Software Models, Extensions and Independent Models in Cocomo Suite: A Review

1 Syed Ali Abbas and Saleem Ullah Lar, 2 Xiaofeng Liao, 3 Raja Aftab Naseem

1 PhD Scholars, College of Computer Science, Chongqing University, PR China
2 Professor, College of Computer Science, Chongqing University, PR China
3 Lecturer Computer Science, University of Azad Jammu & Kashmir

ABSTRACT

Different models over the years have been proposed to support the software cost estimation processes. Cocomo (Constructive Cost Model) is one of the successful models which enabled software engineers to reliably reason about the effort and schedule estimates. Since past few years different derivative models and extensions of Cocomo have been introduced to meet the emergent needs of different aspects of system engineering and software engineering. In this paper we have presented a descriptive overview of Cocomo suite models and extensions. We have inclusively included the models in the Cocomo suite, discussed the methodology they adopt, relevant environment for their use, concept behind their development and the procedures by which they are developed. This paper provides a baseline to researchers, software organizations and practitioners to comprehend the enhancements in Cocomo suite models and differences among these derivatives.

Keywords: Software Estimation, Cocomo Suite, CoQualMo, Defect Introduction, CoCOTs, Software Effort.

1. INTRODUCTION

Since long the matter of interest for software engineers in the field of software cost and effort estimation are the questions: What modeling techniques can produce accurate or better results, How much time and effort is required to develop the software, and what would be the influencing cost and risk factors? The answers to these questions are enclosed in the attempts of researchers resulting into number of classical estimation models for example SDC [1], Wolverton [2], Putnam [3], Meta Model [5], and Cocomo [6] and till present where the software estimation is now coupled with the concrete concepts taken from neural networks [55][56] and fuzzy logic [58][57] to understand and monitor the dynamic process of software development, with the common goal of estimating software effort [7].

The classical estimation modeling techniques were typically developed on the basis of limited and specific data available to solve a specific problem while staying in the same development boundaries [5], hence, prohibiting the practice of such limited techniques in broader scenarios. The growing needs have led the emergence of many rapid-development processes, reuse-driven approaches, reengineering, object-orientations and many other process maturity initiatives [15], which have significantly affected the precision and reliability of classical estimation techniques. Authors in [13] discussed that “….the growing need for the models to estimate different aspects of software development served as a catalyst for the creation of derivative models and extensions that could better address commercial off-the-shelf software integration, system engineering, and system-of-systems architecting and engineering”. To deal with these changing trends based on different aspects of software development many calibrations have been made in few estimation techniques which have given birth many offspring techniques inheriting the best features of their predecessor, hence making them doubtlessly more reliable.

Two techniques namely Jensen model [57] and Cocomo [6] are constantly improving and recalibrating by researchers. On one hand Jensen model has provided a base to develop many commercial tools for software products [56]. On the other hand number of software models, software extensions and independent models based on Cocomo are developed and emerged in Cocomo suite [12]. We are not aiming to defend or prove the superiority of any technique over another. We intend to present an overview of most significant features of the models in the Cocomo suite that includes software models, different extensions and independent models.

Although it is not feasible to include whole literature related to cocomo suite, however we intend to comprehend the fundamentals of cocomo suite models from available literature. Cocomo suite models presented at one place will give our readers an opportunity to identify core features, understand the similarities and the differences and recognize pros and cons among them.

The rest of paper is distributed as follows: Section 2 provides the overview of Software models and their extensions respectively. In section 3 we have provided the descriptive exploration of independent models appearing in Cocomo suite. In section 4 we presented other independent
models in Cocomo suite and paper concludes with summary of whole discussion.

