# A METRICS -BASED MODEL FOR ESTIMATING THE MAINTENANCE EFFORT OF PYTHON SOFTWARE

Catherine Wambui Mukunga[1], John Gichuki Ndia[2] and Geoffrey Mariga Wambugu[3]

[1, 2, 3]School of Computing and Information Technology, Murang'a University of Technology, Murang'a Kenya,

***ABSTRACT***

*Software project management includes a substantial area for estimating software maintenance effort. Estimation of software maintenance effort improves the overall performance and efficiency of software. The Constructive Cost Model (COCOMO) and other effort estimation models are mentioned in literature but are inappropriate for Python programming language. This research aimed to modify the Constructive Cost Model (COCOMO II) by considering a range of Python maintenance effort influencing factors to get estimations and incorporated size and complexity metrics to estimate maintenance effort. A within-subjects experimental design was adopted and an experiment questionnaire was administered to forty subjects aiming to rate the maintainability of twenty Python programs. Data collected from the experiment questionnaire was analyzed using descriptive statistics. Metric values were collected using a developed metric tool. The subject ratings on software maintainability were correlated with the developed model's maintenance effort, a strong correlation of 0.610 was reported meaning that the model is valid.*

***KEYWORDS***

*Software Maintenance, Software Maintenance effort, Software Maintenance estimation model, Python Software, Complexity metrics and size metrics*

## 1. INTRODUCTION

Software maintenance effort estimation still remains a challenge for most software development teams since the current methodologies are still unable to deal with the current software estimation challenges that exist today. Traditional software maintenance effort can be estimated using a variety of models such as the Constructive Cost Model (COCOMO), the Software Life Cycle Management model (SLIM), Function Point, and others but more work needs to be done on developing models that can accommodate programs from new programming languages. The most well-known software cost estimation model is the COCOMO, or generalized Constructive Cost Model, established by [1].The COCOMO II model is the latest update. [2] remark that the COCOMO II model is a regression based software cost estimation model which is the most popular among the traditional models. Several effort estimation models are reported in the literature but are not suitable for Python software. A study by [3] advise on the need to revise the COCOMO model due to inaccuracies, in addition, the COCOMO II model considers the same software development cost drivers to estimate maintenance effort which does not reflect the true picture of the software maintenance phase.

A software metric refers to a measurement of a software's attributes or specifications [4]. Software metrics can be used to measure product output, project estimation, process progress, and process improvements as explained by [5]. Software complexity is described by [6] to be the

challenges and difficulties in software reuse, software comprehension, and software maintenance. Software Complexity metrics have been defined by [6], [7], [8], [9], [10], [11], and [12]. [13] concluded that increasing complexity drives up adaptive maintenance costs. Software size has an impact on its complexity according to research conducted by [14]. [26] state that size metrics are a major factor in the determination of successiful software projects. Additionally, the study by [15] suggests size metrics have a significant impact on project effort, duration, and productivity. One of the widely adopted size metrics is lines of code (LOC) which according to [13] is not effective since a statement may be spread over several lines. [27] did not consider lines of code (LOC) metric in the development of a generic effort estimation model explaining that the metric is language dependent.

Research by [16] presents Python to be a fast-growing programing language. [16] mentioned that Python had become popular due to its simplicity, learnability, and supportability. The most popular programming language overall, out of 100, is Python, according to the Importance of Being Earnest index (TIOBE) of April 2023. The TIOBE index helps in making decisions about what programming language to adopt in building a new software system.

This paper is structured as follows; section two is a discussion of software maintenance effort estimation models; section three discusses the proposed software maintenance effort estimation model; section four discusses how the proposed model was validated; section five is a discussion of the validation results, and section six is a discussion of the conclusion and future work.

## 2. LITERATURE REVIEW

### 2.1. Software Maintenance Effort Estimation Models

The maintenance cost estimation model proposed by [17] is considers the Annual Change Traffic (ACT) metric and operates on a fourth generation language environment. In addition, the model has incorporated several factors grouped into technical and non-technical factors such as internal complexity, quality of source code, Computer Aided Software Engineering tools and others.
[18] developed a model for estimating maintenance effort. The effort was expressed in person hours. The first step of developing the model involved identification of metrics that affect maintenance effort. A correlation coefficient analysis between the metrics and maintenance effort was done to determine the most effective metrics to predict adaptive maintenance effort.

The equation $E = 63 + .1\ DLOC$ was used to compute the maintenance effort. (1)

A maintenance cost estimation model on the basis of regression was developed by [19] for a large application.The work is approached in three phases; creation of data transparency, examining maintenance expenditure, and cost optimization. Effort and product factors are considered such as programming languages and software deviations from the actual result.

[20]Computed maintenance cost by implementing the intermediate COCOMO model. The researcher introduced new cost driver multipliers and concluded that additional multipliers and number of code lines increases the maintenance effort. To estimate the maintenance cost, the annual change of traffic (ACT) metric was introduced which consists the annual changes in the software source code. To compute ACT, NNL which stands for a count of new lines was added to a count of the modified lines and the result divided by a count of original lines.

