Constructive Cost Model II Metrics for Estimating Cost of Indigenous Software

Njoku A. Obilor¹, Agbakwuru A. Onyekachi² Amanze B. Chibuike³, Njoku O. Donatus⁴

¹,²,³ Department of Computer Science, Imo State University, Owerri-Imo State, Nigeria
⁴Department of Computer Science, Federal University of Technology, Owerri, Imo State-Nigeria

Received: 11 Jun 2021; Received in revised form: 02 Jul 2021; Accepted: 13 Jul 2021; Available online: 20 Jul 2021
©2021 The Author(s). Published by AI Publication. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

Keywords— COCOMO II, Cost-estimation, Indigenous software, Nigeria's Computing environment

Abstract—There is growing concern over the frequent cases of cost overruns, and underestimation in software cost, especially, indigenous software products. This has a lot to do with the choice of Cost-estimation tools, techniques and models deployed. Constructive Cost Model (COCOMO) II model has been adjudged as the most reliable and accurate. However, the existing cost drivers/variables of this model (COCOMO II) do not capture fully the uniqueness of Nigeria’s computing environment. This paper has highlighted the strengths and weaknesses of COCOMO II considering the hierarchy of COCOMO. A new algorithm is proposed to effectively enhance cost estimation effort of indigenous software in Nigeria.

I. INTRODUCTION

The Software development has become an essential question [1] because many projects are still not completed on schedule, with under or over estimation of efforts leading to their own particular problems [2]. Therefore, in order to manage budget and schedule of software projects [2], various software cost estimation models have been developed. Accurate software cost estimates are critical to both developers and customers [3]. They can be used for generating request for proposals, contract negotiations, scheduling, monitoring and control.

Cost estimation includes the process or methods that help us in predicting the actual and total cost that will be needed for our software and is considered as one of the complex and challenging activity for the software companies. Their goal is to develop software which is cheap and at the same time deliver good quality. Software cost estimation [4] is used basically by system analysts to get an approximation of the essential resources needed by a particular software project and their schedules. Important parameters in estimating cost are size, time, effort etc. Process of software estimation basically focuses on four steps.

A variety of cost estimation models was developed in the last two decades, including commercial and public models as well[5]. Constructive Cost Model (COCOMO) II is one of the most sophisticated estimation models that allow one to arrive at fairly accurate and reasonable estimates. Estimation helps in setting realistic targets for completing a project. This enables one to obtain a reasonable idea of the project cost. The value chain consists of the creators, distributors, resellers, and consumers.

Cost estimation is one of the most challenging tasks in Software Development. Many system projects have failed in the past due to an inaccurate estimate of the actual cost of delivery. This had happened because effective software estimation model had not been deployed by software organizations at the inception of software development. Underestimating the costs has resulted in management getting software with inadequate functionality, poor quality, under-staffing (resulting in staff burnout) and
failure to complete on time. This has also led to project abandonment. Overestimating a project can be just about as bad for the organization! This results in too many resources being committed to the project and delays the use of your resources on the next project or during contract bidding; result in not winning the contract, which can lead to loss of jobs. A solution to this malady is being sought by developing the COCOMO II cost estimation model to minimize this risk. Without reasonable accurate cost estimation capability, project managers cannot determine how much time and manpower cost the project should take and that means the software portion of the project is out of control from its beginning. With this development, system analysts cannot make realistic hardware-software trade-off analysis during the system design phase. Where the estimation is flawed, software project personnel cannot tell managers and customers whether their proposed budget and schedule are realistic.

There is a growing concern about how our indigenous software products are initiated and planned. For any new project, it is necessary to know how much it will cost to develop and how much development time is needed. These estimates are needed before development is ultimately initiated. In many cases, estimates are made using past experiences as the only guide. This should not be the case because projects differ in many respects, and hence past experiences alone are not enough. In order to achieve a reliable cost and schedule estimates, a number of options abound: delay estimation until late in the project; use decomposition techniques to generate project cost and schedules estimates; develop empirical models for estimation or acquire one or more automated tools. Unfortunately, the first option is not practical, even though attractive. The other options are used to establish the scope and cost estimates in advance. The cost estimate must and should be provided upfront. Amongst many cost-estimation tools, techniques and models, COCOMO II is the most reliable and accurate. This is because, COCOMO II mathematical equation is expandable and extendable to accommodate more variables (cost drivers), to suite unique and peculiar computing environments. Introducing and extending the COCOMO II model to reflect the country’s unique environment gives a better, reliable and accurate prediction of cost, effort, and duration required for the successful delivery of software projects on schedule.

