Volume 2, Issue 5, May 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Effectual Software Design using Software Matrices and COCOMO-II

Dheeraj Agrawal
Department of computer science
SSCET, Bhilai
dheeraj_arg@rediffmail.com

Megha Mishra
Department of computer science
SSCET, Bhilai

*Abstract— **Software engineering means the application of a systematic, disciplined and quantifiable approach to the development, operation, and maintenance of software. As we are becoming more and more reliant on software systems, more effort is being put on development of efficient, reliable and cost-effective software. Software measurement helps us understand software, manage a software project and improve the software process itself. We believe a method of defining software data sets is necessary to ensure that software data are trustworthy. Software companies introducing a measurement program need to establish procedures to collect and store trustworthy measurement data. Without appropriate definitions it is difficult to ensure data values are repeatable and comparable. Software quality analysis of program modules with identifies defect or quality-based class labels. Hence it is vital to develop a tool for effectual software design. This paper presents our approach on design a web based tool for integrating the estimation (using COCOMO II model), planning & tracking, calibration process and software matrices used to evaluate software defects. Software design patterns based on C&K and Mood matrices.***

***Keywords**— COCOMO-II, Matrices, Software design, Estimation, Software Defects.*

I. INTRODUCTION

Software development time and cost estimation are the process of estimating the most realistic use of time and cost required for developing a software. Cost estimation methodology for web-based application is very important for software development as it would be able to assist the management team to estimate the cost. Furthermore, it will ensure that the development of cost is within the planned budget and provides a fundamental motivation towards the development of web-based application project [21] [22]. Software development effort typically includes human effort expended for high-level design, detailed design, coding, unit testing, integration testing, and customer acceptance testing. Effort is often regarded as a surrogate for software development cost since personnel cost is the dominant cost in software development [15]. When project is under development it is necessary to take feedback from the development process and analyse the status of project.

The COCOMO II capability for estimation of Application Generator, System Integration, or Infrastructure developments is based on Application Composition model (for early prototyping efforts) and two increasingly detailed estimation models for subsequent portions of the life cycle, Early Design and Post-Architecture.

In estimating a cost, the tool considers all variables in COCOMO II and requires expert judgment to key in the input of the variables such as project size, project type, cost adjustment factor and cost driven factor [22].

The measurement of software quality is traditionally based upon 1) complexity and 2) design quality Metrics. The first research contributions were aimed at providing operating definitions and metrics of software complexity, focusing on the analysis of the code's information flow [14].

The object-oriented programming paradigm, coupling, cohesion, inheritance, and information hiding have been identified as the basic properties of software design quality [9], [14], [23], [24]. Based on these four basic properties, a number of metrics have been proposed to evaluate the design quality of object-oriented software. The most widely known metrics were first proposed by Chidamber and Kemerer [25] (WMC, NOC, DIT, RFC, LCOM, and CBO) and by F.B. Abreu [9] (COF, PF, AIF, MIF, AHF, and MHF).

Software defects play a key role in software reliability, and the number of remaining defects is one of most important software reliability indexes. Observing the trend of the number of remaining defects during the testing process can provide very useful information on the software reliability [4].

Software tools can improve the quality and maintainability of software, but are expensive to acquire, deploy, and maintain, especially in large organizations individually [21].

II. LITERATURE REVIEW

A. COCOMO II

COCOMO II follows the openness principles used in the original COCOMO. Thus, all of its relationships and algorithms will be publicly available. Also, all of its interfaces are designed to be public, well-defined, and parameterized, so that Complementary pre-processors (analogy, case-based, or other size estimation models), post-processors (project planning and Control tools, project dynamics models, risk analysers), and higher level packages (project management packages, product Negotiation aids), can be combined straightforwardly with COCOMO II. To support the software marketplace sectors above, COCOMO II provides a family of increasingly detailed software cost estimation models, each tuned to the sectors' needs and type of information available to support software cost estimation.

COCOMO II provides the following three-stage series of models for estimation of Application Generator, System Integration, and Infrastructure software projects:

1. The earliest phases or spiral cycles will generally involve prototyping, using the Application Composition model capabilities. The COCOMO II Application Composition model supports these phases, and any other prototyping activities occurring later in the life cycle.
2. The next phases or spiral cycles will generally involve exploration of architectural alternatives or incremental development strategies. To support these activities, COCOMO II provides an early estimation model called the Early Design model. This level of detail in this model is consistent with the general level of information available and the general level of estimation accuracy needed at this stage.
3. Once the project is ready to develop and sustain a fielded system, it should have a life-cycle architecture, which provides more accurate information on cost driver inputs, and enables more accurate cost estimates. To support this stage, COCOMO II provides the Post-Architecture model.