[14] created the Software Maintenance Effort Model (SMPEEM), which calculates the volume of the maintenance function using function points. The Function point's measurement was used

in estimating the sizes of maintenance tasks. The model considers value adjustment factors to adjust the counted function points. The effort is expressed as

Effort = A * (Size) $^{B}$. (2)

A and B are coefficients introduced from the regression results. The model was validated using a survey method. The results confirmed that adjusted function points are a good measure to estimate the maintenance effort of a project.

According to [2], COCOMO II comprises of; the Application Composition Model which estimates effort at the first phase, the Early Design Model which implements the unadjusted function points to determine size, the Reuse model for computing the effort of reusable components and the post-architecture model. COCOMO II defines 17 cost drivers that are used with the Post Architecture model and are rated on a scale of very low to extra high. Precedentness, Development Flexibility, Architecture/Risk Resolution, Team Cohesion, and Process Maturity are the five scale drivers that affect how long a project takes to complete and which exponent is utilized in the Effort Equation.

COCOMO II post architecture model is given as $PM = A.(Size)1.01 + \sum_{=j=1}^{5} SFj. \prod_{i=1}^{17} EMi$(3)

B = 1.01+ 0.01×∑SFi

Where A = 2.45 [16]

The investigations of [15] resulted in the development of the SMEEM (software maintenance effort model). The volume of maintenance and value adjustment parameters that have an impact on story points for effort estimation are calculated by the model. The model's maintenance process is broken down into the computation of factor counts, story point assignments, story point adjustments, calculation of maintenance sizes, and finally calculation of maintenance durations. [15] assert that the paradigm is only applicable in contexts focused on extreme programming and agile development.

[21] sought to improve the COCOMO II model for estimating maintenance size and effort by incorporating characteristics not considered in the COCOMO model. Steps followed in coming up with the model included; analyzing existing literature to identify size metrics, validating the size metrics on the maintenance cost, performing a behavioural analysis to identify the relative significance of the factors, determining a maintenance sizing method, determining the effort model, performing expert judgement on the effort, collection of project data for model validation, testing hypothesis, calibrating the model and evaluating model performance. An experiment with students as subjects and C++ programs as experiment objects confirmed that deleted source lines of code (SLOC) was a determinant of maintenance task effort. This model is limited to real- time software and implements the cost drivers for the COCOMO II model to compute maintenance effort which could lead to unrealistic estimates.

[22] developed a Component-Based Software (CBS) model to estimate the maintenance cost. The model considered the development cost, the annual changes on the source code, and factors that influence the maintenance effort of component based software. Maintenance Cost Estimation is expressed as:

$$AME_{CBS} = ACT * CDT \prod_{i=1}^{n} W_i * F_i \quad (4)$$

Where
AMECBS is the Actual maintenance cost
CDT is the Component Development Cost
ACT are the annual source code changes
Wi is the ith weighting maintenance load
Fi is the Factors value
This model is only used for component- based software.

From the maintenance effort estimation models mentioned in the literature, none of the models is suitable for the Python language. The syntax of Python programs is unique from other Object Oriented based programs [16] and this disqualifies the existing models from precisely predicting the maintenance effort of Python programs. As a result of this realization, a maintenance effort estimation model for Python software would highly be desirable.

## 3. PROPOSED SOFTWARE MAINTENANCE EFFORT ESTIMATION MODEL

In this work, a metrics based model for estimating the software maintenance effort of Python programs was developed. The model consists of three inputs namely; size, complexity, cost drivers, and maintenance effort as the output which is presented using the equation

$$Maintenance\ Effort\ (PM) = (Size * Complexity) * \prod_{i=1}^{24} EM\ i \quad (5)$$

### 3.1. Size metrics

The proposed model considers the System Size metric (SSpy) which is computed by considering the sizes of individual classes in a Python program.
The SSpy metric is a consideration of the number of code blocks in several classes of a Python program. A code block is a collection of Python statements that belong to the same block or indent. The equation below was implemented to arrive at SSpy.

$$System\ size = no\ of\ Code\ Blocks\ in\ class\ a + no\ of\ Code\ Blocks\ in\ class\ b + no\ of\ Code\ Blocks\ in\ class\ c + \cdots + no\ of\ Code\ Blocks\ in\ class\ n \quad (6)$$

### 3.2. Complexity metric

In addition to Size, the proposed model also considered the complexity of the software. The proposed model considered the Weighted System Complexity (WSCpy) metric which is computed by considering the complexities of individual python classes.
The metric Weighted System Complexity (WSCpy) was arrived at by considering the metric Weighted Class Complexity (WCCpy). Weighted Class Complexity (WCCpy) is a consideration of the complexities of variables and methods in a class.

The equation below was implemented to arrive at WSCpy.

$WSC_{py} = WCC_{py}\ of\ class\ a + WCC_{py}\ of\ class\ b + \cdots + WCC_{py}\ of\ class\ n$, and expressed as

$$WSC_{py} = \sum_{i=1}^{n}(i) \quad (7)$$

### 3.3. Cost Drivers

In our previous work [23], an expert opinion survey was conducted to identify and rank the relevant Python maintenance influencing factors. In the expert opinion survey, Python programmers and project managers were required to rate the factors influencing the maintenance cost of python software. Based on their responses, a mean value for each factor was computed by dividing the sum of actual responses per factor by sum of expected responses per attribute and multiplying the value by 100 i.e. Sum of actual responses / sum of expected responses *100. The factors were then ranked on the basis of the mean values. The 24 ranked factors are presented in Table 1.

