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

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.

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

\[
Effort = A \times \left[\text{Size}\right]^B \times \prod_{i=1}^{15} \text{EffortMultiplier}_i
\]  
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

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

$$B = 1.01 + 0.01 \times \sum_{j=1}^{5} \text{ScaleFactor}_j$$

and

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 used 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
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

<table>
<thead>
<tr>
<th>Technique</th>
<th>Inspections</th>
<th>Unit Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strengths</td>
<td>• Simple Programming logic defects</td>
<td>• Simple Programming logic defects</td>
</tr>
<tr>
<td></td>
<td>• Developer blind spots</td>
<td>• Numerical approximations</td>
</tr>
<tr>
<td></td>
<td>• Interface defects</td>
<td>• Program dynamics defects</td>
</tr>
<tr>
<td></td>
<td>• Missing portions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Specification defects</td>
<td></td>
</tr>
<tr>
<td>Weaknesses</td>
<td>• Numerical approximations</td>
<td>• Developer blind spots</td>
</tr>
<tr>
<td></td>
<td>• Program dynamics defects</td>
<td>• Interface defects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Missing portions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Specification defects</td>
</tr>
</tbody>
</table>

As explained in Section 2.2, the baseline Defect Introduction Rates (DIRBaseline) for this model are:

<table>
<thead>
<tr>
<th>Type of Artifact</th>
<th>DIRBaseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Defects</td>
<td>5/KDSI</td>
</tr>
<tr>
<td>Design Defects</td>
<td>25/KDSI</td>
</tr>
<tr>
<td>Coding Defects</td>
<td>15/KDSI</td>
</tr>
<tr>
<td>Documentation Defects</td>
<td>15/KDSI</td>
</tr>
</tbody>
</table>

Using the above baseline rates, the Nominal Defect Introduction (DINom) for each type of defect artifact, j, can be formulated as

\[ DI_{Nom_j} = DIR_{Baseline_j} \times (Size)^B \]

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, \( j \),

\[
DI_{Est} = A_j \cdot DI_{Nom, j} \cdot QAF_j
\]

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

For each type of artifact, \( j \),

\[
QAF_j = \prod_{i=1}^{22} DIR\_driver_{ij}
\]

Summarizing, we have

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

\[
= A_{req} \cdot DI_{Nom, req} \cdot QAF_{req}
\]

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

\[
= A_{req} \cdot DI_{Nom, des} \cdot QAF_{des}
\]

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

\[
= A_{cod} \cdot DI_{Nom, cod} \cdot QAF_{cod}
\]

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

\[
= A_{doc} \cdot DI_{Nom, doc} \cdot QAF_{doc}
\]

Total Defects Introduced

\[
= \sum \{ A_j \cdot (Size)^B \cdot QAF_j \}
\]

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. Let’s say, the Size of the software product is 4K SLOC. The nominal level of Defects Introduced for each type of artifact can then be computed as shown

\[
\begin{align*}
DI_{Nom, req} &= DIR_{Baseline, req} \cdot (Size)^B = 5 \cdot 4 = 20 \\
DI_{Nom, des} &= DIR_{Baseline, des} \cdot (Size)^B = 25 \cdot 4 = 100 \\
DI_{Nom, cod} &= DIR_{Baseline, cod} \cdot (Size)^B = 15 \cdot 4 = 60 \\
DI_{Nom, doc} &= DIR_{Baseline, doc} \cdot (Size)^B = 15 \cdot 4 = 60
\end{align*}
\]

Hence,

**Nominal Defects Introduced (\( DI_{Nom} \)) = 240**

Now, suppose the same product is developed by analysts rated at the 90\(^{th}\) 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. Let’s 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

\[
\begin{align*}
DI_{Est, req} &= 1.25 \cdot 20 \cdot 0.75 = 18 \\
DI_{Est, des} &= 0.75 \cdot 100 \cdot 0.83 = 62 \\
DI_{Est, cod} &= 1.0 \cdot 60 \cdot 0.90 = 54 \\
DI_{Est, doc} &= 0.9 \cdot 60 \cdot 0.83 = 45
\end{align*}
\]

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)

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

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 vs 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>
<thead>
<tr>
<th>ACAP level</th>
<th>Requirements</th>
<th>Design</th>
<th>Code</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VH</td>
<td>Fewer Requirements understanding defects Fewer Requirements traceability defects Fewer Design Completeness, consistency defects 0.75</td>
<td>Fewer defects introduced in fixing defects 0.83</td>
<td>Fewer Coding defects due to requirements, design shortfalls -missing guidelines -ambiguities 0.90</td>
<td>Fewer Documentation defects due to requirements, design shortfalls 0.83</td>
</tr>
<tr>
<td>Nominal</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VL</td>
<td>More Requirements understanding defects More Requirements traceability defects More Design Completeness, consistency defects More defects introduced in fixing defects 1.33</td>
<td>1.20</td>
<td>More Coding defects due to requirements, design shortfalls -missing guidelines -ambiguities 1.11</td>
<td>More Documentation defects due to requirements, design shortfalls 1.20</td>
</tr>
</tbody>
</table>

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<sub>req</sub> = 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.
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

\[
\text{DIR - Driver(VeryLow)} = \sqrt[4]{\text{FinalQualityRange}} = \sqrt[4]{1.77} = 1.33
\]

and

\[
\text{DIR-Drive(VeryHigh)} = (\sqrt[4]{\text{FinalQualityRange}})^4 = (\sqrt[4]{1.77})^4 = 0.75
\]

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.

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 ACAPreq = 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](image-url)
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.

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.

6 REFERENCES


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


ACKNOWLEDGMENTS


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