COCOMO II is a fully documented and widely accepted model, updated from the original COCOMO. COCOMO II uses TOOL, one of the Effort Multipliers in its Post-Architecture model to capture the impacts on software productivity and provides a TOOL rating scale based just on the completeness of tool coverage as a guideline for the evaluation of software tools [6].

B. Metrics and Measures

A metric is a quantitative measure of the degree to which a system, component, or process. Software measurement applies to a software engineering process there by measuring numerous entities encountered along the way. According to Dumke [28], software measurement is directed to three main components in the object-oriented software development.

the process measurement for understanding, evaluation and improvement of the development method, the product measurement for the quantification of the product (quality) characteristics and validation of these measures, the resource measurement for the evaluation of the supports (CASE tools, measurement tools etc.) and the chosen implementation system [7].

C. Chidamber & Kemerer Metrics Suite

This metrics suite was proposed in [25] by S. R. Chidamber and C. F. Kemerer. The structural design metrics proposed by them are explained here.

- Weighted Method per Class (WMC)

It is sum of complexities of all methods in a class. Consider a class C1 with methods M1, . . . ,Mn that are defined in the class. Let c1, . . . cn be complexities of each of these methods.

For this work, complexity of each method is assumed to be unity and so WMC is simply sum of all defined methods.

C&K-Java Binding: This work considers WMC as count of all defined methods inside a class with any access modifier. This does not include inherited and abstract methods. This is because inherited methods do not actually belong to this class. Abstract methods do not have a body and so no complexity measure is possible for them.

- Depth of Inheritance Tree (DIT)

Depth of inheritance of the class is the DIT metric for the class.

C&K-Java Binding: This study takes DIT as the maximum length of the inheritance tree up to the root. A class may implement an interface and that interface may extend one or more interfaces.

- Number of Children (NOC)

Number of immediate sub-classes subordinated to a class in class hierarchy. C&K-Java Binding: It is the number of immediate subclasses of a class. For an interface it is the number of classes implementing it plus number of other interfaces extending this interface.

- Coupling between Objects (CBO)

CBO for a class is count of the number of other classes to which it is coupled. Two classes are coupled together if methods of one use methods or instance variables of other. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

C&K-Java Binding: A class can call methods from another class either through inheritance or using an object of the other class. CBO should measure both forms of these couplings.

- Response for a Class (RFC)

RFC = | RS | where RS is response set for the class. This is a set of methods that can potentially be executed in response to a message received by an object of that class. Since it specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

C&K-Java Binding: This includes all defined and inherited methods inside this class plus methods called on objects of other classes in any method of this class.

- Lack of Cohesion in Methods (LCOM)

The LCOM is a count of the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. The degree of similarity for two methods M1 and M2 in class C1 is given by: s() = {I1}\{I2} where {I1} and {I2} are the sets of instance variables used by M1 and M2 The larger the number of similar methods, the more cohesive the class, which is consistent with traditional notions of cohesion that measure the inter-relatedness between portions of a program. A high cohesion is favoured in class designs.

C&K-Java Binding: Instance variables are the ones with any access modifier.

D. MOOD Metrics Set

F. B. Abreu proposed these system-level metrics in [27]. This set of six metrics measures four main structural mechanisms of object-oriented design that is encapsulation (Method Hiding Factor and Attribute Hiding Factor), inheritance (Method Inheritance Factor and Attribute Inheritance Factor), polymorphism (Polymorphism Factor) and message-passing (Coupling Factor).An explanation of the metrics with Java bindings follows except for coupling factor which was not measured. Common Java Binding Note: This set of metrics applies to system level. While measuring these metrics for standard libraries like J2SE etc.

- Method Hiding Factor (MHF)

MOOD-Java Binding: TC– total number of classes in the system/package. Md(Ci)– number of constructors and methods defined with any access modifier excluding abstract and inherited methods.

