

# Predicting Understandability of a Software Project Using COCOMO II Model Drivers

Ali Afzal Malik, Barry W. Boehm, A. Winsor Brown

Center for Systems and Software Engineering – University of Southern California  
Los Angeles, U. S. A.

alimalik@usc.edu, boehm@usc.edu, awbrown@usc.edu

***Abstract.** This paper presents the results of an empirical study undertaken to investigate the utility of COCOMO II model drivers in predicting the understandability of a software project. Understandability is defined as the degree of clarity of the purpose and requirements of a software system to the developers of that system at the end of the Inception phase. COCOMO II scale factors and cost drivers relevant for prediction are shortlisted and a weighted-sum formula relating these model drivers to understandability is derived through voting. The utility of this formula is judged by examining the COCOMO II model drivers of 24 real-client, MS-student, team projects done at USC. It is found that the weighted-sum formula correctly predicts the understandability of a software project in more than 80% of the cases suggesting a strong relationship between the shortlisted COCOMO II model drivers and the understandability of a software project. This objective way of measuring the understandability of a software project can be extremely useful in determining the time when it is safe to minimize the effort spent on requirements engineering activities.*

## 1. Introduction

As shown graphically in the famous RUP hump chart [Kruchten 2003], the activities related to Requirements Engineering span all phases and iterations of a software development project. The bulk of these activities, however, occur in the Inception and the Elaboration phases. This is when efforts are undertaken to formulate the project's goals and define its scope. The high-level goals are later refined into low-level implementable requirements. The problem, however, is that despite all these efforts there are no guarantees that the team implementing the project has understood its requirements well. A technique which enables one to determine how well these requirements have been understood can, therefore, prove to be extremely useful.

In this paper we present such a technique. The basic idea behind our approach is to reuse the inputs of the software cost-estimation process to predict how well the project has been understood by the development team. These estimation activities are undertaken during the Inception phase for purposes of project planning and control. Reusing the inputs of these activities enables one to achieve the desired results without going out of the way.

**Table 1. Projects summary**

<b>S#</b>	<b>Year</b>	<b>Project</b>	<b>Type</b>
1	2004	Bibliographies on Chinese Religions in Western Languages	Web-based database
2	2004	Data Mining of Digital Library Usage Data	Data mining
3	2004	Data Mining from Report Files	Data mining
4	2005	Data Mining PubMed Results	Data mining
5	2005	EBay Notification System	Stand-alone application
6	2005	Rule-based Editor	GUI
7	2005	CodeCount™ Product Line with XML and C++	Code Counter Tool
8	2006	California Science Center Newsletter System	Web-based database
9	2006	California Science Center Event RSVP System	Web-based database
10	2006	USC Diploma Order/ Tracking Database System	Web-based database
11	2006	USC Civic and Community Relations web application	Web-based database
12	2006	Student's academic progress web application	Web-based database
13	2006	New Economics for Woman (NEW)	Web-based database
14	2006	Web Portal for USC Electronic Resources	Web-based GUI
15	2006	Early Medieval East Asian Tombs	Web-based database
16	2006	USC CONIPMO	Cost model
17	2006	An Eclipse Plug-in for Use Case Authoring	Stand-alone application
18	2007	USC COINCOMO	Cost model
19	2007	BTI Appraisal Projects	Stand-alone database
20	2007	LAMAS Customer Service Application	Web-based database
21	2007	BID review System	Stand-alone database
22	2007	Proctor and Test Site Tracking System	Web-based database
23	2007	E-Mentoring program	Web-based application
24	2007	Los Angeles County Generation Web Initiative	Web-based database

Currently, we have tested our approach on data obtained in an academic setting. In particular, we have analyzed 24 two-semester-long team projects [SE I 2008, SE II 2008] done by MS students at USC. Table 1 above gives a summary of these projects. These projects were done by a group of four to six students and each of these projects used the same MBASE/RUP development process [Boehm et al. 2005, Kruchten 2003]. Each project was a development-intensive project (instead of being a customization-intensive project) which used the COCOMO II tool [Boehm et al. 2000] for software cost-estimation.

We have used a subset of the COCOMO II model drivers to come up with a weighed-sum formula for predicting the understandability of a software project at an early stage. Understandability over here refers to the degree of clarity of the purpose and requirements of a software system to the developers of that system at the end of the

Inception phase. Results indicate that our approach correctly predicts the understandability in over 80% of the academic projects we have analyzed. This suggests a strong relationship between our weighted-sum formula and the understandability of a software project.