Table 1. Ranked software maintenance effort influencing factors.

| Ranking | Factor | Mean in % | Normalised mean |
|---|---|---|---|
| 1 | Code quality | 85.8 | 0.858 |
| 2 | Understandability | 85.71 | 0.8571 |
| 3 | Document Quality | 82.8 | 0.828 |
| 4 | Configuration Management Technology | 82.3 | 0.823 |
| 5 | Modern Programming Specifications | 80.95 | 0.8095 |
| 6 | Database Size | 80.95 | 0.8095 |
| 7 | Software Complexity | 80 | 0.8 |
| 8 | Staff ability and skills | 80 | 0.8 |
| 9 | Testing Quality | 79.52 | 0.7952 |
| 10 | Component Reusability | 78.57 | 0.7857 |
| 11 | Organization Maturity | 77.61 | 0.7761 |
| 12 | Maintenance Staff Ability | 77.14 | 0.7714 |
| 13 | Availability of maintainers | 75.2 | 0.752 |
| 14 | Technology Newness | 73.8 | 0.738 |
| 15 | Programming Style | 69.52 | 0.6952 |
| 16 | Number of maintainers | 69.4 | 0.694 |
| 17 | Hiring model of maintainers | 68.2 | 0.682 |
| 18 | Hardware Stability | 67.61 | 0.6761 |
| 19 | Location diversity of maintainers | 67 | 0.67 |
| 20 | Dependence on External Environment | 63.8 | 0.638 |
| 21 | Component Performance | 63.33 | 0.6333 |
| 22 | CASE Tools | 59.04 | 0.5904 |
| 23 | Application Type | 55.71 | 0.5571 |
| 24 | System Lifespan | 55.23 | 0.5523 |

### 3.4. Maintenance Effort

The proposed model computes maintenance effort. The proposed model's maintenance effort is expressed in Person-hours.

## 4. MODEL VALIDATION

### 4.1. Data Collection

#### 4.1.1. Base Metrics

The base metrics are computed once the Python source file is uploaded to the tool. The base metrics were the number of instance variables, the number of class variables, the number of instance methods, the number of static methods, the number of class methods, and the number of code blocks in a class. The tool user clicks the base metric tab to view base metrics for a Python file.

#### 4.1.2. Derived metrics

These were the metrics derived from the base metric. The derived metrics included; Weighted variable complexity (WVCpy), weighted method complexity (WMCpy), weighted class complexity (WCCpy), weighted system complexity (WSCpy), class size CSpy) and system size (SSpy).

The tool user clicks on the derived metrics tab to view the derived metrics of a Python file.
Metric values for the size and complexity of program one were collected using the PMMET tool as presented in figure 1. The program had a weighted system complexity of 12 and a system size of 16. The tool user clicks on the open button to upload a Python file for analysis. This is followed by clicking the run button to begin metrics computation. The tool user can now view base metrics. Derived metrics can be viewed upon clicking the derived metrics tab.

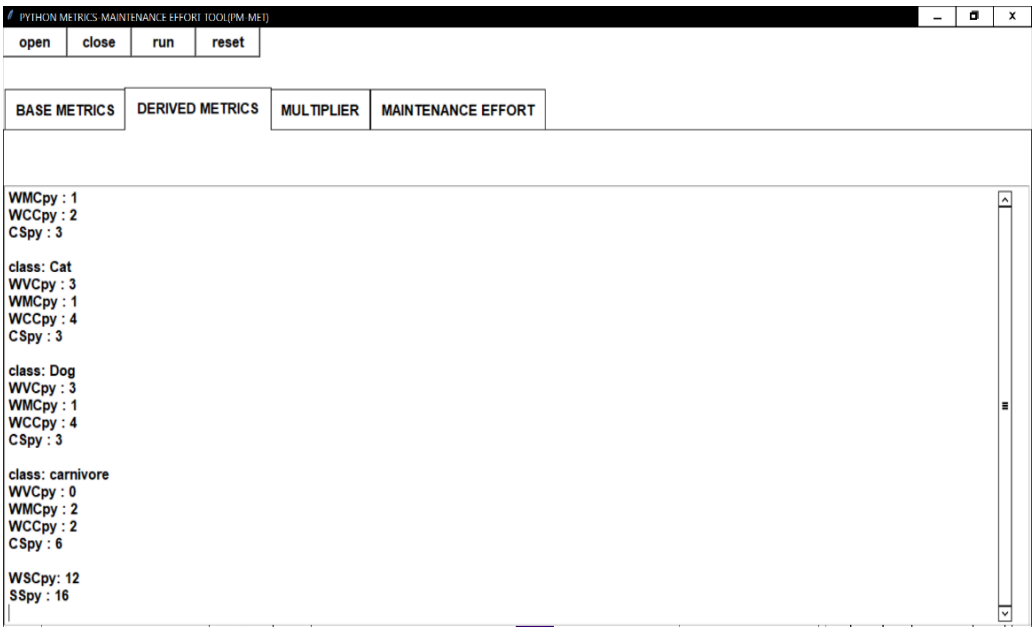


Figure 1. Metric values for a Python program generated by the PMMET tool.