2. SOFTWARE COST MODELS IN COCOMO SUITE

Four models are incorporated as software cost models in Cocomo suite [12] namely (i) Cocomo 81, (ii) Cocomo II, (iii) DBA Cocomo, and (iv) CoInCoMo respectively. We are focusing our attention exclusively on Cocomo II and onwards developments while ignoring the Cocomo 81 because most of the models in the Cocomo suite are extended from Cocomo II. Other three models (Cocomo II, CoInCoMO and DBA CoCoM0) are basically the same model but adapted for different development situations [12]. DBA Cocomo serves as a database version of the existing CocomoII that is intended to provide storage for existing project data and additional functionalities to those data [14]. CoInCoMo is also based on Cocomo II with an added superstructure to accommodate multiple packages and modules. Effort can be calculated for each build or package through Cocomo II and summing it for achieving overall estimate [14]. Both these copy write models are still under progress at University of Southern California. Most of the models in Cocomo suite are developed in 7 step fashion [13]:

(i) After the deep analysis of existing literature
(ii) Analysis of behavior
(iii) Identification of relevant significance of parameters
(iv) Performing expert judgment/Delphi assessment
(v) Gather project data to validate the cost-estimating relationships in the model
(vi) Use Bayesian statistical techniques that provide the ability to balance expert data and historical data
(vii) Gather more data to refine model.

In forthcoming sections we have discussed the cocomo suite models. As it is mentioned before that few models in cocomo suite are still in the phase of calibrations, hence readers are advised to frequently observe for latest updates.

2.1 Cocomo II

The objectives of Cocomo II [13] effort are to develop software cost database and tool support capabilities for continuous model improvement, to provide a quantitative analytic framework, set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules [15]. Cocomo II model includes confirmed features of COCOMO 81 and Ada COCOMO models. It is designed for the modern types of software development such as business software, object-oriented software, and the softwares based on Spiral or Evolutionary models [16][27].

Contrary to Cocomo 81 where the input was taken as KDSI and point estimates of effort and schedule were provided [30], Cocomo II uses KSLOC as input and provides a detailed nonlinear approach of software reuse effects; and also presents a family of 3 sub models [17] [26] adjusted to the information available at different stages of the development process [27]. These models are briefly introduced in following sections:

2.1.1 Application Composition Model

Application Composition model is used in the early stages of development, when the focus is set on user interface prototyping, software and system interaction deliberation, performance assessment, and technology maturity [23]. It helps to compute effort and schedule of projects that use Integrated Computer Aided Software Engineering tools for rapid application development [17]. The total size is estimated in the form of application points and a nominal productivity is measured by two factors i.e developer experience and capability, and integrated computer software engineering tool maturity and capability [55]. Estimated effort is obtained by dividing the size by productivity. The initial size measure is determined by counting the number of screens, reports and the third generation components that will be used in application [19] and then each object point is classified into a complexity weight which is either simple, medium, or difficult. The original number of object instances is multiplied by the weighting factor and then summed to get the total number of object points.

In composition model the new object points are computed by following expression [20].

\[ \text{New Object Points (NOP)} = \left( \frac{\text{Object Points}}{100-\%\text{Reuse}} \right) / 100 \]

Productivity rate, PROD, is estimated from subjective average of developer’s experience and ICASE maturity/Capability [20] and is shown in table 1.
Table 1: Productivity measures for Application Composition Model

<table>
<thead>
<tr>
<th>Developer’s Experience/ Capability</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Maturity/ Capability</td>
<td>Very Low</td>
<td>Low</td>
<td>Nominal</td>
<td>High</td>
<td>Very Low</td>
</tr>
<tr>
<td>Prod</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

The effort in Person Month PM is estimated as [20].

\[ PM = \frac{(NOP)}{PROD} \]

2.1.2 Early Design Model

Early Design model is used when a rough estimate is needed based on incomplete project and product analysis [17]. It uses to evaluate alternative software system architectures where unadjusted function point is used for sizing. The corresponding COCOMO II capability involves the use of function points and a course-grained set of 17 cost drivers [17]. The expression used for Effort estimation is given by:

\[ PM = A \cdot \left[ \text{Size}' \right]^B \cdot \prod_{i=1}^{17} EM_i \]

Where A is the constant that is set to 2.5 and is used to capture the multiplicative effects on effort with projects of increasing size. B accounts for the relative economies or diseconomies of scale encountered for software projects of different sizes [24] and obtained as:

\[ B = 1.01 + 0.01 \cdot \sum_{j=1}^{5} SF_j \]

Where SF is sum of 5 scaling factors and that are Precededness, Development Flexibility, Architecture / Risk Resolution, Team Cohesion and Process maturity. 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 [20]. EM is effort multiplier that is calculated using seven cost drivers (Product Reliability and Complexity, Developed for Reusability, Platform Difficulty, Personnel Capability and Mapping Example, Personnel Experience, Facilities, Schedule) [20].