The remaining paper is organized in the following manner. Section 2 summarizes the most relevant prior work in this area. It also provides the motivation behind this research. Section 3 presents the details of our approach and the methodology we have used in obtaining our data. The results of our approach are contained in Section 4 along with some comments on the utility of these results. Some of the planned future work in this area is contained in Section 5.

## **2. Motivation and Related Work**

According to the 1995 Standish Group report [Standish Group 1995] unclear objectives and incomplete requirements and specifications together account for almost 18% of software project failures. Moreover, as depicted in [Boehm 1981] and as confirmed in [Boehm and Turner 2004], the escalation in the cost of fixing defects in requirements as development proceeds through the various phases of the software development lifecycle is rapid for large projects and considerable for smaller ones. Furthermore, changes in requirements have a significant impact on the budget and schedule of a project [Zowghi and Nurmuliani 2002]. Thus, if the requirements are misunderstood from day one then there is a high risk of overrunning the budget and schedule and of wasting valuable resources due to rework.

An objective mechanism enabling the success-critical stakeholders of a project to determine with some degree of confidence how well the project is understood by the development team, hence, would be extremely useful in minimizing the wastage of resources due to rework. Moreover, it can also be used to indicate when it is safe to minimize the Requirements Engineering activities since once it is determined that a project's requirements are clear then the focus may be shifted to other development activities. In this paper we outline such an objective mechanism.

Our approach banks on the fact that the process of software cost-estimation not only produces useful output in the form of effort, schedule, and staff estimates but also captures valuable information about the software project, the product being developed, the product platform, and the personnel developing the product in the form of its inputs. Thus, leveraging the information contained in these inputs saves a lot of time and hassle. We have found that one use of this information lies in quantifying and predicting the understandability of a software project.

The idea of using software cost model inputs for quantification and prediction of abstract notions is not new. Expert COCOMO [Madachy 1997], for instance, uses COCOMO II cost factors to quantify the risk faced by a software project. This tool uses an automated heuristic to perform risk assessment in conjunction with software cost-estimation activities. We have adopted a fairly similar technique to predict the understandability of a software project.

### 3. Methodology

Apart from the notion of software size, COCOMO II uses twenty-two model drivers as inputs: five scale drivers and seventeen cost drivers [Boehm et al. 2000]. Each scale driver has an exponential impact on the overall effort of a project. With the exception of SCED (Required Development Schedule), each cost driver, is applied at the module level and has a multiplicative impact on effort. Based on our familiarity with the COCOMO II model we identified and shortlisted eight out of the total twenty-two model drivers that were most relevant in predicting understandability. These eight model drivers are given in Table 2 below. The first two (PREC and RESL) are scale drivers while the rest are cost drivers. Out of the six cost drivers, CPLX relates to the software product being developed while the remaining five relate to the personnel developing the product.

**Table 2. Relevant COCOMO II model drivers**

S#	Model Driver	Description
1	PREC	Product precedentedness
2	RESL	Architecture/Risk resolution
3	CPLX	Product complexity
4	ACAP	Analyst capability
5	PCAP	Programmer capability
6	APEX	Applications experience
7	PLEX	Platform experience
8	LTEX	Language and tool experience

The following simple weighted-sum formula was used to relate our shortlisted model drivers with understandability (UNDR):

$$\text{Equation 1: } UNDR = \sum_{i=1}^8 n_i w_i MD_i$$

$MD_i$  represents the value of a model driver while  $w_i$  represents the importance or weight of that model driver in predicting understandability.  $n_i$  is used to capture the nature of a model driver with respect to understandability. It can assume only one of two values - +1 or -1. A value of +1 indicates that a model driver is positively related to understandability. In other words, higher values of that model driver correspond to higher values of understandability and vice versa. A value of -1 for  $n_i$ , on the other hand, indicates that there is a negative relationship between the corresponding model driver and understandability.

The value of each model driver ( $MD_i$ ) can range from ‘Very Low’ to ‘Very High’. In addition, the value of some model drivers (e.g. PREC and RESL) can go up to ‘Extra High’. Table 3 below contains the simple scale we used to convert model driver ratings to numerical values for usage in the weighted-sum formula.

