Modeling Software Defect Introduction

Sunita Devnani-Chulani
USC - Center for Software Engineering
Los Angeles, CA 90089-0781
1-213-740-6470
sdevnani@sunset.usc.edu

ABSTRACT
In software estimation, it is important to recognize the strong relationships between Cost, Schedule and Quality. They form three sides of the same triangle. Beyond a certain point (the "Quality is Free" point), it is difficult to increase the quality without increasing either the cost or schedule or both for the software under development. Similarly, development schedule cannot be drastically compressed without hampering the quality of the software product and/or increasing the cost of development. Software estimation models can play an important role in facilitating the balance of the three factors.

This paper presents an initial version of the Defect Introduction sub-model of the empirical quality modeling extension to the existing COCOMO II software cost estimation model. The Quality model is an estimation model that can be used for predicting number of residual defects/KSLOC (thousands of Source Lines of Code) or defects/FP (Function Point) in a software product. It can be applied in the early activities such as analysis and design as well as in the later stages for refining the estimate when more information is available. It enables 'what-if' analyses that demonstrate the impact of various defect removal techniques and the effects of personnel, project, product and platform characteristics on software quality. It also provides insights on determining ship time, assessment of payoffs for quality investments and understanding of interactions amongst quality strategies.

The Quality model has two sub-models, namely the Defect Introduction model and the Defect Removal model. This paper focuses on the Initial version of the Defect Introduction model. It gives a detailed explanation of the rationale behind the suggested numeric ratings associated with the model parameters.

Keywords
Software Metrics, Software Quality Models, Software Defect Introduction, Software Defect Removal, Software Estimation

1 TOPICS ADDRESSED

The background model which forms the basis of the Cost/Quality model is described in Section 2. Section 3 presents A-Priori Defect Introduction model that is introduced in section 2 as a sub-model of the composite Cost/Quality model. The methodology used to develop the model is described in this section. Section 4 discusses the Delphi Process that was applied to arrive at the numerical values associated with the various parameters of the model. An example of the Delphi results using 'Analyst Capability' is presented. And, finally Section 5 concludes with the ongoing research of the Defect Removal model and future plans for calibrating and validating the model.

2 BACKGROUND MODEL

The Quality model is an extension to the existing COCOMO II [5, 19] model. It is based on 'The Software Defect Introduction and Removal Model' described by Barry Boehm in [3] which is analogous to the 'tank and pipe' model introduced by Capers Jones [15]. The next two subsections describe the COCOMO model and the framework of the Cost/Quality model. Readers familiar with the software cost estimation model, COCOMO, can skip section 2.1.

2.1 The COCOMO Model

The COCOMO II model was preceded by the original COCOMO '81 model published in [3]. The COCOMO '81 model has 3 levels of increasing detail and accuracy; Basic; Intermediate and Detailed. The top-level Basic COCOMO model is good for quick, early, rough-order of magnitude estimates of software costs. It models Effort as a non-linear function of Size as shown in Equation 1.

$$Effort = A \times \left[Size \right]^{B} \qquad \text{Eqn. 1}$$

The next level of detail is the Intermediate COCOMO model (Equation 2); which formulates Effort as a function of Size and a set of Effort Multipliers which account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other significant parameters. The third level of COCOMO '81 is the Detailed COCOMO which accounts for the effect of these parameters on the different phases.

$$Effort = A \times \left[Size \right]^{B} \times \prod_{i=1}^{15} EffortMultiplier_i \qquad \text{Eqn. 2}$$

The COCOMO II research effort started in 1994 and its initial definition and rationale are described in [5].

COCOMO II has a tailorable mix of three models, Applications Composition; Early Design; Post-Architecture, which target the realm of future software practices marketplace. The Cost/Quality model described in this paper is an extension to the Post-Architecture model which has been calibrated to a set of 83 datapoints [6] and is the most mature of the 3 submodels.

The Post-Architecture model, as the name suggests, is typically used after the software architecture is well defined and established. It estimates for the entire development life cycle of the software product and is a detailed extension of the Early-Design model. This model is the closest in structure and formulation to the Intermediate COCOMO '81 (Equation 2) and Ada COCOMO models.

The model uses a set of 17 multiplicative effort multipliers and a set of 5 exponential scale factors to adjust for project, platform, personnel, and product characteristics. The effort multipliers have a nominal rating of 1.0 assigned to them. Depending on the effect the effort multiplier has on effort the value is either greater than 1.0 (detrimental effect) or less than 1.0 (reduces development effort). The scale factors determine the economies/diseconomies of scale of the software under development replacing the development modes in the COCOMO '81 model and refining the exponent in the Ada COCOMO model. For further explanation and comparisons between the various COCOMO models the reader is urged to read [5]. A multiplicative constant, A, is used to calibrate the model locally for a better fit and it captures the linear effects of effort in projects of increasing size. The Post-Architecture model described above has the following form

$$Effort = A \times \left[Size \right]^{B} \times \prod_{i=1}^{17} EffortMultiplier_i \text{ where}$$

$$B = 1.01 + 0.01 \times \sum_{j=1}^{5} ScaleFactor_j \text{ and} \qquad \text{Eqn. 3}$$