### 4.1.3. Effort Adjustment Factor (EAF)

An Effort Adjustment Factor (EAF) value was computed by finding a product value of the factors influencing the maintenance effort of a Python program and the value recorded by the PMMET tool. The formula to compute the Effort Adjustment Factor value (EAF) is explained by [1]. There are 24 factors that influence the maintenance effort from Table 1. A software maintainer can select multiple factors and each factor has an effort multiplier value (weight). Order of occurrence of various cost drivers has a significant impact on overall efforts in project estimation. According to [25]variations to cost drivers in the COCOMO model contribute to an improved effort estimate. A highly ranked factor will contribute to a higher maintenance effort influencing power which will result in a higher maintenance effort value. A lowly ranked factor will contribute to a lower maintenance effort influencing power which will result in a lower maintenance effort value. The effort adjustment factor value (EAF) for Python program one is computed by the PMMET tool and presented in Figure 2.

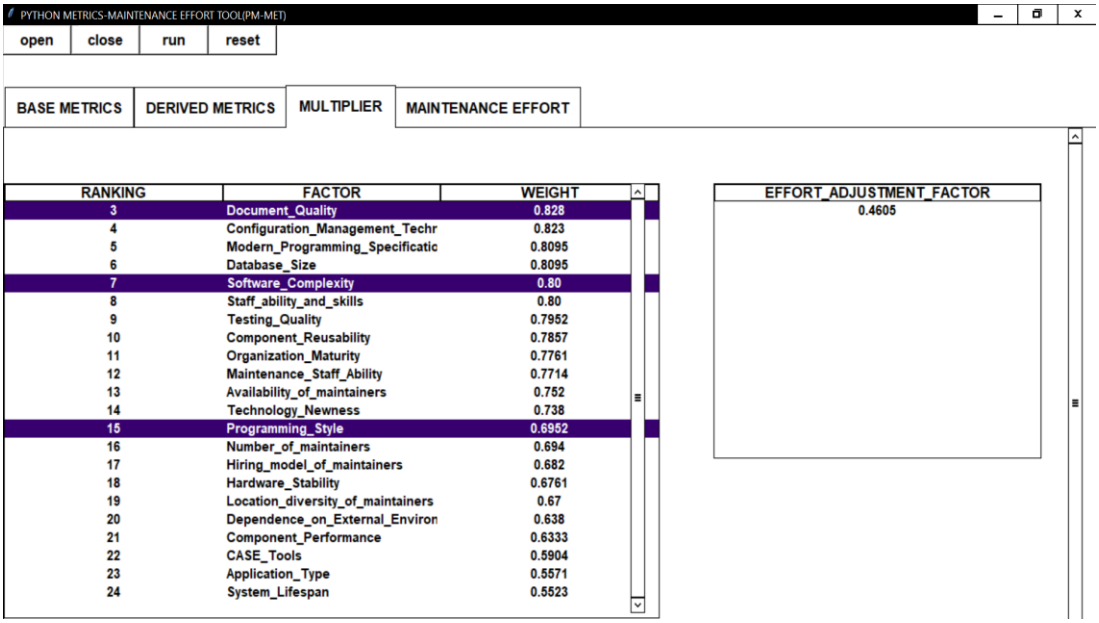


Figure 2 EAF value for a Python program generated by the PMMET tool.

Figure 2 presents the Effort Adjustment Factor (EAF) value computed from three factors influencing the maintenance effort of program one. The factors are selected by the software maintainer and in this example, document quality, software complexity, and programming style were selected. The weights of the factors were multiplied to get an EAF value of 0.4605.

### 4.1.4. Model Maintenance Effort

Once the tool generates the size and complexity metrics values and computes the effort adjustment factor, the last step is to compute the maintenance effort using the equation defined in section 3.4 of this work. The tool computes maintenance effort by finding a product of size, complexity, and EAF values and displays the output. The computed maintenance effort for program one is presented in Figure 3.

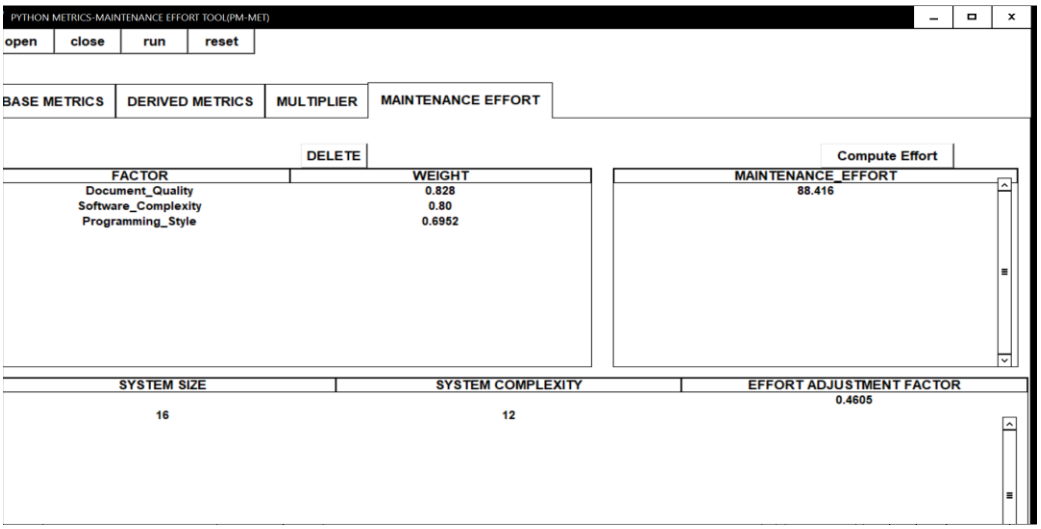


Figure 3. Software maintenance effort value for Python program one computed by the PMMET tool.