**Table 3. Model drivers rating scale**

Driver Rating	Rating Symbol	Numerical Value
Very Low	VL	1
Low	L	2
Nominal	N	3
High	H	4
Very High	VH	5
Extra High	XH	6

The weights ( $w_i$ ) for each model driver were obtained through a simple voting procedure. Since, as a first step, we planned to validate the utility of our approach in an academic setting we requested students enrolled in one of our Software Engineering II (SE II) class to vote on the importance of these eight model drivers in predicting understandability. Votes were cast on a scale of 1 to 5 with 1 being the least important and 5 being the most important. A total of 22 students participated in this voting process. The weights obtained thus are given in Table 4 below.

**Table 4. Model driver weights**

<b>i</b>	<b>MD<sub>i</sub></b>	<b>w<sub>i</sub></b>
1	PREC	3.64
2	RESL	3.23
3	CPLX	3.64
4	ACAP	3.55
5	PCAP	3.95
6	APEX	3.64
7	PLEX	3.32
8	LTEX	3.36

In general, the weights obtained above matched our expectations. There were, however, a few surprises. For instance, we had expected the product complexity (CPLX) to have the greatest influence on understandability. It seems that students, in general, give more weightage to programmer capability (PCAP) instead. Also, we had not expected architecture/risk resolution to get the lowest weight.

As far as the nature ( $n_i$ ) of a model driver is concerned, an intuitive analysis suffices. Apart from product complexity (CPLX), all of the remaining seven model drivers have a positive relationship with understandability. More capable analyst teams (i.e. higher values of ACAP), for instance, increase the chances of the development team understanding the requirements of the project well. Highly complex products (i.e. higher values of CPLX), on the other hand, decrease the chances of the development team understanding the requirements of the project well.

Once the weight ( $w_i$ ) and nature ( $n_i$ ) for each model driver ( $MD_i$ ) had been determined, the lowest and highest numerical value of understandability –  $UNDR_{Low}$  and  $UNDR_{High}$ , respectively – were determined using the formulae below.

$$\text{Equation 2: } UNDR_{Low} = \sum_{i=1}^8 n_i w_i \min(MD_i)$$

$$\text{Equation 3: } UNDR_{High} = \sum_{i=1}^8 n_i w_i \max(MD_i)$$

The  $\min()$  function is defined as

```

min (MDi) {
    if (ni = +1) return minimum numerical value of MDi
    else return maximum numerical value of MDi
}

```

whereas the  $\max()$  function is defined as

```

max (MDi) {
    if (ni = +1) return maximum numerical value of MDi
    else return minimum numerical value of MDi
}

```

As is clear from the definition of the  $\min()$  and  $\max()$  functions above, the output of these functions depends on the nature ( $n_i$ ) of the model driver ( $MD_i$ ). If the nature is such that the model driver has a positive relationship with understandability (i.e.  $n_i$  is +1) then the value returned by these functions matches intuition. Otherwise, if the relationship is negative (i.e.  $n_i$  is -1), then a value completely opposite to intuition is returned. This check on the nature of the model drivers was implanted to ensure that model drivers having a negative relationship with understandability (such as CPLX in the case of COCOMO II) do not give distorted results for extreme values of understandability.

Table 5 below shows the detailed procedure for calculating  $UNDR_{Low}$  and  $UNDR_{High}$ .

**Table 5. Detailed working for  $UNDR_{Low}$  and  $UNDR_{High}$**

i	MD <sub>i</sub>	n <sub>i</sub>	w <sub>i</sub>	min(MD <sub>i</sub> )	max(MD <sub>i</sub> )
1	PREC	+1	3.64	1	6
2	RESL	+1	3.23	1	6
3	CPLX	-1	3.64	6	1
4	ACAP	+1	3.55	1	5
5	PCAP	+1	3.95	1	5
6	APEX	+1	3.64	1	5
7	PLEX	+1	3.32	1	5

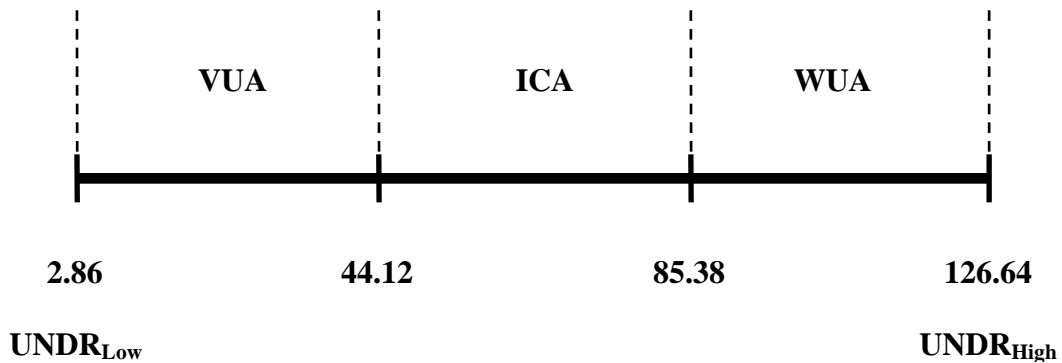
8	LTEX	+1	3.36	1	5
---	------	----	------	---	---