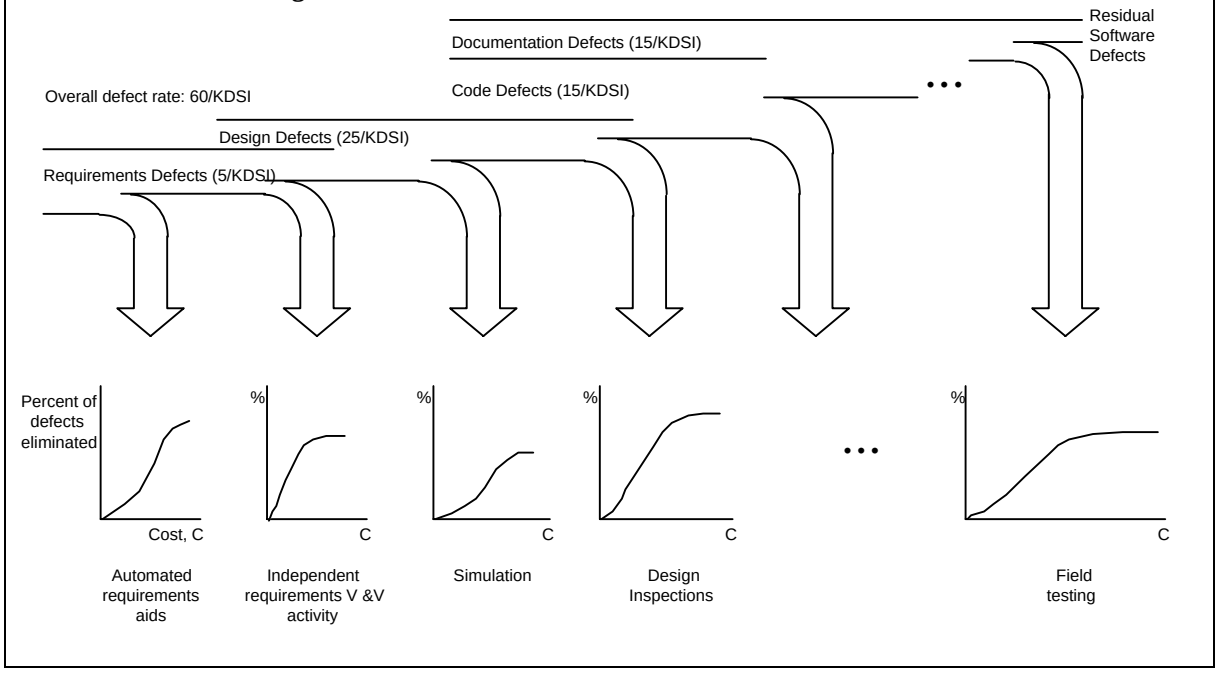
A = Multiplicative Constant
Size = Size of the software project measured in terms of KSLOC (thousands of Source Lines of Code [17], Function Points [20] or Object Points [1])

2.2 The Cost/Quality Model

The model depicted in Figure 1 shows that defects conceptually flow into a holding tank through various defect-source pipes & are drained off through various defect-elimination pipes. The defect source pipes are modeled as the "Software Defect Introduction Model". Figure 1 shows the overall Defect Introduction Rate (DIR) as 60 defects/KDSI of which 5/KDSI are Requirements defects, 25/KDSI are Design defects, 15/KDSI are Coding defects and 15/KDSI are Documentation defects. These DIRs are based on the research results reported in [4, 15, 18]. For the Defect Introduction model described in this paper these rates will be uses as baseline DIRs.

The defect elimination pipes are being modeled as the "Software Defect Removal Model". There are several removal techniques such as Inspections [9], Formal Methods [11], Cleanroom Development [8] etc. Each of the them has a characteristic production function which is a plot of the effort expended on the activity versus the benefit achieved in terms of percentage of defects removed. Ideally, selecting a particular level of investment for each of the defect-removal activities should determine the final quality of the software

Figure 1: Software Defect Introduction and Removal Model



project (Number of defects / Size). However, the process of combining the production functions is not easy. This is due to the high overlap between several defect-removal techniques. In order to develop a composite defect-removal production function, it is important to understand the relative overlap in removing defects belonging to the same class by different defect-removal activities. Several studies done in the late 1970s and the early 1980s [4, 15, 18] reported the percentage of errors removed by each error removal activity (such as artifact inspections, peer reviews, simulation, unit testing, coverage testing, formal methods etc.) but they did not report production functions for these activities.