From the maintenance effort tab, the software maintainer can confirm the selected factors, the generated metric values of size and complexity, and EAF value before computing maintenance effort.

A total of twenty Python programs were considered in maintenance effort computation by the PMMET tool.

The following steps were followed towards computing software maintenance effort using PMMET tool.

Step1: Upload the Python file to the tool
Step2: click on run button
Step 3: The tool displays the base and derived metrics of uploaded Python file.
Step 4: The maintainer clicks on multiplier tab to select the factors affecting maintenance effort of the Python file.
Step 5: The tool computes an Effort Adjustment Factor (EAF)
Step 6: Maintainer clicks on the maintenance effort tab then the run button. The values for size, complexity, and EAF are presented to the maintainer for confirmation before computing effort.
Step 7: The maintainer clicks on compute effort button and views the computed maintenance effort of the Python file.

The maintenance effort computed for the twenty Python programs using the PMMET tool is presented in Table 2.

Table 2. Maintenance Effort for twenty Python programs computed by PMMET tool.

| **Program ID** | **Size** | **Complexity** | **Ranking of factors influencing effort** | **EAF** | **Model Effort** |
|---|---|---|---|---|---|
| program 1 | 16 | 12 | 3,7,15 | 0.4605 | 88.41 |
| program 2 | 13 | 10 | 1,3,5 | 0.575 | 74.76 |
| program 3 | 11 | 8 | 1,3,5,7,9,10 | 0.2874 | 25.29 |
| program 4 | 30 | 29 | 2,4,6,9 | 0.454 | 395.04 |
| program 5 | 18 | 18 | 1,2,3,4,5,12 | 0.3129 | 101.37 |

| | | | | | |
|---|---|---|---|---|---|
| program 6 | 20 | 8 | 2,3,5,8,12 | 0.3545 | 56.72 |
| program 7 | 5 | 4 | 3,5,7,9 | 0.426 | 8.52 |
| program 8 | 8 | 4 | 2,4,8,11,16 | 0.3039 | 9.72 |
| program 9 | 6 | 6 | 1,2,3,4,5,14 | 0.2993 | 10.79 |
| program 10 | 9 | 7 | 3,5,8,13 | 0.403 | 25.40 |
| program 11 | 6 | 17 | 1,2,4,7,9 | 0.385 | 39.27 |
| program 12 | 18 | 16 | 1,2,3,,5,7,8,,10 | 0.247 | 71.38 |
| program 13 | 23 | 15 | 1,2,3,6,8,21,22,23 | 0.082 | 28.33 |
| program 14 | 14 | 37 | 1,2,3,5,6,7,12,21 | 0.155 | 80.77 |
| program 15 | 30 | 29 | 1,2,3,4,5,12 | 0.312 | 272.24 |
| program 16 | 23 | 41 | 1,2,4,6 | 0.489 | 462.0 |
| program 17 | 7 | 9 | 2,3,4,7 | 0.467 | 29.43 |
| program 18 | 4 | 8 | 1,2,3,5,6,7,12,21 | 0.155 | 4.99 |
| program 19 | 7 | 7 | 1,2,5,10,14 | 0.345 | 16.90 |
| program 20 | 20 | 8 | 2,3,4,7 | 0.467 | 74.76 |

## 4.2. Context definition

Second, third, and fourth year students belonging to Python programmers club of Kirinyaga University School of Pure and Applied Sciences department of Computing Studies were presented with twenty Python files. The twenty python files can be accessed from https://github.com/huckbyte/python-tool/tree/main/source%20codes.

The subjects were asked about the features of Python programming language to assess their understanding of the language. All the forty participants had knowledge of classes, objects, methods, control structures, and Python indentation. 95% of the subjects had knowledge on inheritance and nesting. 80 % had knowledge on abstraction. From the responses received under subject assessment section, it was concluded that all the subjects were qualified to take part in the study. All the subjects were taken through a refresher course on Python object oriented programming intensively for two hours.

## 4.3. Experiment Strategy

A within- subject experiment design was adopted where every participant was assigned the same Python files for analysis. The subjects analyzed the Python files individually for two hours. The experiment objects consisted of twenty Python files which were already checked for syntax errors. Each participant was provided with twenty Python files containing different size and complexity metric values generated by the PMMET metric tool. The metrics tool was only used by the researcher to calculate metric values for the Python files and was not used by the participants.

## 4.4. Pilot study

This study was conducted using within- subjects experimental design in which all the subjects received all treatments. The study aimed to find out whether the metric values generated by the PMMET metric tool correlated with the subjects' rating of software maintainability of Python programs. The study also helped demystify whether the proposed metrics were contributors to increased levels of maintenance effort. A convenience random sample of ten subjects

participated in the pilot study. Results of the pilot study suggested that the questionnaire was suitable for use in the study.