The values of  $\min(\text{MD}_i)$  and  $\max(\text{MD}_i)$  are obtained using the function definitions above along with the rating scale defined in Table 3. Solving equations 2 and 3 using values in the last four columns of Table 5 above gives the following results:

$$\text{UNDR}_{\text{Low}} = 2.86$$

$$\text{UNDR}_{\text{High}} = 126.64$$

The range between  $\text{UNDR}_{\text{Low}}$  and  $\text{UNDR}_{\text{High}}$  represents a spectrum of potential understandability values of real projects given the weights obtained in our academic setting. For convenient grouping of projects with respect to their understandability value, this range is divided into three equal portions. These portions are shown in Figure 1 below. The portion containing the smallest understandability values (2.86 – 44.12) represents the vaguely-understood applications (VUA) whereas the portion containing the highest understandability values (85.38 – 126.64) represents the well-understood applications (WUA). The middle portion represents intermediate clarity applications (ICA).



**Figure 1. Spectrum of understandability values with groups**

Here it must be explicitly pointed out that the ranges defining the three groups – VUA, ICA, and WUA – are completely adjustable. For one, the endpoints of the understandability spectrum heavily depend on the weights obtained for the various model drivers. For another, the size of the ranges doesn't have to be equal. Size of the individual understandability groups can be adjusted according to the nature of the projects being considered.

#### **4. Results**

Table 6 below summarizes the results of this empirical study. Each row in this table represents a single project. The serial numbers in the first column correspond to the serial numbers in Table 1 above. Each project's UNDR value and UNDR group are calculated using the methodology described in Section 3 above. The fourth column contains the projects' actual UNDR group. Values in this column were obtained by consulting the relevant success-critical stakeholders for each of these projects. These stakeholders were either clients of these projects or members of the instructional staff

(e.g. instructors, teaching assistants, etc.) actively involved in evaluating these projects from day one. The last column (“Matches reality?”) indicates whether the predicted UNDR group matches the actual UNDR group.

**Table 6. Actual and predicted UNDR values of projects**

S#	UNDR Value	Predicted UNDR Group	Actual UNDR Group	Matches reality?
1	64.05	ICA	ICA	Yes
2	52.23	ICA	ICA	Yes
3	73.41	ICA	ICA	Yes
4	80.82	ICA	ICA	Yes
5	49.59	ICA	VUA	No
6	76.59	ICA	ICA	Yes
7	70.09	ICA	ICA	Yes
8	59.50	ICA	ICA	Yes
9	67.09	ICA	ICA	Yes
10	63.45	ICA	ICA	Yes
11	62.95	ICA	ICA	Yes
12	73.64	ICA	ICA	Yes
13	59.59	ICA	ICA	Yes
14	31.77	VUA	VUA	Yes
15	69.68	ICA	ICA	Yes
16	84.59	WUA	WUA	Yes
17	24.50	VUA	VUA	Yes
18	76.91	ICA	ICA	Yes
19	62.82	ICA	VUA	No
20	77.18	ICA	ICA	Yes
21	56.18	ICA	ICA	Yes
22	84.18	ICA	WUA	No
23	73.41	ICA	ICA	Yes
24	66.32	ICA	WUA	No

A quick glance at the “Actual UNDR Group” column reveals that the majority of these projects lie in the ICA group. This was predicted with 100% accuracy by our weighted sum approach. In other words, all projects that were actually in the ICA group were predicted as such by our approach. Out of the four projects classified in the VUA group two were predicted accurately. The accuracy in predicting WUA applications, however, was not up to the mark. Only one out of a total of three projects in the WUA group was identified correctly. The overall prediction accuracy of our approach applied to this dataset is about 83%.