**Hierarchy of Constructive Cost Model**

The Constructive Cost Model (COCOMO) is widely used algorithmic software cost model. It was proposed by Boehm [6]. It has following hierarchy-

a) Model 1 (Basic COCOMO Model): The basic COCOMO model computes software development effort and cost as a function of program size expressed in estimated lines of code (LOC) [7]. Being the first of the COCOMO set of models, the formula used by this model is:

\[ \text{Effort} = a^* (\text{KLOC})^b \]  

(1)

where, KLOC - denotes the code size , and a, b – constants such that value of these constants (Khaliefu et al., 2012) depends on the type of project, which is whether it’s organic, semi-detached or embedded.

b) Model 2 (Intermediate COCOMO Model): This takes the Basic COCOMO effort and schedule computation as its starting point. It then applies a series of multipliers to the Basic COCOMO figures. It takes into account factors such as required product reliability, database size, execution and storage constraint, personnel attributes and the use of Software tools. In this we obtain nominal effort estimation and the value of constants a, b differs from that of basic COCOMO. Formula used in this model is:

\[ \text{Effort} = a^* (\text{KLOC})^b * \text{EAF} \]  

(2)

Here the effort adjustment factor is represented by EAF.

c) Model 3 (Detailed COCOMO model): This model is slightly better than the Intermediate one. It has 17 cost drivers, instead of 15 which Intermediate Model has. This works on each sub-system separately and serves as a boon for large systems made up of non-homogenous subsystems.

Constructive Cost Models presumes the system and software requirements to be stable and predefined. But usually this situation is not always valid. This model provides some advantages but it also has some disadvantages. Advantages: Simple to estimate cost. Disadvantages: Because estimation in COCOMO Model is done at early stages of software development, many a times it may lead to estimation failures.

As a result of these problems the newest version of COCOMO which is COCOMO II was developed in 1990 and uses broader set of data. It uses source lines of code, function point and object points as inputs. It also includes some modifications to the effort multiplier cost drivers of previous COCOMO. The obtained output is in the form of size and effort estimates later developed into a project schedule. Advantages: COCOMO II proves to be an industry standard model, and has a clear and effective calibration process. Disadvantages: Calculation of duration for small projects is unreasonable.
II. METHODOLOGY AND SYSTEM ANALYSIS

Object-oriented analysis and design methodology (OOADM) which is adopted in this study is a set of standards for analysis and development of the COCOMO II software effort estimation. It uses a formal methodical approach to the analysis and design of information system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach one organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with. In other traditional analysis methodologies, the two aspects: processes and data are considered separately.

Sources of Data / Methods of Data Collection

In order to carry out a detailed analysis of the existing system, both primary and secondary data will be collected from different sources. Both secondary and primary data will be used to get facts on the subject where primary data will be collected from actual institutions and secondary data will be collected from literature review that include understanding and observing available COCOMO 11 software effort estimation. Secondary data will also be gathered from a number of sources in order to carry out an insightful investigation into the existing systems, its working procedures, and its mode of operation. Secondary data include: internet sources, journals, books, newspapers and COCOMO 81.

a) Data Collection Tools: Due to the sensitive nature of the study, the methods used for primary data collection were limited to the person(s) involved who were reluctant to have any written document from them, the result where the following methods:

b) Person/Telephone Interviews: This is done by interviewing software project key employees from their personal experience on areas on the COCOMO 11 software effort estimation that were prone to misuse by users or area already that had been misused by users. The key employees include cost drivers, constraints and software process.

c) Prototype System: This method proved to be very useful. Even though the software projects developers were reluctant to give information on the subject, when provided with a prototype system.