The size is estimated with following expression.

\[ \text{Size}' = \text{Size} \cdot \{1+ (\text{BRAK}/100)\} \]

Where BRAK is percentage of code thrown away due to requirements volatility [20].

The initial version of COCOMO II provides a simple schedule estimation capability similar to those in COCOMO and Ada COCOMO. The initial baseline schedule equation for all sub models of COCOMO II is:

\[ TDEV = [A \cdot PM^{(0.33 + 0.2 \cdot (B – 1.01))} \cdot (\text{SCED}%)]/100 \]

Where TDEV is the calendar time in months. A is a constant value that is set to 3.0, PM is the estimated person-months excluding the Required Development Schedule (SCED) effort multiplier, B is the sum of project scale factors and SCED% is constraint imposed on the project team developing the software which refers to the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. SCED values for stretched out or accelerations given as.

Table 2: Required Development Schedule (SCED)

<table>
<thead>
<tr>
<th>SCED</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>75% of nominal</td>
<td>85%</td>
<td>100%</td>
<td>130%</td>
<td>160%</td>
<td></td>
</tr>
</tbody>
</table>

2.1.3 Post-Architecture Model

Post Architecture model is used when top level design is complete and detailed information about the project is known [17]. The post architecture model includes a set of 17 cost drivers [20] and a set of 5 factors determining the projects scaling component [19]. The equations for effort estimation, size and development time are the same as in Early design modeling differing only in the case of cost drivers and it can also be used to estimate the Maintenance effort [20].

\[ PM = A \cdot \left[ \text{Size}' \right]^B \cdot \prod_{i=1}^{17} EM_i \]

3. SOFTWARE EXTENSIONS IN COCOMO SUITE

This section includes the discussion on the models that are software extensions in Cocomo Suite [12]. Security Extension 2004 is not included in our
discussion due to its limited availability. Rest of models is presented below.

3.1 CoQualMo

COnstructive QUALity MOdel (CoQualMo) is proposed by Sunita Chulani and Barry Boehm [31] that correlates cost, schedule and quality factors in software development process [31]. CoQualMo specifically focuses on quality cost intended to estimate cost earlier in the development period and it uses coarse-grained categories of defect-detection techniques [32] hence allowing determining an estimate of the number of faults contained in the software [33]. Coqualmo helps to predict potential defect density in the system by adding defect introduction and defect removal parameters in its predecessor Cocomo [34].

CoQualMo works on the basic concept derived from Boehm’s defect introduction and removal model [6] and Capers Jones’ tank and pipe model [35] where defects abstractly flow into a tank, just like water tank system, through various defect source pipes. Defects are introduced and removed by additional pipes representing processes that may introduce or remove defects from the system [36]. CoQualMo represents these defects introduction and removal into two sub-models, namely Defect Introduction Sub Model and Defect Removal Sub Model [31].

3.1.1 Defect Introduction (DI) Sub Model

The input to Defect Introduction model includes Source Lines of Code and/or Function Points as the sizing parameter, and set of 21 multiplicative DI-drivers [table I, 31] derived from a subset of the cost drivers from COCOMO II [13]. These drivers are divided into four categories namely platform, product, personnel and project. The output to this Defect Introduction DI model is the predicted number of defects detected during software process [37] with Critical, high or medium impact [31].