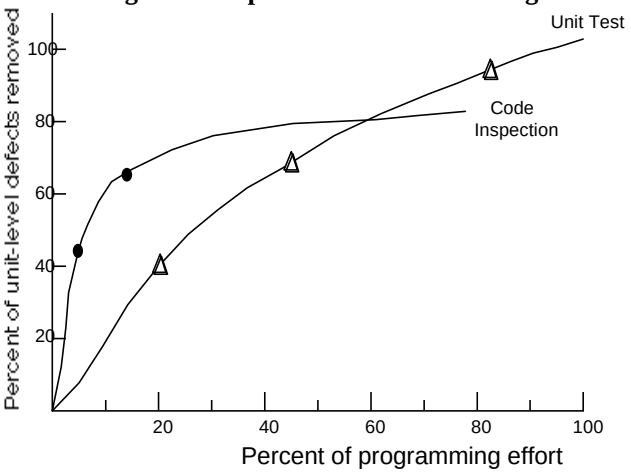
Table 1 shows the strengths and weaknesses of Inspection and Unit Testing. The two activities have a very strong overlap in removing the most common class of defects : simple programming blunders and logic defects.

Table 1: Strengths and Weaknesses of Inspection and Unit Testing

Technique	Inspections	Unit Test
Strengths	• Simple Programming logic defects • Developer blind spots • Interface defects • Missing portions • Specification defects	• Simple Programming logic defects • Numerical approximations • Program dynamics defects
Weaknesses	• Numerical approximations • Program dynamics defects	• Developer blind spots • Interface defects • Missing portions • Specification defects

Figure 2 shows that if 40% each of the programming effort is invested in Inspection and Unit Testing, then 135 % (60% by Unit testing and 75% by Inspection Testing) of the defects are removed which is obviously not possible. This clearly explains the high correlation between the two defect-removal activities and the difficulties in developing a composite production function.

Figure 2: Inspections and Unit Testing



The following two sections describe the Defect Introduction model; its rationale and formulation.

3 A-PRIORI DEFECT INTRODUCTION MODEL

As depicted in section 2, the Software Cost/Quality model is composed of the (i) Defect Introduction model and the (ii) Defect Removal model. The Defect Introduction model is the primary focus of this paper. The Defect Removal model is still in its early research phase and will be published in the near future.

3.1 Defect Introduction Model

Defects can be introduced in several activities of the software development life cycle and are classified based on their origin. The four types of defect artifacts based on this classification for the focus of this research are Requirements Defects (e.g. leaving out a required Cancel option in an Input screen), Design Defects (e.g. error in the algorithm), Coding Defects (e.g. looping 9 instead of 10 times) and Documentation Defects (e.g. incorrect instructions in User's manual to Cancel an operation). Capers Jones [16] in the glossary of his book under "Defect Origins" also has a category called "bad fixes". For the model discussed in this paper, "bad fixes" are accounted for by the varying ratings of certain DIR drivers (for example, Analyst Capability).

As explained in Section 2.2, the baseline Defect Introduction Rates ($DIR_{Baseline}$) for this model are:

Type of Artifact	**$DIR_{Baseline}$**
Requirements Defects	5/KDSI
Design Defects	25/KDSI
Coding Defects	15/KDSI
Documentation Defects	15/KDSI

Using the above baseline rates, the Nominal[1] Defect Introduction (DI_{Nom}) for each type of defect artifact, j, can be formulated as

$$DI_{Nom\,j} = DIR_{Baseline\,j} \; X \; (Size)^{B} \qquad \text{Eqn. 4}$$

And for now, **B** (which is equivalent to the Scale Factors in COCOMO II) is set to 1. Further investigation of this parameter is required but can be carried out when enough project data is available. It is unclear if Defect Introduction Rates will exhibit economies or diseconomies of scale as indicated in [2] and [10]. The question is if Size doubles, then will the Defect Introduction Rate increase by more than

[1] The "nominal" level of defects is without the effect of the 22 project-specific DIR drivers.

twice the original rate? This would indicate diseconomies of scale implying B > 1. Or will Defect Introduction Rate increase by a factor less than twice the original rate, indicating economies of scale, giving B < 1?

Equation 4 doesn't capture the effects of hardware constraints, personnel quality and experience, use of modern tools and techniques, and other significant parameters. It models Defect Introduction as a function of the baseline DIR and Size. This DIR is good for an order-of-magnitude estimate, but additional factors are necessary for better estimates of individual projects. Equation 4 is similar to equation 1 of the BASIC COCOMO '81 model. To increase the accuracy of the Defect Introduction model, the author used a set of 22 significant DIR drivers. The process incorporated to determine these DIR drivers and their impact on Defect Introduction is described in Section 4.

Equation 5 formulates the Estimated DI (DI_{Est}) as a function of the baseline DIR, Size and the 22 DIR-drivers. The DIR-drivers, discussed in further detail in Section 4, are aggregated as a product into a Quality Adjustment Factor (QAF). For each type of artifact, j,

$$DI_{Est\,j} = A_j \; X \; DI_{Nom\,j} \; X \; QAF_j \qquad \text{Eqn. 5}$$

where :
A_j = Calibration Constant for the jth artifact
QAF_j = Quality Adjustment Factor for each type of artifact (Requirements, Design, Coding, Documentation)