Analysis of the Existing System

According to the current comparison and based on the principals of the algorithmic and non-algorithmic methods. For using the non-algorithmic methods it is necessary to have the enough information about the previous projects of similar type, because these methods perform the estimation by analysis of the historical data. Also, non-algorithmic methods are easy to learn because all of them follow the human behavior. On the other hand, Algorithmic methods are based on mathematics and some experimental equations. They are usually hard to learn and they need to the much data about the current project state. But if enough data is reachable, these methods present the reliable results. In addition, algorithmic methods usually are complementary to each other, for example, COCOMO uses the SLOC and Function Point as two input metrics and generally if these two metrics are accurate, the COCOMO presents the accurate results too. Finally, for selecting the best method to estimate, looking at available information of the current project and the same previous projects data could be useful.

COCOMO II model: it is a collection of three variants, Application composition model, early design model, and Post architecture model. This is an extension of intermediate COCOMO model and defined as:

$$\text{Effort} = 2.9 \times (\text{KLOC})^{1.10}$$  \hspace{1cm} (3)

Table 1 shows the advantages and disadvantages of existing method.

| Method       | Type       | Advantages                                    | Disadvantages                                      |
|--------------|------------|-----------------------------------------------|----------------------------------------------------|
| COCOMO       | Algorithmic| Clear results, very common                     | Not much data is required; it is not suitable for any project |
| Expert       | Non-A      | Fast prediction, Adapt to special projects    | Its success depends on expert. Usually is done incomplete |
| Function     | Algorithmic| Language free, its results are better than SLOC| Mechanization is hard to do, quality of output is not guaranteed |
| Model        |            |                                              |                                                    |
| Cost Drivers |            |                                               |                                                    |
| Constraints  |            |                                               |                                                    |
| Software     |            |                                               |                                                    |
| Process      |            |                                               |                                                    |

Table 1: Advantages and Disadvantages of the Existing Methods
III. ANALYSIS OF THE PROPOSED SYSTEM

This research has generated algorithmic effort estimation for COCOMO II measurement. The proposed system is built to help all the practitioners measure the size of computerized business information system. Such sizes are needed as a component of measurement of productivity in system development and maintenance activities and as a component of estimating the effort needed for such activities. Nowadays, software developers recognize the importance of the realistic estimates of effort to success management of software projects and having realistic estimates at an early stage of project life cycle which allow project manager and development organizations to manage resource effectively. The process starts with the planning phase activities and refined throughout the development.

The proposed system is designed to establish better and more realistic estimations for software projects. The system is designed and built with an infusion of some dummy variables and also features a user friendly graphic user interface (GUI).

The study introduces certain cost drivers that are peculiar to Nigeria’s computing environment and indeed the third world countries. These are issues that relate to our computing Environmental. They are Indigenous Environmental Cost Factors.

The following are the new values added in the proposed system and are summarized in Table 2:

| Cost Drivers                  | Rating  | Values |
|-------------------------------|---------|--------|
| Power Supply (PS)             |         |        |
| PS                            | Very Poor | 1.75   |
|                                | Poor     | 1.5    |
|                                | Good     | 1.1    |
|                                | Excellent| 1.0    |
| Public Relations Need/Goodwill (PRN) | Normal | 1.0    |
| Government Policies (GTP)     | Consistent| 1.0    |
|                               | Inconsistent| 1.5   |
| Corporate/Social Responsibilities (COSR) | Rural | 1.75   |
|                                | Semi-Rural| 1.5    |
|                                | Urban    | 1.2    |

Table 2: Indigenous Environment (New) Cost Factor

COCOMO II Model Structure And Its Variables

Upon data collection, the following variables were proposed. Definitions of the variables are explained below. Effort is a dependent variable referring to total man-hour effort required to build a software project. Independent variables include Development kit (Dev-kit), Designer-experience (Design-exp), No-of-programmers (No-prog), Complexity (Comp) and Education-level (Edu-level).

a) Effort: This variable emphasizes the effort (man-hour) spent by project developers to design application software. Effort is measured either in man-hour or man-month depending on the size of software projects. In the study, one considers man-hour is because the software projects are small to medium. Some software projects didn’t last several months. For those software projects studied, only the time spent in analyzing and designing by project designers is counted. While the time spent to discuss with clients and end users are excluded. The measurement used to count the effort is the total number of man-hours for single software project. The software company has a very good practice to record detailed information, such as time spent for each project, the number of project designers assigned to a project and the development tool used, of each developed software project. Therefore, the data collection process was easy and straight forward.