The expression used by authors in [31] for defect introduction is given as:

\[ \sum_{j=1}^{3} A_j \cdot (\text{Size})^B_j \cdot \text{QAF}_j \]

Where \( j \) is the three phases of life cycle, i-e requirements, design and code, \( A \) is the multiplicative calibration constant, \( \text{Size} \) of the software project measured in terms of thousands of Source Lines of Code, \( B \) refers to economies / diseconomies of scale initially set to 1, and QAF (Quality Adjustment Factor) is the Defect Introduction driver for the \( j \)th artifact and the \( i_{th} \) factor [31]. For details see [31] [38].

Authors in [31] assigned numerical values with expert judgment to each of the ratings of the DI drivers to provide Defect Introduction model a pragmatic base. The set of values proposed by authors [table II, 31] demonstrates that if defect introduction driver’s value is greater than 1 then it has a damaging effect on the number of defects introduced otherwise less number of defects are introduced [31]. Some initial data analysis yielded 9 requirement defects/KSLOC, 19 design defects/KSLOC, and 33 code defects/KSLOC nominal Defect Introduction rates for COQUALMO[39].

3.1.2 Defect Removal (DR) Sub Model

DR model covers the identification and elimination of defects in later phases of the software project [36] and estimates the number of defects removed by several defect removal activities namely Automated Analysis (includes syntax, code etc analyzers), People Reviews (peer group related activities) and Execution Testing and Tools (tools used for testing) [31]. Each profile has 6 levels of increasing defect removal capability with ‘Very Low’ as the least effective and ‘Extra High’ being the most effective in defect removal [Table 3, 31]. The Very Low defect removal rating yields high delivered defect density, on the other hand an Extra High rating can reduce the delivered defect density [13].

Authors in [31] formulated the initial version of their defect removal expression for any single artifact \( j \) as:

\[ \text{DRes}_{EST,j} = C_j \cdot \text{DI}_{EST,j} \cdot \prod_{i} (1 - \text{DRF}_{ij}) \]

Where \( \text{DRes}_{EST,j} \) is estimated residual defects for \( j \)th artifact, \( C_j \) is Calibration Constant for the \( j \)th artifact \( \text{DI}_{EST,j} \) Estimated number of defects of artifact type \( j \) introduced \( i = 1 \) to 3 for each DR profile, and \( \text{DRF}_{ij} \) is Defect Removal Fraction for defect removal profile \( i \) and artifact type \( j \)[31].

CoQualMo’s sub models can be integrated with Cocomo II by adding up defect removal profile levels with Cocomo II attributes to predict the number of non-trivial residual requirements, design and code defects along with the size estimates [31]. This model provides well enough insight into determining the impacts of defect introduction and removal on quality.

3.2 iDAVE

Information Dependability Attribute Value Enhancement (iDAVE) model is developed as a value-
based model with the objectives to identify an appropriate dependability investment level(s) for a software project with dependability requirements as different stakeholders depend on different system capabilities (such as availability, safety, or security) in different situations [41]. iDAVE estimates the Return On Investment (ROI) [12] to achieve desired values for dependability attributes starting from the baseline dependability investment level. iDave model provides a quantitative practice for stakeholders to establish their satisfactory and preferred levels of availability requirements for different software/scenario based on their different ROI [40].

The basic iDAVE is the amalgamation of cost estimating relationships (CER’s) from the Constructive Cost Model COCOMO II, dependability estimating relationships (DER’s) from the Constructive Quality Model COQUALMO, and value estimating relationships (VER’s) supplied by the system’s stakeholders [40].

Cost Estimating Relationships CER’s from Cocomo II facilitates to express time-phased information processing capabilities, and to estimate time-phased investment costs in terms of size and other project attributes [40]. Dependability-attribute-estimating relationships DER’s from the COQUALMO facilitates to specify time-phased levels of investment in improving dependability attributes, from which COQUALMO estimates the resulting time-phased dependability attribute levels [41]. Finally, the iDAVE model needs initial dependability value estimating relationships (VER’s) to relate estimated cost investments and dependability levels to resulting benefit flows and return on investment (ROI) estimates [40][41].

For more information on the overall structure of iDAVE see [14] [40] [41].