For each type of artifact, j,

$$QAF_j = \prod_{i=1}^{22} DIR\text{-}driver_{ij} \qquad \text{Eqn. 6}$$

Summarizing, we have

Requirements Defects Introduced ($DI_{Est\,;\,req}$)

$$= A_{req} \; X \; DI_{Nom;\,req} \; X \; QAF_{req}$$

Design Defects Introduced ($DI_{Est;\,des}$)

$$= A_{req} \; X \; DI_{Nom;\,des} \; X \; QAF_{des}$$

Coding Defects Introduced ($DI_{Est\,cod}$)

$$= A_{cod} \; X \; DI_{Nom;\,cod} \; X \; QAF_{cod}$$

Documentation Defects Introduced ($DI_{Est;\,doc}$)

$$= A_{doc} \; X \; DI_{Nom;\,doc} \; X \; QAF_{doc}$$

Total Defects Introduced

$$= \Sigma [A_j \; X \; (Size)^B \; X \; QAF_j] \qquad \text{Eqn. 7}$$

The model formulated in Equation 8 is analogous to the Intermediate COCOMO '81 model (Equation 2) and the COCOMO II Post Architecture model (Equation 3).

3.2 Example of the application of the Defect Introduction Model

The model described above can be illustrated using a simple example. Lets say, the Size of the software product is 4KSLOC. The nominal level of Defects Introduced for each type of artifact can then by computed as shown

$$DI_{Nom;\,req} = DIR_{Baseline;\,req} \; X \; (Size)^B = 5 \; X \; 4 = 20$$
$$DI_{Nom;\,des} = DIR_{Baseline;\,des} \; X \; (Size)^B = 25 \; X \; 4 = 100$$
$$DI_{Nom;\,cod} = DIR_{Baseline;\,cod} \; X \; (Size)^B = 15 \; X \; 4 = 60$$
$$DI_{Nom;\,doc} = DIR_{Baseline;\,doc} \; X \; (Size)^B = 15 \; X \; 4 = 60$$

Hence,

Nominal Defects Introduced (DI_{Nom}) = 240

Now, suppose the same product is developed by analysts rated at the 90th percentile and the other DIR drivers remain unchanged. From the description of the COCOMO II parameters [19], the Analyst Capability (ACAP) rating is set to Very High. The corresponding rating (discussed in Section 4) for $ACAP_{req}$=VH is 0.75, $ACAP_{des}$ = VH is 0.83, $ACAP_{cod}$ = VH is 0.90 and $ACAP_{doc}$ = VH is 0.83 resulting in QAF_{req} = 0.75, QAF_{des} = 0.83, QAF_{cod} = 0.90, QAF_{doc} = 0.83 with every other parameter set at Nominal. Lets say, calibration[2] results yield A_{req} = 1.25, A_{des} = 0.75, A_{cod} = 1.0, A_{doc} = 0.9 for the four multiplicative constants. The level of Defect Introduction will be as follows

$$DI_{Est;\,req} = 1.25 \; X \; 20 \; X \; 0.75 = 18$$
$$DI_{Est;\,des} = 0.75 \; X \; 100 \; X \; 0.83 = 62$$
$$DI_{Est;\,cod} = 1.0 \; X \; 60 \; X \; 0.90 = 54$$
$$DI_{Est;\,doc} = 0.9 \; X \; 60 \; X \; 0.83 = 45$$

Hence,

Total Defects Introduced = 179

The above example clearly shows the reduction in the number of defects introduced by having good analysts on the development team.

4 MODEL PARAMETERS

As introduced in Section 3, the Quality Adjustment Factor is a product of 22 DIR drivers. The modeling methodology used to determine the DIR drivers has been outlined below. A similar methodology has been used on the several versions of COCOMO and related models like the COTS Integration Cost Model (not yet published) and has been proven to be quite successful. The definition of the model (Steps 1-4) has been completed and validation using data (Step 5-6) is currently being done. Steps 2-4 are described in more detail in section 4.1

1) Analyze literature for factors affecting DIR

As an initial set of DIR drivers, the author took the 22 cost drivers from the COCOMO II Post Architecture model. The author found another factor 'Disciplined Methods' (DISC)

[2] The calibration process is still in progress and hence the values of A_{req}, A_{des}, A_{cod}, A_{doc} have been assumed.

to be quite significant as it captured effects of processes such as the PSP [13], the Cleanroom development approach [8], etc. The initial set of 23 DIR drivers is shown in table 2.

Table 2: Initial Set of DIR Drivers (FLEX was dropped from the set after the 2-Round Delphi)

Category	Post-Architecture Model
Platform	Required Software Reliability (RELY) Data Base Size (DATA) Required Reusability (RUSE) Documentation Match to Life-Cycle Needs (DOCU) Product Complexity (CPLX)
Product	Execution Time Constraint (TIME) Main Storage Constraint (STOR) Platform Volatility (PVOL)
Personnel	Analyst Capability (ACAP) Programmer Capability (PCAP) Applications Experience (AEXP) Platform Experience (PEXP) Language and Tool Experience (LTEX) Personnel Continuity (PCON)
Project	Use of Software Tools (TOOL) Multisite Development (SITE) Required Development Schedule (SCED) Disciplined Methods (DISC)
Scale Factors	Precedentedness (PREC) Development Flexibility (FLEX) Architecture/Risk Resolution (RESL) Team Cohesion (TEAM) Process Maturity (PMAT)

2) Perform behavioral analyses to determine the effect of factor levels on DIRs

A behavioral analysis for the 23 (22 from COCOMO II + DISC) DIR drivers was done. The effects of the factors on DIR by phase or activity was analyzed qualitatively. One of the COCOMO II factors, Development Flexibility, FLEX, was found to have an insignificant impact on DIR; although it was still included in the Delphi analyses.

