to look at the difference between MM_{EST} and MM_{ACT}. The problem with this absolute error approach was that the importance of the size of the error varied with project size. For example, on an 80KLOC, an absolute error of 10KLOC seemed likely to cause serious project disruption in terms of staffing, whereas the same error on a 1000KLOC project seemed much less of a problem. In light of this, Boehm [13] and others have recommended a percentage error test, as follows:

$$\text{PercentageError} = \left| \frac{MM_{EST} - MM_{ACT}}{MM_{ACT}} \right| \quad (36)$$

This test eliminated the problem caused by project size and better reflected the impact of any error. However, the analysis in this research concentrated on the pairwise comparison framework estimates' average performance over the entire set of projects. Errors were of two types: underestimates, where $MM_{EST} < MM_{ACT}$; and overestimates, where $MM_{EST} > MM_{ACT}$. Both of these errors can have serious impacts on projects. Large underestimates will cause the project to be understaffed, and as the deadline approaches, project management will be tempted to add new staff members. These results in a Phenomenon known as Brooks' law: "Adding manpower to a late software project makes it later" [14]. Otherwise productive staffs are assigned to teaching the new team members, and with this, cost and schedule goals slip even further. Overestimates can also be costly in that staff members, noting the project slack, become less productive (Parkinson's Law: "Work expands to fill the time available for its completion") or add so-called "gold plating," defined as additional systems features that are not required by the user [13].

In light of the seriousness of both types of errors, overestimates and underestimates, Conte et al. [8] have suggested a magnitude of relative error, or MRE test, as follows:

$$\text{MRE} = \left| \frac{\frac{MM_{EST} - MM_{ACT}}{MM_{ACT}}}{N} \right| \quad (35)$$

Where N is the number of projects used in this research

By means of this test, the two types of errors do not cancel each other out when an average of multiple errors is taken, and therefore was used as the test in this research. Graphs were drawn to give good pictorial comparison between the actual size and the estimates from pairwise framework using EXCEL

5. RESULTS AND DISCUSSION

5.1 Project attributes

Projects selected possessed two attributes: First, they were small to medium in size. The average project size in this study is just under 600 SLOC .The project selected from the database were also fairly recent with the oldest developed in 2001.All projects except project number eight had not used any previous code i.e. there was no code reuse. Project number eight was just selected because it was exclusively developed incrementally therefore fitted well with this research.

Table 7 below compares the actual size of the softwares and the estimates generated by pairwise methodologies. The focus of this research is on the degree to which the pairwise comparison methodology estimates compare to the actual size.

Table 7 Showing error margin between actual size and estimates from pairwise methodology

Project No	Actual size (KLOC)	Estimate of Pairwise methodology (KLOC)	Absolute error margin (KLOC)	% error	MRE
1	42.1	39.64	2.46	4.75	0.0475
2	72.12	69.49	2.63	3.65	0.0365
3	29.52	28.23	1.29	3.01	0.0301
4	42.82	43.11	0.29	0.68	0.0068
5	16.73	17.22	0.49	2.93	0.0293
6	196.22	196.41	0.19	0.097	0.00097
7	67.31	70.10	2.79	4.15	0.0415
8	52.76	49.44	3.32	6.29	0.0629
9	46.92	47.01	0.09	0.19	0.0019
10	37.54	38.09	0.55	1.47	0.0147
11	14.723	13.967	0.756	5.13	0.0513
12	24.666	24.061	0.605	2.453	0.02453
13	30.464	29.000	1.464	4.806	0.04806
14	109.659	109.410	0.249	0.227	0.00227
Mean ($\bar{X}$)	55.968	55.36986			

If the estimates match the actual size perfectly then error margin was zero clearly this was not the case in any of the projects in table 7 .The error margin was computed to check the deviation of the pairwise size estimate methodology from the actual value. Errors were of two types: underestimates,

Where pairwise estimates $<$ actual size and overestimates, where pairwise estimates > actual size. Both of these errors can have serious impacts on projects. Large underestimates has serious impacts on projects. Large underestimates will cause the project to be understaffed, and as the deadline approaches, project management will be tempted to add new staff members. These results in a phenomenon known as Brook's law: "Adding manpower to a late software project makes it later" [6]. Otherwise productive staff is assigned to teaching the new team members, and with this, cost and schedule goals slip even further. Overestimates can also be costly in that staff members, noting the project slack, become less productive (Parkinson's Law: "Work expands to fill the time available for its completion") or add so-called "gold plating," defined as additional systems features that are not required by the user [5]. In light of the seriousness of both types of errors, overestimates and underestimates, Conte et al. [8] have suggested a magnitude of relative error, or MRE

$$\text{MRE} = \left| \frac{ACTUAL\ ESTIMATE - PAIRWISE\ ESTIMATE}{ACTUAL\ ESTIMATE} \right|$$

By means of this test, the two types of errors do not cancel each other out when an average of multiple errors is taken, and therefore was taken as the test used in this analysis.