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

where:

$$V(M_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1}$$

and:

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1 & \text{iff j} \neq \text{i and } C_j \text{ may call } M_{mi} \\ 0 & \text{otherwise} \end{cases}$$

- Attribute Hiding Factor (AHF)

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}$$

where:

$$V(A_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(A_{mi}, C_j)}{TC - 1}$$

and:

$$is\_visible(A_{mi}, C_j) = \begin{cases} 1 & \text{iff j} \neq \text{i and } C_j \text{ may reference } A_{mi} \\ 0 & \text{otherwise} \end{cases}$$

MOOD-Java Binding: Ad (Ci) – number of all attributes with any access modifier but not including inherited.

- Method Inheritance Factor (MIF)

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

Where Ma (Ci) = Md (Ci) + Mi (Ci)

The numerator is the sum of inherited methods in all classes of the system. The denominator is the total number of available methods in all classes.

MOOD-Java Binding: Mi (Ci) – number of inherited methods but not overridden, Md (Ci) – number of defined non-abstract methods with any access modifier, Ma (Ci) – number of methods that class Ci can call.

- Attribute Inheritance Factor (AIF)

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

Where, Aa (Ci) = Ad (Ci)+Ai (Ci)

It is defined analogous to MIF.

MOOD-Java Binding: Ai (Ci) – number of inherited attributed, Ad (Ci)– number of defined attributes with any access modifier, (Ci)– number of attributes that class Ci can reference.

E. Software Defect Estimation

Software defects play a key role in software reliability, and the number of remaining defects is one of most important software reliability indexes. Observing the trend of the number of remaining defects during the testing process can provide very useful information on the software reliability. However, the number of remaining defects is not known and has to be estimated [9]. Therefore, it is important to study the trend of the remaining software defect estimation (ISCA algorithm describe next section).

There has been some research on the trend of the software defects. Early studies of defect occurrences suggest that it follows a Rayleigh curve [2], [30], [31] roughly proportional to project staffing. McConnell [32] discusses the relationship between defect rate and development time, indicating that the projects achieving the lowest defect rates also achieve the shorts schedules.

F. Design Patterns

Design patterns describe good solutions to common and recurring problems in software design. They have been widely applied in many software systems in industry. However, pattern related information is typically not available in large system implementations. Recovering these design pattern instances in software systems can help not only to understand the original design decisions and tradeoffs but also to change the systems with quality assurance the design patterns using an XML file, which include their structural, behavioural and semantic characteristics. These pattern characteristics are used in different phases. During structural analysis phase, our tool extracts the structural information of the pattern and encodes it into a matrix and weights in a similar way as we encode the system. Thus, the structural analysis can be reduced to the matching of the design pattern matrix with the system matrix as well as the weights of the design pattern classes with the weights of the system classes [1], [11].

III. APPROACH OVERVIEW

The system will automate the process of the estimation using the COCOMO II model [21] for effort estimation. The system will also help in tracking the status of project by tracking daily input from each developer in the organization and will show the status in the form of a Gantt chart. The system will generate the reports for the projects. Identify Software defects play a key role in software reliability, investigates an approach to incorporate the time dependencies between the fault detection, and the number of remaining defects is one of most important software reliability indexes. Administrator control overall system module.

A. System Overview

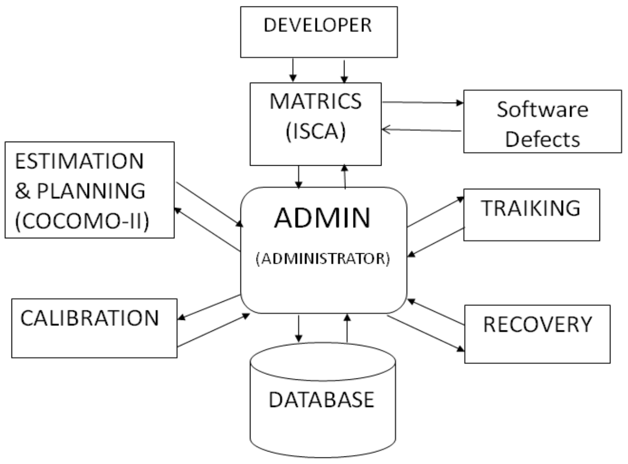

Fig.1 A Proposal of System Design Tool

B. Development Phase

The proposed System has three development phases.

1. Phase I

Phase I was dedicated to the database design, designing the system and for developing the part which estimates the size, effort and schedule for the project along with the programs for inserting the data into the backend and for its manipulation. An interactive and user friendly interface with an accurate estimation model was the goal of this phase [12].