3) Identify the relative significance of the DIR drivers on the DIRs

After a thorough study of the behavioral analyses was done, the relative significance of each DIR driver on the DIR was identified. Well-defined DIR driver levels and initial quantitative relationships between DIR driver levels and DIR was developed.

4) Perform expert-judgment Delphi assessment of quantitative relationships

A 2-Round Delphi process was performed by the author to assess the quantitative relationships (derived in Step 3), their potential range of variability, and to refine the factor level definitions. The driver, FLEX, was dropped based on the results of the Delphi showing its insignificance on DIRs. The Initial version of the Defect Introduction Model was then formulated using 22 DIR drivers.

5) Gather project data and determine statistical significance of the various DIR drivers

Using the refined DIR driver level definitions, gather project data on the drivers and resulting DIRs. Obtain the data-determined DIR driver levels and their statistical significance on DIR.

6) Determine a Bayesian A-Posteriori set of model parameters.

Using the expert-determined Delphi DIR drivers as a-priori values, determine a Bayesian a-posteriori set of model DIR drivers as a weighted average of the a-priori values and the data-determined values, with the weights determined by the statistical significance of the data-based results.

7) Gather more data to refine model

Continue to gather data, and refine the model to be increasingly data-determined v/s expert-determined.

4.1 Behavioral Analyses and Delphi Process

A thorough behavioral analyses for each DIR driver was done (Step 2 of Modeling Methodology). An example is provided in Table 4 (discussed in Section 4.2).

For the empirical formulation of the Defect Introduction Model, as with COCOMO II, it was essential to assign numerical values to each of the ratings of the DIR drivers. Based on expert-judgment an initial set of values was proposed for the model (Step 3 of Modeling Methodology). The DIR drivers range from VL (very low) to XH (extra high) and depending on their corresponding values either increase or decrease the level of defect introduction as compared to the nominal level of defect introduction. If the DIR driver > 1 then it has a detrimental effect on the DIR and overall software quality; and if the DIR driver < 1 then it reduces the DIR increasing the quality of the software being developed. This is analogous to the effect the COCOMO II Multiplicative Cost Drivers have on Effort (described briefly in Section 2.1).

A 2-round Delphi [12] was incorporated for further group consensus (Step 4 of Modeling Methodology). Readers not familiar with the above techniques can refer to Chapter 22 of [3] for an overview of common methods used for software estimation .

The Delphi process was formulated using the "Quality Range" for each DIR driver. The Quality Range is defined as the ratio between the largest DIR driver and the smallest DIR driver. The nine participants selected for the Delphi process were representatives of Commercial, Aerospace, Government and FFRDC and Consortia organizations. Each

of the participants had notable expertise in the area of Software Metrics and Quality Management and hence contributed significantly to the initial version of the model.

The steps that the author took for the entire Delphi process are outlined below.

Round 1 - Steps

1 Provided Participants with Round 1 Delphi Questionnaire with a proposed set of values for the Quality Ranges. This set was proposed based on experience.
2 Received nine completed Round 1 Delphi Questionnaires.
3 Ensured validity of responses by correspondence with the participants.
4 Did simple analysis based on ranges and medians of the responses.

Round 2 - Steps

1 Provided participants with Round 2 Delphi Questionnaire -- based on analysis of Round 1.
2 Repeated steps 2, 3, 4 (above)
3 Converged to Final Delphi Results which resulted in the definition of the initial model

4.2 An Example Defect Introduction Rate Driver

An example of how the Delphi process was carried out is shown using the Analyst Capability DIR driver. Analysts are personnel that work on requirements, high level design and detailed design. The major attributes that should be considered in this rating are Analysis and Design ability, efficiency and thoroughness, and the ability to communicate and cooperate effectively. Analysts that fall in the 15th percentile are rated very low and those that fall in the 90th percentile are rated very high as shown in Table 3.

Table 3: Analyst Capability Ratings

	Very Low	Low	Nominal	High	Very High
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile

Table 4 explains the impact ACAP has on the Defect Introduction Rate for the two extreme values i.e. VH and VL.

If ACAP = VH then the Number of Defects Introduced is lower as compared to ACAP = VL. Based on expert judgment a value of 0.75 is assigned to $ACAP_{req}$ = VH. This means that the Defect Introduction Rate of Requirements Defects is reduced by 75% if the rating of Analyst Capability is VH. Similar explanation can be given for the other ratings. Note again, that the Nominal rating is always 1.0.