This test eliminates the problem caused by project size and better reflects the impact of any error .From the error margin it was noted that percentage error ranged 6.29 to 0.097 which corresponds to MRE of 0.063 and 0.00097 respectively. It can also be noted that as the project get bigger the methodology tend to generate more accurate results. From the results shown in the table 7 it is clearly depicted that with real experience from the developer the estimates developed by pairwise methodology can result in more accurate estimates especially in large projects which are associated more with massive risk as compared to relatively smaller risks.

The respective means of actual size and estimates computed from pairwise comparison methodology were computed as shown in table 7 and the difference in their mean is 0.59814 representing 1.07% error in the estimates generated by pairwise methodology. This clearly shows that the method generates close estimates that can be relied upon by software managers in making major decisions prior to embarking on the process of software development

The values of the actual size of the projects, the size estimates generated by the pairwise comparison methodology and the error margin is shown in the bar chart below in order to give a clear impression of how the estimates are close. As clearly depicted in the bar chart in figure 2 the estimates generated by pairwise methodology and the actual size are so close. The error margin is also shown in the bar chart to give a clear impression of how close the estimates are to the actual value.

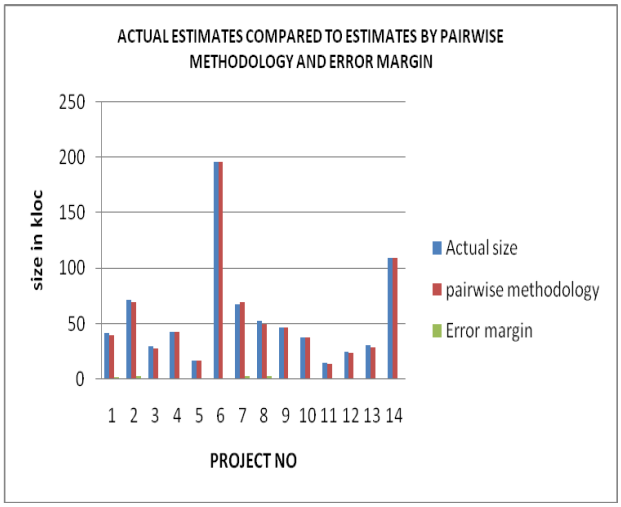

Figure 2: Bar chart showing comparison of actual size, estimates by pairwise methodology and error margin.

Putman's methodology of Loc estimation was also used to estimate the size of the fourteen projects and the results are shown in table 8.

Table 8 Comparing actual size versus estimates from Putman's methodology

Project No	Actual size (KLOC)	Estimate of Putman's methodology (KLOC)	Absolute error margin (KLOC)	% error	MRE
1	42.1	45.89	3.79	9.00	
2	72.12	76.87	4.75	6.59	
3	29.52	32.66	3.14	10.64	
4	42.82	45.45	2.63	6.14	
5	16.73	16.00	0.73	4.36	
6	196.22	200.43	4.21	2.15	
7	67.31	69.04	1.73	2.57	
8	52.76	50.99	1.77	3.35	
9	46.92	50.01	3.09	6.59	
10	37.54	38.99	1.45	3.86	
11	14.723	14.900	0.177	1.20	
12	24.666	27.000	2.334	9.46	
13	30.464	34.010	3.546	11.64	
14	109.659	115.879	6.22	5.67	
MEAN $(\overline{X})$	55.968	58.43707			

The error margin was computed to check the deviation of the estimates generated by the Putman's methodology from the actual size. From the error margin it was noted that percentage error ranged 11.64 to 1.20 with 11.64 being the highest error percentage in deviation of the estimate from the actual size.

In order to show clearly how the estimates deviate from the actual size a calculation of mean is computed in table 8 and the difference in the mean is 2.46907 which represent 4.41%. Error in the deviation is shown

The values of the actual size of the projects, the size estimates generated by the Putman's methodology and the error margin is shown in the bar chart in figure 3 in order to give a clear impression of how the estimates compare to the actual size.