1. Estimation of the size of the intended project. This results in either source lines of code (SLOC) or function point counts (FPC) or new object points (NOP) for the project but other measures for the size are also available.

2. Estimation of the effort for the project in man-months or man-hours.

3. Estimation of the schedule in calendar-months.

The information source for estimation can be the project proposal, system specification or software requirement specification. If the size estimation is being done in the later stages such as design or during coding, then design specifications and other work products can be used as information source for estimation [12], [7].

1. By Analogy: If similar projects have been experienced by the organization then with the help of past experience the size for the new project can be estimated. This is performed by dividing the new project into small modules and comparing those modules with the past project data. This method can give almost the accurate estimate for the project size if the past projects were similar to the new one [19].

2. By Parametric Measurement: The size could be estimated by counting features of the project and using them as parameter for any parametric measurement approach like object point analysis or function point analysis. Even if the organization has no experience of the intended project, the features of the project can be used for parametric measurement.

2. Phase II

Phase II develop a Metrics Attributes Calculation Module (MACM). These modules are used to Calculate Attributes of Mood and Ck matrices.

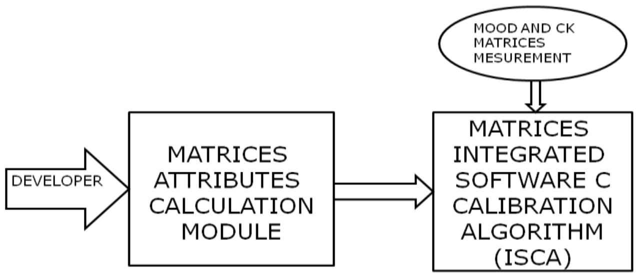
MOOD AND CK MATRICES MESUREMENT
DEVELOPER
MATRICES ATTRIBUTES CALCULATION MODULE
MATRICES INTEGRATED SOFTWARE C CALIBRATION ALGORITHM (ISCA)

Fig. 2 Design a tool for calculating the Matrices attribute.

3. Phase III

Phase III Design an Algorithm Integrated Software Calibration Algorithms (ISCA). Using ISCA algorithm to identify software Defects. Major work was done in this phase.

```
Procedure ISCA ()
STEP-1  Gen_Matric ()
STEP-2  Gen_Multidimensional_Matric ()
STEP-3  Cal_Defect ()
End procedure
```

STEP-1 Gen_Matric () Algorithm

```
Algorithm- Gen_Matric ()
//* WMC= Total number of Methods/total number of
class
MHF, AHF, MIF, AIF, CIF, CF
Programmer 1, 2, 3, 4, 5……………………..N.
Mat [1………N][1………N]
*//
Step1   i ← 1
Step2   loop i <=N
Step3    j ← 1
Step4   Mat[i][j] ← LOC i,j
         j ← j+1
        Mat[i][j] ← WMC i,j
         j ← j+1
        Mat[i][j] ← MHF i,j
        j ← j+1
        Mat[i][j] ← AHF i,j
         j ← j+1
        Mat[i][j] ← AIF i,j
         j ← j+1
        Mat[i][j] ← CF i,j
        If j ≤ N Then
        break
Step6           end Step4 loop
Step7           i ← i+1
Step8           end Step3 loop
         End procedure
```

STEP-2 Gen_Multidimensional_Matric () Algorithm

```
Algorithm- Gen_Multidimensional_Matric
Procedure Gen_Multidimensional_Matric (no_of_days, mat)
Step1   NOD ← no_of_days
//* Mat [1……N] [1……..N] *//
Step2   i ← 1
Step3   j ← 1
Step4   loop i<=N
Step5       loop j<=N
Step6           Mat[i][j] ← Mat[i][j]/NOD
Step7           j ← j+1
Step8       end Step4 loop
Step9       i ← i+1
Step10  end Step3 loop
End procedure
```