## 4.5. Experimental Planning

Twenty Python files were shared with the students and the questionnaire. The instructions on how to carry out the exercise were explained for clarity purposes.

To establish whether there is any relationship between Python metrics and subject ratings on maintainability of Python programs was achieved by testing the following hypothesis.

i. Null Hypothesis (H0): There is no correlation between the Python size metric and the subject rating of maintainability of Python files.
ii. Alternative Hypothesis (H1): There is a correlation between the Python size metric and the subjects rating of maintainability on Python files.
iii. Null Hypothesis (H0): There is no correlation between the Python complexity metric and the subject rating of maintainability of Python files.
iv. Alternative Hypothesis (H1): There is a correlation between the Python complexity metric and the subjects rating of maintainability on Python files.
v. To establish whether there is any relationship between the proposed model's maintenance effort and subject ratings on maintainability of Python programs was achieved by testing the following hypothesis.
vi. Null Hypothesis (H0): The proposed model's maintenance effort has no effect on the subject ratings on maintainability.
vii. Alternative Hypothesis (H1): The proposed model's maintenance effort has a significant effect on the subject ratings on maintainability.

## 4.6. Experiment results

### 4.6.1. Python programs maintainability descriptive analysis

An experiment questionnaire was issued to forty students of Kirinyaga University, Kenya studying computer science related courses. The subjects were presented with twenty Python programs and were required to rate the extent to which the programs were maintainable. The SPSS statistical software was used to examine the acquired data. The descriptive statistics results of the twenty programs are presented in Table 3.

Table 3: Python programs maintainability descriptive analysis results

| Program no. | N | Minimum | Maximum | Mean | Std. Deviation |
|---|---|---|---|---|---|
| program1 | 37 | 2.00 | 4.00 | 2.6216 | .54525 |
| program2 | 37 | 2.00 | 4.00 | 3.2432 | .59654 |
| program3 | 37 | 2.00 | 4.00 | 3.4054 | .55073 |
| program4 | 37 | 3.00 | 5.00 | 3.9459 | .62120 |
| program5 | 37 | 2.00 | 5.00 | 2.9459 | .77981 |
| program6 | 37 | 3.00 | 4.00 | 3.4865 | .50671 |
| program7 | 37 | 1.00 | 5.00 | 2.9730 | 1.14228 |
| program8 | 37 | 1.00 | 3.00 | 1.6216 | .59401 |
| program9 | 37 | 1.00 | 5.00 | 2.4595 | 1.14491 |
| program10 | 37 | 1.00 | 5.00 | 2.4054 | .89627 |

| | | | | | |
|---|---|---|---|---|---|
| program11 | 37 | 1.00 | 5.00 | 3.1892 | 1.15079 |
| program12 | 37 | 2.00 | 4.00 | 2.5946 | .59905 |
| program13 | 37 | 2.00 | 5.00 | 3.0811 | .79507 |
| program14 | 37 | 2.00 | 5.00 | 4.0811 | .68225 |
| program15 | 37 | 2.00 | 5.00 | 3.7297 | .87078 |
| program16 | 37 | 3.00 | 5.00 | 4.3243 | .70923 |
| program17 | 37 | 1.00 | 4.00 | 1.6757 | .78365 |
| program18 | 37 | 1.00 | 3.00 | 1.6216 | .72078 |
| program19 | 37 | 1.00 | 4.00 | 2.2432 | .76031 |
| program20 | 37 | 1.00 | 5.00 | 3.0811 | .92431 |
| Valid N (listwise) | 37 | | | | |

### 4.6.2. Size, Complexity, Subjective data mean, and PMMET model effort values

Table three presents the values for size, complexity metrics, the mean value for subject ratings on maintainability (subjective data), and the proposed model's maintenance effort.

Table 4: Size, Complexity, Subjective data mean, and PMMET model effort values

| **Program no** | **Size** | **Complexity** | **Subjective data** | **Model Maintenance effort** |
|---|---|---|---|---|
| 1 | 16 | 12 | 2.62 | 88.41 |
| 2 | 13 | 10 | 3.24 | 74.76 |
| 3 | 11 | 8 | 3.41 | 25.29 |
| 4 | 30 | 29 | 3.95 | 395.04 |
| 5 | 18 | 18 | 2.95 | 101.37 |
| 6 | 20 | 8 | 3.49 | 56.72 |
| 7 | 5 | 4 | 2.97 | 8.52 |
| 8 | 8 | 4 | 1.62 | 9.72 |
| 9 | 6 | 6 | 2.46 | 10.79 |
| 10 | 9 | 7 | 2.41 | 25.40 |
| 11 | 6 | 17 | 3.19 | 39.27 |
| 12 | 18 | 16 | 2.59 | 71.38 |
| 13 | 23 | 15 | 3.08 | 28.33 |
| 14 | 14 | 37 | 4.08 | 80.77 |
| 15 | 30 | 29 | 3.73 | 272.24 |
| 16 | 23 | 41 | 4.32 | 462.0 |
| 17 | 7 | 9 | 1.68 | 29.43 |
| 18 | 4 | 8 | 1.62 | 4.99 |
| 19 | 7 | 7 | 2.24 | 16.90 |
| 20 | 20 | 8 | 3.08 | 74.76 |

### 4.6.3 Relationship between Python size metric and subject ratings on maintainability

A spearman's rank-order correlation was run to determine the relationship between Python metrics and subject ratings on maintainability. The results are presented in Table 4. Size metric is significantly correlated to the subjects rating of Python maintainability by a correlation coefficient of 0.646 and a p value of 0.002.