3.3 Constructive Rapid Application Development Model (CORADMO)

CoRADMo is a Cocomo II extension that focuses on software development cost by using RAD techniques. CORADMO estimates the schedule (months), Personnel and Effort (person month) based on the distribution of effort and schedule to the various stages, and impacts of the selected schedule [52]. The

CoRADMo model has following five drivers having its rating level and environment to be used [4].

Reuse and Very High Level Language (RHVL) reflects the impact of reuse of code and/or the use of very high level languages specifically during Inception and Elaboration stages.

Development Process Reengineering and Streamlining (DPRS) captures the degree to which the project and organizations allow streamlined or re-engineered development processes.

Collaboration Efficiency (CLAB) captures the possible reduction in schedule and effort due to the effective collaborations among team and team members.

Architecture / Risk Resolution (RESL) enables parallel construction activities without the COCOMO II assumed effect of increased integration and testing costs.

Prepositioning Assets (PPOS) reflects the degree to which assets are pre-tailored and furnished to the project for use on demand.

For more details on Coradmo drivers refer [52].

CORADMO needs a phased schedule and effort distribution because the effects of the RAD strategies are different for the different stages in life cycle [54] [4]. This input is acquired from another Cocomo suite model that is Constructive Phased Schedule and Effort Model (COPSEMO) which intended to estimate schedule in months, personnel required and adjusted effort in person months. COPSEMO is based on the distribution of effort and schedule to the various stages, and impacts of the selected schedule driver ratings on the schedule, Personnel and Effort [4]. Cocomo suite also includes a senior management strategic planning decision assistant model supported by an implementation approach. It is Constructive Productivity-Improvement Model (COPROMO) which is based on the use of COCOMO II and CORADMO as valuation mechanisms [53]. For more details on COPROPMO and COPSEMO visit [52].

3.4 COPLIMO

Most of the models today usually underestimate Return On Investment (ROI) for product lines and apply writing for reuse to the entire product. Barry Boehm addresses these shortfalls in his Constructive Product Line Investment Model (COPLIMO) which is based on relative cost of reuse and relative cost of writing reuse. The focus of
COPLIMO is on the modeling of proportions of the software that product-specific newly built software, fully reused black box product line component and product line component that are reused with adaptation [21]. COPLIMO assumes the use of a set of assets for building a set of related products and relies on the availability of a range of parametric values that must be accurately calibrated [25].

COPLIMO is sectioned into basic life cycle model and an extended life cycle model. Basic model assumes that a change to a product causes the same percentage of change on reused code, adapted code and product unique code [28]. Basic model has two sub models namely development model for product line evolution and post development model for product line. Extended life cycle allows products to have different parameter values [28].

4. OTHER INDEPENDENT ESTIMATION MODELS IN COCOMO SUITE

Authors in [12] have grouped 4 independent models in Cocomo suite namely CoCOTS, CoSYSMo, CoSoSiMo and Costing Secure Systems 2004. Costing Secure Systems is not included in our discussion due to lack of data availability. Rest of these models is discussed below.

4.1 Constructive Commercial-Off-The Shelf CoCOTS Integration Cost Model

COCOTS refers to reasonably predict the lifetime cost of Commercial-off-the-shelf COTS pre-built components that are integrated in any existing software system to enhance its capabilities [22]. Authors in [22] have discussed following sub models of COCOTS:

Assessment is the process by which COTS components are selected for use in the larger system being developed.

Tailoring refers to those activities that would have to be performed to prepare a particular COTS program for use regardless of the system into which it is being incorporated or even if operating as a stand-alone item.

Glue code development and testing refers to the new code external to the COTS component itself that must be written in order to plug the component into the larger system.

Volatility refers to the frequency with which new versions or updates of the COTS software being used in a larger system are released by the vendors over the course of the system's development and subsequent deployment.