STEP-3 Cal_Defect ()

```
Procedure Defect (Mat[i][j])
Step1   j ← 1
Step2   While (j<=N)
Step3       i ← 1
Step4       While (i<=N)
Step5           Select (j)
Step6           Case 1
                    SumDIT ← SumDIT + Mat[i][j]
                    Break
                Case 2
                    SumWMC ← SumWMC + Mat[i][j]
                    Break
                Case 3
                    SumMHF ← SumMHF + Mat[i][j]
                    Break
                Case 4
                    SumAHF ← SumAHF + Mat[i][j]
                    Break
                Case 5
                    SumMIF ← SumMIF + Mat[i][j]
                    Break
                Case 6
                    SumAIF ← SumAIF + Mat[i][j]
                    Break
                Case 7
                    SumCF ← SumCF + Mat[i][j]
Step7           End Select
Step8           i ← i+1
Step9       End While
Step10      j ← j+1
Step11  End While
Step12  SumDIT ← SumDIT / N
        SumWMC ← SumWMC / N
        SumMHF ← SumMHF / N
        SumAHF ← SumAHF / N
        SumMIF ← SumMIF / N
        SumAIF ← SumAIF / N
        SumCF ← SumCF / N
Step13  If (SumDIT ≥ 0.5 and SumDIT≤1)
        And
        If (SumMHF ≥ 0.5 and SumMHF ≤1)
        And
        If (SumAHF≥ 0.5 and SumAHF ≤ 1)
        And
        If (SumMIF ≥ 0.5 and SumMIF ≤ 1)
        And
        If (SumAIF ≥ 0.5 and SumAIF ≤ 1)
        And
        If (SumCF ≥ 0.5 and SumCF ≤ 1)
Step14      PRINT "NO DEFECT"
Step15  Else PRINT "DEFECT"
Step16  END IF
```

IV. RESULT OF ISCA

Anecdotal and empirical evidence reported in the literature suggest, such as reduced time to market, reduced development costs, improved quality of the software, reduced costs of planning , and enhanced trust, motivation, and information and knowledge transfer among developers and project leader.

Fig 3 provides the comparison of earlier and our approach. It shows that the ISCA significantly reduce software defects and reduce software cost.

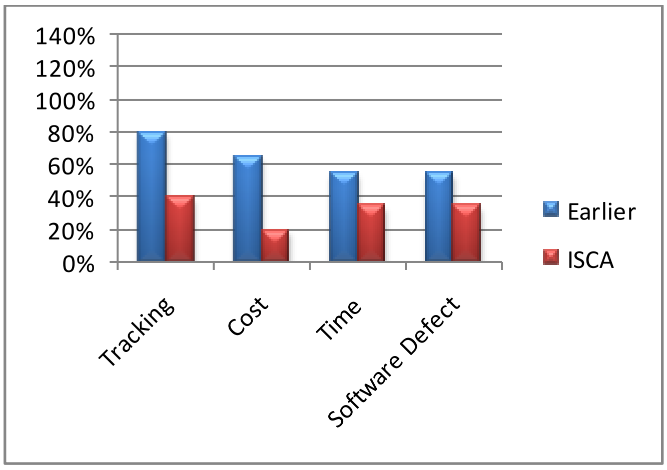

Fig. 3 Comparison of earlier Approach and ISCA.

V. CONCLUSION

Software estimation helps project management to plan the project. Tools available for the project estimation are great helps in the process. But estimating the project and then planning it without caring about the status of project at any instant of time is a problem worth to be considered. The process known as tracking is an important process that needs to be integrated with the estimation and planning process. The core of software crisis starts with the wrong estimation. We are introducing software matrices approach to calibrate software. ISCA algorithm tracks the developer and with various software matrices attribute are used to find out software defects cohesion, coupling etc.

REFERENCES

[1] Jing Dong, Senior Member, IEEE, Yajing Zhao, and Yongtao Sun, "A matrix based approach to recovering design pattern," IEEE

TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, VOL. 39, NO. 6, NOVEMBER 2009.

[2] Naeem Seliya, Member, IEEE, and Taghi M. Khoshgoftaar, "software quality analysis of UP Modules," IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, VOL. 37, NO. 2, MARCH 2007.

[3] Anandasivam Gopal, Tridas Mukhopadhyay, and M.S. Krishnan, "the impact of on software matrices," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 8, AUGUST 2005.

[4] Cheng-Gang Bai, Kai-Yuan Cai, Qing-Pei Hu, and Szu-Hui Ng, "on the trend of remaining software defect estimation," IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, VOL. 38, NO. 5, SEPTEMBER 2008.