As explained above, in Step 1 of Round 1, the author provided the nine participants with the Initial Quality Range. Hence, as shown in the above table, Initial Quality Range for the Requirements activity is 1.33/0.75 = 1.77. After about 3 weeks, the author received the responses from Round 1. After validating the responses with direct correspondence with some of the participants; the author did a simple analysis and derived the range and median of the responses. The results of this analysis, shown in the table as Range - Round1 and Median - Round 1, were provided to the participants along with their Round 1 response. This was the first step of Round 2. Again, after a turn around of about 2 weeks, the Round 2 results came in and a similar analysis (as in Round 1) was done. The median of the Round 2 responses resulted in the Final Quality Range. It was observed that the range in Round 2 was typically narrower

Table 4: Analyst Capability (ACAP) Differences in Defect Introduction

ACAP level	Requirements	Design	Code	Documentation
VH	Fewer Requirements understanding defects Fewer Requirements Completeness, consistency defects 0.75	Fewer Requirements traceability defects Fewer Design Completeness, consistency defects Fewer defects introduced in fixing defects 0.83	Fewer Coding defects due to requirements, design shortfalls -missing guidelines -ambiguities 0.90	Fewer Documentation defects due to requirements, design shortfalls 0.83
Nominal	Nominal level of defect introduction 1.0			
VL	More Requirements understanding defects More Requirements Completeness, consistency defects 1.33	More Requirements traceability defects More Design Completeness, consistency defects More defects introduced in fixing defects 1.20	More Coding defects due to requirements, design shortfalls -missing guidelines -ambiguities 1.11	More Documentation defects due to requirements, design shortfalls 1.20
Initial Quality Range	1.77	1.45	1.23	1.45
Range - Round 1	1.4-2	1.3-1.8	1-1.4	1.15-1.5
Median - Round 1	1.92	1.48	1.23	1.23
Range - Round 2	1.7-2	1.4-1.69	1.2-1.41	1.25-1.45
Final Quality Range	1.77	1.45	1.23	1.45

than the range in Round 1.

New DIR drivers for the several intermediate levels were computed using the Final Quality Range. For example, the Final Quality Range for Requirements Defects due to ACAP is 1.77. The Very Low and Very High ratings associated with ACAP for Requirements Defects were computed using geometric interpolation as shown below

$$DIR - Driver(VeryLow) = \sqrt{FinalQualityRange} = \sqrt{177} = 1.33$$

and

$$DIR\text{-}Driver(VeryHigh) = (\sqrt{FinalQualityRange})^{-1} = (\sqrt{177})^{-1} = 0.75$$

Eqn. 8

Example : Consider a software project which has 100 Requirements Defects, and has analysts of nominal level of capability working on it. Suppose, now the team of analysts is replaced by very-high rated analysts. Assume also that all other parameters remain unchanged.

Table 5: Analyst Capability (ACAP) DIR Driver

ACAP level / Type of Artifact	***Requirements***	Design	Coding	Documentation
Very High	***0.75***	0.83	0.90	0.83
High	***0.87***	0.91	0.95	0.91
Nominal	***1.0***	1.0	1.0	1.0
Low	***1.15***	1.10	1.05	1.10
Very Low	***1.33***	1.20	1.11	1.20

Using table 4, the number of Requirements Defects will be reduced to 75 (100 X 0.75). The reasons for the decrease in the number of defects introduced in the Requirements activity are presented in table 4. Thus the 25% decrease in Requirements Defects is due to the very-high level of capability of the analysts causing fewer Requirements understanding and Requirements Completeness, Consistency defects as compared to nominal level. At this point, the reader is urged to revisit the example discussed in section 3.

The values assigned to ACAP for each type of artifact are graphically represented in Figure 3 and summarized in Table 5. From the figure, it can be seen that ACAP = VH results in $ACAP_{req}$ = 0.75. Similarly, the other ratings are also represented graphically.