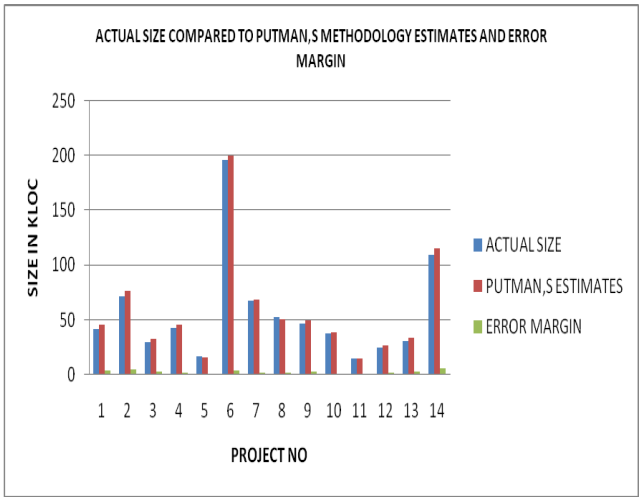

Figure 3: Bar chart showing comparison of actual size, estimates by pairwise methodology and error margin.

6. CONCLUSION

Compared to Putman's methodology the pairwise methodology was superior in generating estimates of size in incremental development of software. As noted in table 7 deviation of the mean of estimates generated by pairwise comparison methodology deviated from the actual size by 1.07% while those of Putman's methodology deviated by4.41% this therefore confirms the superiority of the pairwise methodology. The table 9 and bar chart 4 shows the pairwise error margin as compared to those of Putman's methodology

Table 9: comparison of error margins

Project number	Error margin (Putman's method)	Error margin (pairwise method)
1	3.79	2.46
2	4.75	2.63
3	3.14	1.29
4	2.63	0.29
5	0.73	0.49
6	4.21	0.19
7	1.73	2.79
8	1.77	3.32
9	3.09	0.09
10	1.45	0.55
11	0.177	0.756
12	2.334	0.605
13	3.546	1.464
14	6.22	0.249

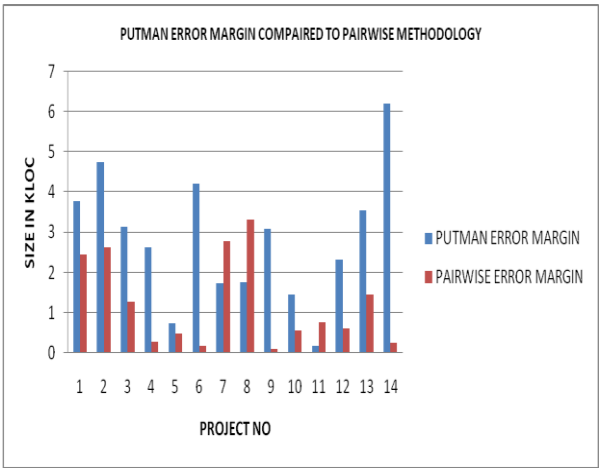

Figure 4: Bar chart comparing error margins generated by the two methods

7. ACKNOWLEDGMENTS

The authors would like to thank Dr Stephen Kimani and Calvins Otieno for their contribution to the ideas and proofreading of this paper.

8. REFERENCES

[1] Dalcher, D.: 'Falling down is part of growing up; the study of failure and the software engineering community'. Proc. 7th SEI Education in Software Engineering Conf., San Antonio, Texas (Springer-Verlag, New York, 1994), pp. 489–496 [6]Lambert, J. "A Software Sizing Model," *Journal of Parametrics*, Vol. Vi, 1986, pp75-87.

[2] Standish Group. 2000. 'Chaos 2000' (Standish, Dennis, Mass, 2000)

[3] Saaty, T. "A Scaling method for Priorities in a Hierarchical Structure". .J. Math. Psychology Vol. 15 1977, p 234-281.

[4] Zahedi, F., *The Analytic Hierarchy Process- A Survey of the Method and its Application.* Interfaces, 1986. **16**: p. 96-108.

[5] Saaty, T. *The Analytic Hierarchy Process*, McGraw-Hill, New York, NY: 1980.

[6] Bozoki, G. "Software Size Estimator (SSE)," Centre National d'Etudes Spatiales (CNES), Toulouse, France, June 1986.

[7] Miranda, E. "Improving Subjective Estimates Using Paired Comparisons," *IEEE Software*, Jan/Feb 2001.Proceedings of the 10th International Symposium on Software Metrics (METRICS'04) Saaty, T. "A Scaling method for Priorities in a Hierarchical Structure". .J. Math. Psychology Vol. 15 1977, p 234-281.

[8] Benediktsson, O., and Dalcher, D.: 'Effort estimation in incremental software development', IEE Proc., Softw., 2003, 150, (6), pp. 351–357

[9] A.J. Albrecht and J. Gaffney, "Software function, source lines of code, and development effort prediction: **A** software science validation", IEEE *Transactions on Software Engineering*, Vo1.9, No.6, pp.639-648, Nov.1983.

[10] Calibrating function point backfiring conversion ratios using neuro-fuzzy technique Justin wong, Danny ho, Luiz fernando capretz.