b) Dev-kit: This variable is to measure the complexity of system development kit used by project designers. Usually, the complexity of a development kit correlate to the time required to develop software projects, as a good development kit can make programmers more productive during system development. When a suitable development kit is used, it can support the construction process by automating tasks executed at every stage of system development life cycle. It facilitates interaction among project designers by diagramming a dynamic, iterative process, rather than one in which changes are cumbersome. It is also a useful tool to enable project designers to clarify end user’s requirements at the very early stage of system development life cycle. CASE tool is the common development kit used to support development process in many companies. This factor is measured with a five-point Likert-like scale ranging from (1) very low productivity to (5) very high productivity.
c) Designer-exp: This variable is to measure the actual working experience of project designers designing application software in computer industry. The actual experience of project designers in developing software projects and the experience in a specific kind of programming language are key determinants. By common sense, an experienced project designer can reduce the number of errors to program codes if he has good mastering of that type of programming language and has a number of years in developing software projects. This leads to a minimum time in developing and maintaining programs in the future. Thus, the more the number of years of service that a designer serves in the industry, the higher the level of working experience the designer has gained. We take the average of years of experience among the team members if there is more than one participates in a project.

d) No-prog: This variable is to count the number of project designers working collaboratively as a team. In order to make sure a late project which can be completed on time, there are project designers who often add extra programmers. Sometimes, this arrangement may not work well, especially when there is lack of proper communication among project designers and no training offered before the development. This could definitely slow down the development process and lead to many problems. However, the situation may not happen in our study, because the software projects developed by a team of project designers are small to medium in term of LOC. A project designer is relatively easy to make an accurate estimate before a software project starts. Therefore, there are no additional members who are invited to a late project. For this variable, according to the detailed information of the developed projects, one is in an easy position to collect the number of project developers responsible for each project being developed.

e) Comp: This variable refers to the degree of program complexity designed. A thorough understanding of the software development process improves the relationship between program complexity and maintenance effort. That is, high complexity of software projects increases the difficulty of project designers to quickly and accurately understand the programs before they are developed or repaired. The higher the level of complexity of a program is, the greater the effort required by project designer. Especially, when a program has highly interactive modules to communicate not only within it, but also with modules from other programs. This will increase the time required by project designers in designing the software projects. In the study, this variable is to measure and examine system specifications and design specifications prepared by the company during analysis and design phases. Due to the characteristics of collected software projects, they all are business oriented programs. The determination process for program complexity is under the control of project designers. For this variable, the data is collected using a five-point Liker-like scale ranging from (1) very low complexity to (5) very high complexity.

f) Edu-level: This variable is to measure the level of education that a project designer has acquired in related field. Many companies prefer to recruit programmers who are equipped not only with extensive working experience in industry but also those who have well training with at least a bachelor degree or higher in related field. Project designers with higher level of education usually can solve programming problems more easily than those who don’t. To measure the factor, we use a five-point Liker-like scale ranged from (1) very low level of education to (5) very high level of education.

A linear regression model is hypothesized following discussion of the variables and it is shown in the following equation:

\[
Effort = \alpha + \beta_1Dev_kit + \beta_2Designer_exp + \beta_3No_pro + \beta_4Comp + \beta_5Edu_level
\]

(4)

where: \(\alpha, \beta_1, \beta_2, \beta_3, \beta_4 \) and \(\beta_5\) are constants; \(Dev_kit\) - Software Development tools/kits; \(Designer_exp\) – Experience of the Software Designer; \(No_pro\) – number of programmers; \(Comp\) – Complexity of Software \(Edu_level\) – highest level of education

The full COCOMO II model includes three stages:

Stage I: Supports estimation of prototyping or applications composition efforts.

Stage 2: Supports estimation in the Early Design stages of a project, when less is known about the project’s cost drivers.
Stage 3: Supports estimation in the Post-Architecture stage of a project.