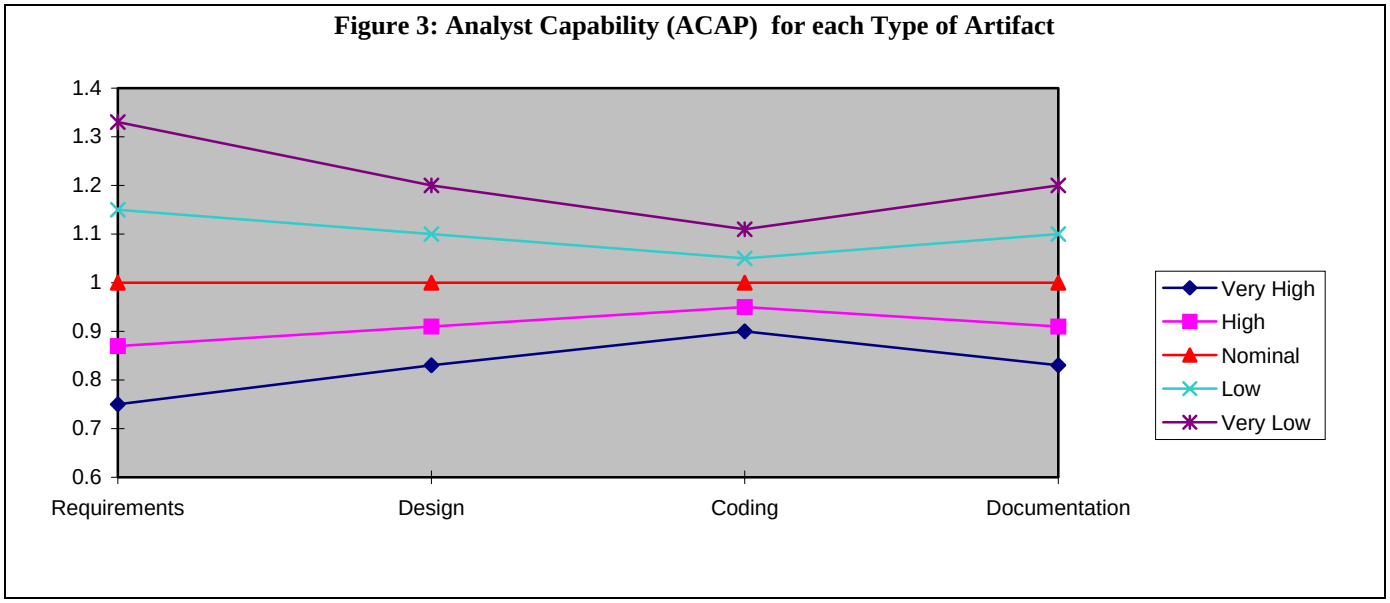
A detailed description of each of the other 21 DIR drivers and its impact on defect introduction for each type of defect artifact can be found in [7].

4.3 Software Quality Range

'Quality Range', as defined in Section 4.1, is the ratio between the largest DIR driver and the DIR driver. It is an indicator of the relative ability of the attribute to affect software quality.

Figure 4 (on the next page) provides a graphical view of the relative Quality Ranges for Introduction of Requirements Defects provided by the 22 Defect Driver Attributes. It shows each factor's Quality Range. If all other Defect Driver Attributes are held constant, a Very Low (VL) rating for Disciplined Methods (DISC)will result in a software project with 2.5 times the number of residual Requirements Defects as compared to a Very High (VH) rating.

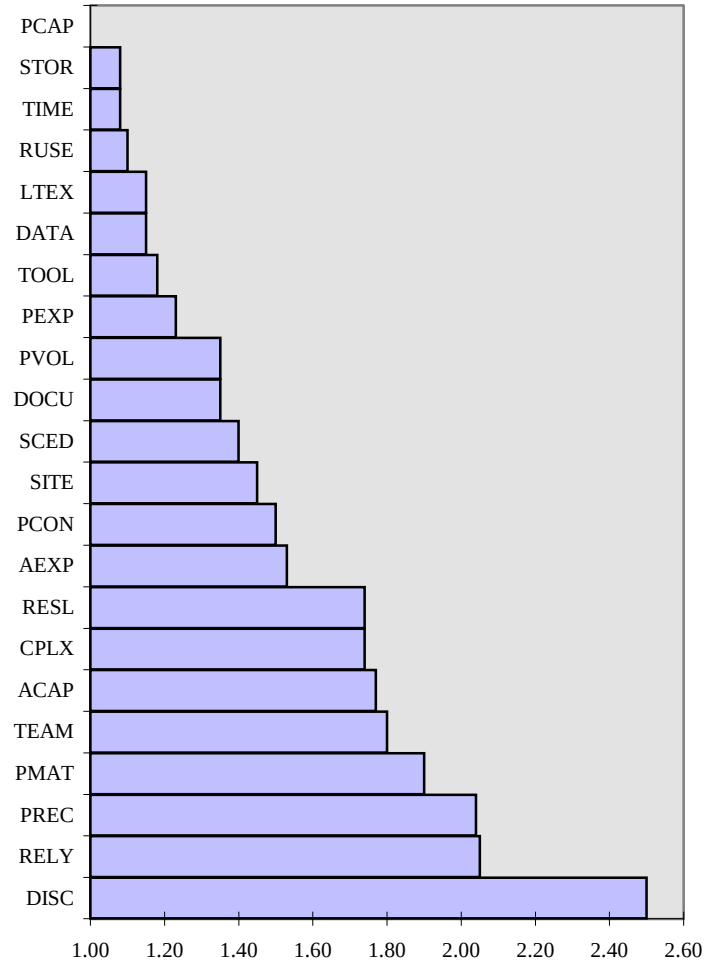
Figure 3: Analyst Capability (ACAP) for each Type of Artifact



5 CONCLUSIONS AND ONGOING RESEARCH

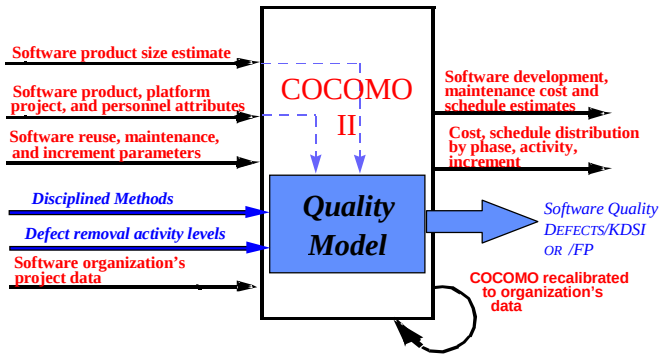
In Section 2, the overall Cost/Quality model was described. This paper described the Initial Defect Introduction sub-model. The Defect Removal sub-model that was introduced in Section 2.2 being developed. Once both the submodels are well-defined they will compose the complete Software Cost/Quality model. This model will be integrated to the existing cost estimation COCOMO II model (Fig. 5). The inputs and outputs shown in italics will be in addition to the original COCOMO II inputs and outputs.