Table 5 Correlation results between Python size metric and subject ratings on maintainability

| Python metric | Correlation Coefficient | Sig.(2-tailed) |
|---|---|---|
| size | 0.646 | 0.002 |

**. Correlation is significant at the 0.01 level (2-tailed)

#### 4.6.4. Relationship between Python complexity metric and subject ratings on maintainability

A spearman's rank-order correlation was run to determine the relationship between Python complexity metrics and subject ratings on maintainability. The results are presented in Table 5. The complexity metric is significantly correlated to the subjects rating of Python maintainability by a correlation coefficient of 0.667 and a p value of 0.001.

Table 6 Correlation results between Python complexity metric and subject ratings on maintainability

| Python metric | Correlation Coefficient | Sig.(2-tailed) |
|---|---|---|
| complexity | 0.667 | 0.001 |

**. Correlation is significant at the 0.01 level (2-tailed)

#### 4.6.5 Relationship between the model's maintenance effort and subject ratings on maintainability

The association between model effort and subject assessments on maintainability was investigated using a spearman's rank-order correlation. The results are presented in Table 6. Maintenance effort is significantly correlated to the subject's ratings on maintainability by a correlation coefficient of 0.610 and a p value of 0.004

Table 7 Correlation results between model effort and subject ratings on maintainability

| | Correlation Coefficient | Sig.(2-tailed) |
|---|---|---|
| Model's effort | 0.610 | 0.004 |

**. Correlation is significant at the 0.01 level (2-tailed)

## 5. DISCUSSION

The proposed model for software maintenance effort estimation accepts three inputs namely; size, complexity, and an Effort Adjustment Factor value (EAF). Twenty Python projects were considered. Size and complexity metric values were computed using the PMMET tool. Effort multiplier values were determined through expert opinion and used in computing an Effort Adjustment Factor (EAF).

An analysis was performed to determine whether there is a correlation between the size metric and the ratings given by the subjects for maintainability. The results indicated a significant correlation between size metric and maintainability. As a result, the null hypothesis that there is no significant correlation between the size metric and subject ratings on maintainability was rejected and the alternative hypothesis was supported.

A second investigation was carried out to establish whether there is a relationship between Python complexity metrics and subject ratings on maintainability. The results indicated a significant correlation between complexity subject ratings on maintainability. As a result, the null hypothesis that there is no significant correlation between the complexity metric and subject ratings on maintainability was rejected and the alternative hypothesis was supported.

The correlation analysis results on python size and complexity metrics confirm that large size programs that have high levels of complexity will definitely require higher effort compared to small sized programs with low complexity.

The model's maintenance effort and the subjects' maintainability evaluations were the focus of a third experiment to determine whether there is a correlation.

The results indicated a correlation of 0.610 between the model's effort and subject ratings on maintainability. As a result, the null hypothesis that there is no significant correlation between the model effort and subject ratings on maintainability was rejected and the alternative hypothesis was supported. The experiment results confirms that the proposed model is valid.

## 6. CONCLUSION AND FUTURE WORK

This research aimed to develop a metrics based maintenance effort estimation model for Python programs. The model accepts three inputs namely; size, complexity, and an effort adjustment factor value (EAF). The EAF is obtained by calculating the product value of the factors influencing maintenance effort in a project or program. Results of the metrics tool, model's maintenance effort, and subject ratings were compared, and the findings indicated that metrics and subject ratings are strongly related and that the metrics are important factors in the maintainability of a Python program.

A limitation of the developed model was that the dataset used comprised of small sized python programs. Future tasks would be to implement the estimation model on large sized Python projects and employing a machine learning strategy then evaluating the outcomes. The researchers hope that this study will be of great benefit to Python developers and maintainers to aid in estimating the maintenance effort of small sized Python programs.

## ACKNOWLEDGEMENT

The Authors want to thank Professor Geoffrey Muchiri of Murang'a University of Technology, Murang'a Kenya. His comments and suggestions have contributed greatly to this work.

## REFERENCES

[1] Singh and Shivani, "EVALUATION OF AN ALGORITHM OF SOFTWARE DEFECTS OF UNDERSTANDABILITY USING A NEW METRICS OF SOFTWARE," *ESMSJ,* pp. 49 -53, 2021.

[2] Balagun, "Comparative Analysis of Complexity of C++ and Python Programming Languages," *Asian Journal of Social Science and Management Technology,* pp. 1-12, 2022.

[3] B. Boehm, "Software Engineering Economics," *IEEE,* 1983.

[4] Noor, Hayat, Hamid, Wakeel and Nasim, "Software metrics :investigating success factors , challenges , solutions and new research directions," *International journal of scientific and technology research,* pp. 38-44, 2020.

[5] Maheswaran and Aloysius, "Empirical Validation Of Object Oriented Cognitive Complexity Metrics

Using Maintenance Effort Prediction," *International Journal of Scientific Research in Computer Science Applications and Management Studie,* 2018.