This version of COCOMO II implements stage 2 formulas to estimate the effort, schedule, and cost required to develop a software product. It also provides the breakdown of effort and schedule into software life-cycle phases and activities from both the Waterfall model and the M base Model. The M base model is fully described in Software Cost Estimation with COCOMO II. The stages of the model are shown in Figure 1.

![COCOMO Model Stages Diagram](image)

**COCOMO II Model Effort Estimation Equations**

a) **Effort Estimation:** Obtaining the values of \( A \), \( B \), \( EM_i \), and \( SF_j \) in COCOMO II is managed by calibrating the parameters and effort for the 161 projects in the model database. The main formula is below (Eq. 5) and acquires size of the software development as input, combined with predefined constant \( A \), an exponent \( E \) inclosing five scale factors, and 17 so called effort multipliers.

The predefined constant estimates productivity in PM/KSLOC for the case where a project’s economies and diseconomies of scale are in balance. Productivity alters as the exponent changes for the reason that of non-linear effects on size. The constant is originally set when COCOMO II is calibrated to the project database which reflects a global productivity average.

The application size exponent is aggregated of five scale factors (SF) that describe relative economies or diseconomies of scale that are encountered for software projects of dissimilar magnitude. A project exhibits economies of scale if the exponent is less than one i.e. effort is non-linearly reduced. Economies and diseconomies of scale are in balance should the exponent hold a value of one. A project exhibits diseconomies of scale if the exponent is more than one i.e. effort is non-linearly increased in Eq. (7).

Boehm, [9] selected the scale factors in a foundation on the underlying principle that they have a significant exponential effect on effort or productivity disparity. As seen from the below formula, the five scale factors are summed up and utilized to establish a figure for the scale exponent.

b) **Cost Drivers:** Cost drivers are characteristics of software development that influence effort in carrying out a certain project. Unlike the scale factors, cost drivers are selected based on the rationale that they have a linear effect on effort. There are 17 effort multipliers (EM) that are utilized in the COCOMO II model to emulate the development effort. What will be exposed in the subsequent chapter is that every multiplicative cost driver is assigned the same rating level with the distinction being the combination of assigned weights. Annotated by [9] is the possibility to assign transitional rating levels and weights for the effort multipliers. They are furthermore leveled to establish a mean value that supplantly reflects upon a more reasonable figure. Even though the model specifies a finite number of cost drivers, COCOMO II allows the user to define its own set of effort multipliers to better correspond to prevailing circumstances in any given development. Cost drivers are rated and founded on a sturdy rationale that they autonomously give details on a considerable source of effort and/or productivity discrepancy. Nominal levels do not impact effort whilst a value beneath/over one decreases/increases it.

\[
PM = \frac{1}{17} \prod_{i=1}^{17} \left( EM_i \right) \cdot A \cdot \left( 1 + \frac{PEM}{100} \right) \cdot \text{Size} \left( 0.9 + 0.01 \frac{5}{j=1} \sum SF_j \right)
\]

where:

\[
\text{Size} = \frac{KN\text{SLOC}}{100} + \frac{(AA + S + 0.4 \text{DM} + 0.3 \text{CM} + 0.3 \text{IN})}{100}
\]

\[
B = 0.91 + 0.01 \sum_{j=1}^{5} SF_j
\]

| Symbol | Description |
|--------|-------------|
| A      | Constant, currently calibrated as 2.45 |
| AA     | Assessment and assimilation |
| ADAPT  | Percentage of components adapted (represents the effort required in understanding software) |
AT | Percentage of components that are automatically translated
ATPROD | Automatic translation productivity
REVL | Breakage: Percentage of code thrown away due to requirements volatility
CM | Percentage of code modified
DM | Percentage of design modified
EM | Effort Multipliers: RELY, DATA, CPLX, RUSE, DOCU, TIME, STOR, PVOL, ACAP, PCAP, PCON, APEX, PLEX, LTEX, TOOL, SITE
IM | Percentage of integration and test modified
KASLOC | Size of the adapted component expressed in thousands of adapted source lines of code
KNSLOC | Size of component expressed in thousands of new source lines of adapted source lines of code
PM | Person months of estimated effort
SF | Scale Factors: PREC, FLEX, RESL, TEAM, PMAT
SU | Software understanding (zero if DM = 0 and CM = 0)