Figure 4: Requirements Quality Ranges



The Cost/Quality model can be used to predict the cost/schedule/quality of the software under development. Sensitivity analyses to understand the interactions between these parameters can be done. The author is targeting to have the complete A-Priori model description by March 1997 and the data-determined A-Posteriori model available by early '98.

Figure 5: COCOMO II with Quality Model Extension



6 REFERENCES

1. Banker R. D., Kauffman R. J., Kumar R., "An Empirical Test of Object-based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment", Journal of Management Information Systems/Winter 1991-92, Vol. 8, No. 3, pp. 127-150.

2. Banker R. D., Chang H., Kemerer C., "Evidence on Economies of Scale in Software Development", Information and Software Technology, 1994, pp. 275-282

3. Boehm B. W., Software Engineering Economics, 1981, Prentice-Hall

4. Boehm B. W., "Developing Small Scale Application Software Products: Some Experimental Results", Proceedings, IFIP 8th World Computer Congress, 1980.

5. Boehm, B. W., Clark B., Horowitz E., Westland C., Madachy R., Selby R., "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", Annals of Software Engineering Special Volume on Software Process and Product Measurement, 1995, J.D. Arthur and S.M. Henry, Eds., J.C. Baltzer AG, Science Publishers, Amsterdam, The Netherlands, Vol 1, pp. 45 - 60

6. Boehm B., Clark B, Devnani-Chulani S., "Calibration Results of COCOMOII.1997", Technical Report, USC-CSE-97-507, 1997, Computer Science Department, University of Southern California, Center for Software Engineering, Los Angeles, CA 90089-0781

7. Devnani-Chulani S., "Results of Delphi for the Defect Introduction Model - Sub-Model of the Cost/Quality Model Extension to COCOMO II" , Technical Report, USC-CSE-97-504, 1997, Computer Science Department, University of Southern California, Center for Software Engineering, Los Angeles, CA 90089-0781

8. Dyer M., The Cleanroom Approach to Quality Software Development, 1992, Wiley Series in Software Engineering Practice

9. Fagan M.E., “Design and Code Inspections to Reduce Errors in Program Development”, IBM Systems Journal, 1976, pp. 182-211

10. Gulledge T. R., and Hutzler W. P., Analytical Methods in Software Engineering Economics, 1993, Springer-Verlag

11. Hatton L., Pfleeger S. L., “Investigating the Influence of Formal Methods”, February 1997, IEEE Computer

12. Helmer O., Social Technology, Basic Books, New York, 1966

13. Humphrey W. S., Discipline for Software Engineering, 1995, SEI Series in Software Engineering

14. Jones C., “Programming Defect Removal”, Proceedings, GUIDE 40, 1975

15. Jones C., “Measuring Programming Quality and Productivity”, IBM Systems J., 17, 1, 1978, p. 39-63

16. Jones C., Assessment and Control of Software Risks, 1994, Yourdon Press Computing Series

17. Park, “Software Size Measurement: A Framework for Counting Source Statements”, CMU-SEI-92-TR-20, 1992, Software Engineering Institute, Pittsburg, PA

18. Thayer T., Lipow M., Nelson E., Software Reliability, North Holland, 1978

19. “COCOMO II Model Definition Manual”, Computer Science Department, University of Southern California, Center for Software Engineering, Los Angeles, CA 90089-0781

20. “Function Point Counting Practices Manual”, International Function Point Users Group (IFPUG), Release 4.0, 1994

ACKNOWLEDGMENTS

This research is sponsored by the Affiliates of the USC Center for Software Engineering: Aerospace Corporation, Air Force Cost Analysis Agency, AT&T, Bellcore, DISA, Electronic Data Systems Corporation, E-Systems, Hughes Aircraft Company, Interactive Development Environments, Institute for Defense Analysis, Jet Propulsion Laboratory, Litton Data Systems, Lockheed Martin Corporation, Loral Federal Systems, Motorola Inc., Northrop Grumman Corporation, Rational Software Corporation, Rockwell International, Science Applications International Corporation, Software Engineering Institute (CMU), Software Productivity Consortium, Sun Microsystems, Inc., Texas Instruments, TRW, U.S. Air Force Rome Laboratory, U.S. Army Research Laboratory, and Xerox Corporation.

Special thanks to my advisor, Prof. Barry Boehm, to whom I am particularly indebted for helpful comments and complete support.