COCOTS is gradually progressing, however, the equations provided by USC(University of Southern California) COCOTS model version 1.0 to predict size, cost and effort in Person Months is given as [29].

\[
\text{ESIZE} = \text{UFP} \times (1.0 + \frac{\text{BRAK}}{100})
\]

\[
\text{PM} = A \times (\text{ESIZE})^B \times \prod_{i=1}^{13} (\text{Em}_i)
\]

Cost = (PM) \times (Average Labor rate per Person Month)

Where

UFP = estimated sizing of the COTS glue code in Unadjusted Function Points.
BRAK = estimated percentage of glue code breakage during development
ESIZE = effective size of the developed glue code.
A = a linear scaling constant calibrated to provide an accurate effort estimate
B = a nonlinear scaling constant that accounts for the influence of factors that have exponential rather than multiplicative affects.
This is temporarily set = 1
EM = the thirteen effort multipliers or cost drivers, each of which assumes one of
PM = the estimated effort in person-months for the COTS integration task.

4.2 CoSoSiMo

Many organizations today are concerned for more capable systems to cope with the emerging needs. One way to enhance system capabilities is the integration of any existing system with some other network centric system of systems (S-o-S) [42]. One may perceive the system-of systems as a set of systems comprised of some combination of other systems and system elements [43]. Normally the early presented cost models or tools were only focused on the development activities of a single system, hence can not properly be applied to precisely estimate the Lead System Integrator (LSI) effort associated with architecting the SoS framework [44]. When the scope of the system is specification and integration of two or more separate systems then Constructive System-of-Systems (SoS) Integration Cost Model (COSOSIMO) is the option [45] that is designed to estimate the effort associated with the Lead System Integrator (LSI) activities to define the SoS architecture, identify
sources to either supply or develop the required SoS component systems, and eventually integrate and test these high level component systems [42] [46].

The first version of COSOSIMO [Lane 2004] was based upon expert judgment of key members of the COCOMO Working Group and a hierarchical view of SoS architectures [42]. It is a two-tiered model to calculate the number of integration person months (IPM) required for the SoS LSI effort [42]:

\[
\text{Level 1 IPM (S_i)} = A_1 \left( \sum_{j=1}^{n_i} S_{ij} \right)^{B_i} \\
\text{Level 0 IPM (SoS)} = A_0 \left( \sum_{j=1}^{m} \text{IPM (S_j)} \right)^{B_0}
\]

Where the size of the subsystem within the SoS, \( S_{ij} \) is derived from historical project data, \( n_i \) is the number of subsystem level 2 components comprising the ith subsystem, \( m \) is the number of subsystem level 1 components comprising the SoS, \( B_i \) is the effort exponent for the ith subsystem based on the subsystem’s exponential scale factors and \( B_0 \) is the effort exponent for the SoS based on the SoS’s exponential scale factors.

The initial scale factors for CoSoSiMo included:

- Integration Simplicity
- Integration Risk resolution
- Integration Stability
- Component Readiness
- Integration Team Capability
- Maturity of the integration Process

Recent calibrations in COSOSIMO have resulted in three sub-models namely: Planning/Requirements Management/Architecture (PRA) sub-model; Source selection and Supplier oversight (SS) sub-model; and an SoS Integration and Testing (I&T) sub-model [47][48]. The summary of these three sub-models is presented in table 3 including their size and cost drivers. Information in table 3 is obtained from [47] and can be referred for details on size and cost drivers in CoSoSiMo sub-models.

**Table 3: CoSoSiMo Sub-models Summary**

<table>
<thead>
<tr>
<th>Sub-Model</th>
<th>Activities associated with</th>
<th>Cost Drivers</th>
<th>Size Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Selection and Supplier Oversight</td>
<td>Identification of system suppliers and vendors, the development of Requests for Proposals,</td>
<td>1. Requirements Understanding</td>
<td>2. Number of SoS Interface Protocols.</td>
</tr>
<tr>
<td>Integration and Testing I&amp;T</td>
<td>Integration and Test planning, set up of the integration and test</td>
<td>1. Requirements Understanding</td>
<td>2. Number of Unique Component Systems</td>
</tr>
</tbody>
</table>
environments and tools, development of test data and procedures, and the actual execution and tracking of integration, verification/validation tests