[5] Y. P. Wu,Q. P.Hu,M. Xie and S. H. Ng, Member, IEEE, "modelling and analysis of software fault detection and correction process by time dependency," IEEE TRANSACTIONS ON RELIABILITY, VOL. 56, NO. 4, DECEMBER 2007.

[6] Jongmoon Baik, Barry Boehm and Bert M. Steece, "disaggregating and calibrating the case tool variable in cocomo II," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 28, NO. 11, NOVEMBER 2002.

[7] Hilda B. Klasky, "A STUDY OF SOFTWARE METRICS," thesis submitted to the Graduate School-New Brunswick New Brunswick, New Jersey May, 2003.

[8] Barbara A. Kitchenham,Robert T. Hughes, and Stephen G. Linkman, "Modeling Software Measurement Data," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 27, NO. 9, SEPTEMBER 2001

[9] Ayaz Farooq, "Conception and Prototypical Implementation of a Web Service as an Empirical-based Consulting about Java Technologies," Master Thesis October 12, 2005.

[10] Magne Jørgensen and Martin Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 1, JANUARY 2007

[11] Alexander Egyed, "Automatically Detecting and Tracking Inconsistencies in Software Design Models," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 2, MARCH/APRIL 2011.

[12] Nilesh Chandra Shukla, "A tool for software project management for estimation, planning & tracking and calibration," Indian institute of information technology Allahabad jun,2007.

[13] Chiaming Yen, Wu-Jeng Li, and Jui-Cheng Lin "A web based collaborative computer aided sequential control design tool," April 2003.

[14] Eugenio Capra, Chiara Francalanci, and Francesco Merlo, "An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 6, NOVEMBER/DECEMBER 2008.

[15] Manish Agrawal and Kaushal Chari, "Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects,"IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 3, MARCH 2007.

[16] Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum, "Selecting Best Practices for Effort Estimation," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 32, NO. 11, NOVEMBER 2006.

[17] Bo Yang, Huajun Hu, and Lixin Jia, "A Study of Uncertainty in Software Cost and Its Impact on Optimal Software Release Time," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 6, NOVEMBER/DECEMBER 2008.

[18] C. van Koten, "An Effort Prediction Model for Data Centred Fourth Generation Language Software Development," The Information Science, Discussion Paper Series Number 2003/04 ISSN 1172-6024.

[19] Ning Nan and Donald E. Harter, "Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 35, NO. 5, SEPTEMBER/OCTOBER 2009.

[20] Kal Toth, "Experiences with Open Source Software Engineering Tools,"November/December 2006 IEEE SOFTWARE.

[21] Attarzadeh, I.; Ow, "Improved estimation accuracy of cocomo using an adaptive fuzzy logic model," S.H. Publication Year: 2011 IEEE CONFERENCE PUBLICATIONS.

[22] Bin Mansor, Z, "E-cost estimation using expert judgment and COCOMOII," appears in information technology (ITSim) 2010 International symposium in volume 3.

[23] F. Brito e Abreu, "The MOOD Metrics Set," Proc. ECOOP Workshop Metrics, 1995.

[24] J.Y. Chen and J.F. Lu, "A New Metric for Object-Oriented Design," J. Information System and Software Technology, vol. 35, no. 4, pp. 232-240, 1993.

[25] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476-493, June 1994.

[26] Reiner R. Dumke and Erik Foltin. Metrics-based evaluation of object-oriented software development methods. In CSMR, pages 193–196, 1998.

[27] F. B. Abreu and Walcelio Melo. Evaluating the impact of object-oriented design on software quality. In METRICS '96: Proceedings of the 3rd International Symposium on Software Metrics, page 90, Washington, DC, USA, 1996. IEEE Computer Society.

[28] Reiner R. Dumke and Erik Foltin. Metrics-based evaluation of object-oriented software development methods. In CSMR 1998.

[29] D. N. Card, "Managing software quality with defects," in Proc. Comput. Softw. Appl. Conf., Aug. 2002, pp. 472–474.

[30] C. Y. Huang, S. Y. Kuo, and I. Y. Chen, "Analysis of a software reliability growth model with logistic testing-effort function," in Proc. 8th Int. Symp. Softw. Rel. Eng., Nov. 1997, pp. 378–388.

[31] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, Software Cost Estimation with COCOMO II. Prentice Hall, 2000.

[32] S. McConnell, "Software quality at top speed," Softw. Develop., vol. 4,no. 8, pp. 38–42, Aug. 1996.