The exact reason for the four discrepancies between the actual UNDR group and the predicted UNDR group is hard to determine but, apparently, at least a couple of things could give rise to these discrepancies. Firstly, since the UNDR value (and hence the predicted UNDR group) of a project is highly dependent on the accurate assignment of values to the relevant COCOMO II model drivers, inappropriate setting of these model drivers by the students in a project team can produce an incorrect UNDR value. Secondly, this could be due to the distance between the body of students that worked on these projects and the one that cast their votes on the relative importance of the model drivers. Even a slight change in value propositions could lead to incorrect prediction of UNDR group as demonstrated by the projects with serial numbers 5 and 22. The UNDR value of project number 5 (49.59) is near the boundary line (44.12) separating the VUA and ICA groups whereas the UNDR value of project number 22 (84.18) lies almost on the boundary line (85.38) between the ICA and WUA groups.

## **5. Future Work**

In this empirical study we have developed and tested a framework for quantifying the extremely important yet abstract notion of the understandability of a software project. Our approach enables the prediction of understandability by reusing the inputs of the cost- and schedule-estimation process. While we have observed a strong relationship between our derived weighted-sum formula and the understandability of a software project our results are still preliminary and have limited applicability.

First and foremost, our results are heavily dependent on the weights obtained during the voting process. Given the fact that the voting took place in an academic setting our weighted sum formula may not apply in an industrial setting. One pending item in our research, therefore, is to consult the experts from the industry and to use techniques such as Wideband Delphi [Boehm 1981] to come up with a different set of model driver-weights for industry-grade projects. This, in turn, will require gathering and analyzing cost-estimation data of commercial projects for verification.

Another aspect of our approach that limits its applicability is the fact that our analysis has used the inputs of just one cost-estimation model – COCOMO II. Other widely-used models such as Putnam's SLIM [Putnam 1978] and RCA's PRICE-S [Freiman and Park 1979] use a different set of inputs. Thus, investigation of data obtained by using these other models may lead to further insights in quantifying and predicting understandability.

Also worth analyzing is the role played by understandability in the elaboration of high-level goals of a project to its corresponding low-level requirements. This phenomenon, known as requirements elaboration, was quantified and measured by Malik and Boehm in an empirical study [Malik and Boehm 2008]. What remains to be seen, however, is the extent of the contribution of various factors such as the novelty, complexity, and understandability of a project towards the degree of elaboration of its requirements.

## 6. References

- Boehm, B. (1981), *Software Engineering Economics*, Prentice-Hall.
- Boehm, B. (1996). “Anchoring the Software Process”, *IEEE Software* 13(4), pages 73–82.
- Boehm, B., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D., and Steece, B. (2000), *Software Cost Estimation with COCOMO II*, Prentice Hall.
- Boehm, B., Klappholz, D., Colbert, E., et al. (2005). “Guidelines for Lean Model-Based (System) Architecting and Software Engineering (LeanMBASE)”, Center for Software Engineering, University of Southern California.
- Boehm, B. and Turner, R. (2004), *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley.
- Freiman, F.R. and Park, R. E. (1979). “PRICE Software Model–Version 3: An Overview”, *Proc. IEEE-PINY Workshop on Quantitative Software Models*, pages 32–41.
- Kruchten, P. (2003), *The Rational Unified Process: An Introduction*, Addison-Wesley.
- Madachy, R. (1997). “Heuristic Risk Assessment Using Cost Factors”, *IEEE Software* 4 (3), pages 51-59.
- Malik, A. A. and Boehm, B. (2008). “An Empirical Study of Requirements Elaboration”, *The 22<sup>nd</sup> Brazilian Symposium on Software Engineering (SBES’08)*.
- Putnam, L. H. (1978). “A General Empirical Solution to the Macro Software Sizing and Estimating Problem”, *IEEE Trans. Software Engr.*, pages 345–361.
- SE I (2008). Links to websites of all past semesters of Software Engineering I (CSCI 577A) course at USC, [http://sunset.usc.edu/csse/courseroot/course\\_list.html#577a](http://sunset.usc.edu/csse/courseroot/course_list.html#577a)
- SE II (2008). Links to websites of all past semesters of Software Engineering II (CSCI 577B) course at USC, [http://sunset.usc.edu/csse/courseroot/course\\_list.html#577b](http://sunset.usc.edu/csse/courseroot/course_list.html#577b)
- Standish Group (1995). “CHAOS”, <http://www.standishgroup.com>
- Zowghi, D. and Nurmuliani, N. (2002). “A Study of the Impact of Requirements Volatility on Software Project Performance”, *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, pages 3-11.