| Effort is currently underway to review and calibrate the model. This model is a good effort in order to support activities for estimating the LSI effort that may allow users to develop initial estimates and then accomplish tradeoffs based on architecture and development process alternatives [44]. |

| **3. Level of Service Requirement.** |
| **4. LSI I&T Team Cohesion.** |
| **5. LSI I&T Team Capability** |
| **6. LSI I&T Process Maturity** |
| **7. I&T tool Support** |
| **8. I&T process cost/Schedule Compatibility** |
| **9. SoS I&T Risk Resolution** |
| **10. Component System Maturity and Stability.** |
| **11. Component System Readiness** |

### 4.3 CoSYSMo

The objective of CoSYSMo (Constructive Systems Engineering Cost Model) is to provide accurate, pragmatic and tailor able estimates for time and effort related with system engineering tasks provided by ISO/IEC 15288 [49]. CoSYSMo has gone through three iterations discussed below [50].

The first version of COSYSMO (Straw man CoSYSMo) contained 16 systems engineering cost drivers. The factors identified were named application factors and team factors. Function points and use cases are recognized as measures of systems engineering functional size. CoSYSMo-IP is 2nd version which is a revised set of cost drivers incorporated measures for functional size that were independent of the software size. This version has provided three different types of parameters namely additive, multiplicative, and exponential. The current version, referred simply as COSYSMO is designed to estimate the number of person months as a function of a system’s functional size with considerations of diseconomies of scale. The size and cost drivers were determined with a Delphi approach (For details on cost and size drivers refer [51]) which resulted in the emergence of following regression equation of the model [49] [50].

\[
\text{Effort} = A \prod c_i (\text{Size})^p
\]

Where A is calibration constant, \(c_i\) is calculated as the product of i cost constant, \(\text{Size}\) represents economies/diseconomies of scale, and \(\text{Size}\) is the weighted average of the size predictors.

### 5. DISCUSSION & SUMMARY

This paper is an attempt to portray the primary methodologies of Cocomo suite models, logic behind their development and how they can be used to support software estimation needs. Different models over the years have been proposed to support the software cost estimation processes. Cocomo is one of the successful models which enabled software engineers to reliably reason about the effort and schedule estimates. As the need, scope and complexity of the software development has grown, the demands towards recalibrations also have arisen to meet these challenges. This growth ultimately results into the calibrations in Cocomo by proposing derivatives and extensions to meet with different aspects of software development like System engineering, SoS architecting and engineering, COTS integration etc. These derivations and extensions are grouped in Cocomo suite. Our study has presented an overview of the models in the Cocomo suite, their methodology, scenario for their possible use, logic behind their need and the way in which they are developed [12].

The models in Cocomo suite provide a specialized set of estimates to address specific aspects of development, hence understanding the scope of each model is a key factor to recognize the service or output it provides. Nearly every model in Cocomo suite is developed independently and no model in Cocomo suite covers the full life cycle effort for today’s large software intensive systems, which elevate complications when they are integrated. Although they can work in parallel but at certain stages overlapping is seen, for example Cocomo II is designed to estimate software effort associated with analysis and design, implementation and test, CoSYSMo estimates effort associated with system level engineering activities.

It is implied that an approach may be adopted to discern between integrated set of models against unified model. To investigate a improved model to accomplish comprehensive results with the center of attention on integration of software and system engineering aspects, compatible on different platforms, flexible to accommodate
changes and future trends will be a high-quality effort rather to put forward overlapping and scenario specific set of models. Perhaps it is safe to say that different techniques may be amalgamated for identifying the best attributes among them. Possibly it would be productive if researchers calibrate other formal estimation techniques, like Farr and Zagorski, Walston and Felix [58], Wolverton model [2], Kustanowitz model [59], Putnam [3] and many more, rather to propose methods to replace them.

REFERENCES


[22] C. Abts, B. W. Boehm, and E. B. Clark, “Empirical Observations on COTS software Integration Effort Based on the initial COCOTS calibration Database,” In


[56] http://www.seisage.net/jensen_model.htm