Schedule Estimation Equation

Determine time to develop (TDEV) with an estimated effort, PM, that excludes the effect of the SCED effort multiplier.

\[
TDEV = \left[3.67 \times (PM)^{0.18 + 0.1(5-1.01)}\right] \times \frac{SCED\%}{100} \tag{7}
\]

where:

\[
B = 0.91 + 0.01 \sum_{j=1}^{5} SF_j \tag{8}
\]

Scale Factors: Equation (8) defines the exponent, B, used in Eq. (7). Table 4 provides the rating levels for the COCOMO II scale drivers. The selection of scale drivers is based on the rationale that they are a significant source of exponential variation on a project’s effort or productivity variation. Each scale driver has a range of rating levels, from Very Low to Extra High. Each rating level has a weight, W, and the specific value of the weight is called a scale factor. A project’s scale factors, W, are summed across all of the factors, and used to determine a scale exponent, B.

| Symbol | Description |
|--------|-------------|
| PM | Person months of estimated effort from Early Design or Post-Architecture models (excluding the effect of the SCED effort multiplier) |
| SF | Scale Factors: PREC, FLEX, RESL, TEAM, PMAT |
| TDEV | Time to develop |
| SCED | Schedule |
| SCED% | The compression / expansion percentage in the SCED effort multiplier |

Table 4: COCOMO Scale Drivers

In COCOMO II, the logical source statement has been chosen as the standard line of code. Defining a line of code is difficult due to conceptual differences involved in accounting for executable statements and data declarations in different languages. The goal is to measure the amount of intellectual work put into program development, but difficulties arise when trying to define consistent measures across different languages. Breakage due to change of requirements also complicates sizing. To minimize these problems, the Software Engineering Institute (SEI) definition checklist for a logical source statement is used in defining the line of code measure. The Software Engineering Institute (SEI) has developed this checklist as part of a system of definition checklist, report forms and supplemental forms to support measurement definitions.
Post-architecture model

COCOMO II helps in the reasoning about cost implications of software decisions that need to be made, and for effort estimates when planning a new software development activity. The model uses historical projects as data points by adding them to a calibration database which is then calibrated by applying statistical techniques. The post-architecture model is utilized once the project is ready to be developed and sustain a fielded system meaning that the project should have a life-cycle architecture package which provides comprehensive information on cost driver inputs and enables more accurate cost estimates. All further references to COCOMO II can be assumed to be in regard to the post-architecture model.

For the Rational Unified Process (RUP) model, all software development activities such as documentation, planning and control, and configuration management (CM) are included, while database administration is not. For all models, the software portions of a hardware-software project are included (e.g., software CM, software project management) but general CM and management are not [9]. COCOMO II estimates definitions of labor categories, thus they include project managers and program librarians, but exclude computer center operators, personnel-department personnel, secretaries, higher management, janitors, etc. A person-month (PM) consists of 152 working hours and has by [9] been found consistent with practical experience with the average monthly time off (excluding holidays, vacation, and sick leave).

It is of outmost importance for good model estimations to have a sufficient size estimate.[9] elucidates that determining size can be challenging and COCOMO II only utilizes size data that influences effort thus, new code and modified implementations is included in this size baseline category. Normal application development is typically composed of new code; code reused from other sources –with or without modifications – and automatically translated code. Adjustment factors capture the quantity of design, code and testing that was altered. It also considers the understandability of the code and the programmer familiarity with the code.

COCOMO II expresses size in thousands of SLOC (KSLOC) and excludes non-delivered support software such as test drivers. They are included should such they be implemented in the same fashion as distributed code. Determinants are the degree of incorporated reviews, test plans, and documentation. [9] conveys that “the goal is to measure the amount of intellectual work put into program development”. The definition of a SLOC can be quite different in nature because of conceptual dissimilarities in different languages. As a consequence, backfiring tables are often introduced to counterbalance such circumstances. This is fairly reoccurring when accounting size in diverse generation languages. However, an organization that specializes in one programming language is not exposed to such conditions. A SLOC definition checklist is made available in the Appendix and somewhat departs from the Software Engineering Institute (SEI) definition to fit the COCOMO II models definitions and assumptions. Moreover, the sidebar demonstrates some local deviations that were interpreted from the – to some extent – general guidelines. Code produced with source code generators is managed by counting separate operator directives as SLOC. Concurring with [9], it is divulged to be highly complex to count directives in an exceedingly visual programming system. A subsequent section will unearth the settlement of this troublesome predicament.