[6] Misra and Akman, "Weighted class complexity: A measure of complexity for object oriented systems," *Journal of Information Science and Engineering,* 2008.

[7] Misra, Koyuncu, Crasso, Mateos and Zunino, "A suite of cognitive complexity metrics," in *International conference on computational science and its applications,* Berlin, 2012.

[8] Chidamber and Kemerer, "A metric suite for object oriented design," *IEEE,* pp. 476 -493, 1994.

[9] Chillar and Bhasin, "A new Weighted composite complexity measure for object oriented systems," *International Journal of Information and Communication Technology research,* pp. 101- 108, 2011.

[10] R. S. Chillar, Kajlaand and U. Chhilla, "Developing a nested class complexity metric for nested classes," in *6th international comference on compter and electrical engineering (ICCEE 2013)*, Paris, 2013.

[11] Hourani, Wasmi and Alrawashdeh, "A code complexity model of Object Oriented Programming (OOP)," in *Jordan international joint conference on Electrical Engineering and Information Technology (JEEIT),* Jordan, 2019.

[12] Ahn, Suh, Seungryeol kim and Hyunsoo KIm, "The software maintenance project effort estimation model based on function points," *Journal of software maintenance and evolution:Research and practice,* pp. 71-85, 2003.

[13] Choudhari and Suman, "Story Points Based Effort Estimation Model for Software Maintenance," *Elsevier,* pp. 761-765, 2012.

[14] Boehm, Abts, Brown and Chulani, Software cost estimation with COCOMO II, Prentice Hall, 2009.

[15] Srinath, "The Fastest Growing Programming Language," *International Research Journal of Engineering and Technology (IRJET),* pp. 354- 357, 2017.

[16] Obot, Udo and Obike, "Software Team Productivity Factor in Constructive Cost Model for Software Development Effort Estimation," *IJSE,* 2021.

[17] Islam and Katiyar, "Development of a software maintenance cost estimation model: 4th GL perspective," *International Journal of Technical Research and Applications,* vol. 2, no. 6, pp. 65-68, 2014.

[18] Hayes, Patel and Zhao, "A Metrics-Based Software Maintenance Effort Model,," *IEEE,* 2004.

[19] Buchmann,, Frischbier and Putz, "Towards an Estimation Model for Software Maintenance Costs," *IEEE,* pp. 313-316, 2011.

[20] Syavasya, "An Approach to Find Maintenance Costs Using Cost Drivers of Cocomo Intermediate Model," *International Journal Of Computational Engineering Research* , pp. 154-158, 2013.

[21] Mukunga, Ndia and Wambugu, "Factors Affecting Software Maintenance Cost of Python Programs," *International Journal of Software Engineering,* vol. 10, no. 2, pp. 22-36, 2022.

[22] Vu Nguyen, "Improved Size and Effort Estimation Models for Software Maintenance," *IEEE,* 2010.

[23] H. Sneed, Advances in Software Maintenance Management, IEEE Explore, 2003.

[24] Siddhi and Rajpoot, " Cost Estimation of Maintenance Phase for Component Based software," *Journal of Computer Engineering (IOSRJCE),* vol. 1, no. 3, pp. 1-8, 2012.

[25] Singh, Tiwari, Mishra and Misra, "Tuning of Cost Drivers by Significance Occurrences and Their Calibration with Novel Software Effort Estimation Method," *Advances in Software Engineering,* pp. 1-10, 2013.

[26] Munialo, Muketha and Omieno, "Size Metrics for service - oriented architecture," *International Journal of Software Engineering & Applications,* vol. 10, no. 2, pp. 67- 83, 2019.

[27] Kuan, "actors on software effort estimation," *International Journal of Software Engineering &Applications,* vol. 8, no. 1, pp. 23 - 32, 2017.

## AUTHORS

Short Biography

**Catherine Wambui Mukunga** is a Ph.D. student at Murang'a University of Technology, Kenya. She received her M.Sc. in Computer Science in 2014 from the University of Nairobi, Kenya and her BSc. in Information Technology in 2009 from Jomokenyatta University of Agriculture and Technology, Kenya. Her research activities are related to Software Engineering Metrics, Software Project Management and Machine Learning.

**Dr. John Gichuki Ndia** is a lecturer at Murang'a University of Technology, Kenya. He obtained his Bachelor of Information Technology from Busoga University, Uganda in 2009, his MSc. in Data Communication from KCA University, Kenya in 2013, and his PhD in Information Technology from Masinde Muliro University of Science and Technology, Kenya in 2020. His Research interests include Software Engineering, Computer Networks Security and AI Applications. He is a Professional Member of Institute of Electrical and Electronics Engineers (IEEE) and the International Association of Engineers (IAENG).

**Dr. Geoffrey Mariga Wambugu** is a senior lecturer at Murang' a University of Technology, Kenya. He obtained his BSc Degree in Mathematics and Computer Science from Jomo Kenyatta University of Agriculture and Technology in 2000, and his MSc Degree in Information Systems from the University of Nairobi in 2012. He holds a Doctor of Philosophy in Information Technology degree from Jomo Kenyatta University of Agriculture and Technology. His interests include Machine Learning and Text Analytics.