IV. HIGH LEVEL MODEL OF THE PROPOSED SYSTEM

This section presents the model of the proposed system.

![Block diagram of proposed high level model](https://www.iijaers.com)

Application Composition Model

In the beginning of a project when the developer does not have any detailed design and maybe not even formulated the requirements, this model should be used. It is based on object points as an estimation of the software’s size. To calculate object points is way to estimate the size of software, early in development process. The very first thing to do when an object point analyses should be made, is to identify screens, reports and 3GL components. After that the objects should be classified in the difficulty levels simple, medium and difficult. In the same way as with function points every class and difficulty level is assigned a number which functions as weight.
Advantages of the new system:

1. It is an open cost model, in which all details are published. Very profound information is easily available.
2. COCOMO II adjusts for software reuse and reengineering where automated tools are used for translation of existing software.
3. It can be used to determine the actual size of the project by algorithmic methods as well as historical data or expert opinions.
4. The COCOMO II software cost estimation model provides a tailor able cost estimation capability well matched to the major current and likely future software process trends.
5. It offers clear and effective calibration process.
6. COCOMO II has effective tool support (also for the various extensions).
7. Well-documented, ‘independent’ model which is not tied to a specific software vendor
8. Algorithmic cost models like COCOMO II support quantitative option analysis as they allow the costs of different options to be compared.

V. CONCLUSION

An Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control is not possible without a sound and reliable estimate. As a whole, the software industry does not estimate projects well and doesn’t use estimates appropriately. We suffer far more than one should as a result and we need to focus some effort on improving the situation. Thus, the software engineering community has put tremendous effort to develop models that can help estimators to generate the accurate cost estimate of a software project. In the last three decades, many software estimation models and methods have been proposed, evaluated, and used.

There are many software cost estimation methods available including algorithmic methods, estimating by analogy, expert judgment method, top-down method, and bottom-up method. No one method is necessarily better or worse than the other but COCOMO II is preferred over other methods because it is the most suitable for large and lesser known projects. COCOMO II has capabilities to deal with the current software process and is served as a framework for an extensive current data collection and analysis effort to further refine and calibrate the model's estimation capabilities. The COCOMO models provide clear and consistent definitions of processes, inputs, outputs, and assumptions, thus help estimators reason their estimates and generate more accurate estimates than using their intuition. It is obvious that the proposed system has both advantages and disadvantages. But the advantages far outweigh the disadvantages thereby justifying the proposed system.

REFERENCES

[1] Albrecht, A. J. and Gaffiney, J. E. (2010). Software function, source lines of code, and development effort prediction: a software science validation. IEEE Transaction on Software Engineering, 639-647.
[2] Putnam, L. H. (2008). A General Empirical Solution to the macro software sizing and estimation problem. IEEE Transactions on Software Engineering, 345-361.
[3] Caper, J. (2007). Estimating Software Cost. Tata: Mc-Graw Hill.
[4] Pressman, R. S. (2005). Software Engineering: A Practitioner’s Approach. (6th ed.). McGraw-Hill, New York, USA.
[5] Osuagwu, O. E. (2008). Software Engineering: A Pragmatic and Technical Perspective. Owerri: Oliverson Industrial Publishing House.
[6] Black, R. K. et al (2014). BCS Software Production Data, Final Technical Report, RADC-TR-77-116. Boeing Computer Services, Inc.
[7] Chris, F. K. (2010). An Empirical Validation of Software Cost Estimation Models. Management of Computing Communications of ACM, 30(5), 416-429.
[8] Khalifelu, Z. A., and Farhad, S. G. (2012). Comparison and Evaluation of data mining techniques with algorithmic models in software cost estimation. Procedia Technology.
[9] Boehm, B. W. (2010). Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. Analis of Software Engineering Special Volume on Software Process and Product Measurement, Science Publishers, Amsterdam, Netherlands, 1(3), 45-60.