SOFTWARE DEVELOPMENT
COST ESTIMATION USING COCOMO II
BASED META MODEL

Alexander Hjalmarsson

Degree project in
ICS
Master thesis
Stockholm, Sweden 2013

XR-EE-ICS 2013:005
Acknowledgements

I would like to thank everyone who have contributed, opposed to, assisted or helped me in any way during this thesis. I would especially like to thank my supervisor at KTH, Matus Korman, for his valuable support and guidance throughout the work.

I would also like to thank Dr. Robert Lagerström for his ideas, suggestions and discussions during the study.

At last, special thanks goes to the case study company and my supervisor there who helped me throughout the entire thesis and have given me valuable contacts within the company.
Abstract
Large amounts of software are running on what is considered to be legacy platforms. These systems are often business critical and cannot be phased out without a proper replacement. The generations of developers that have developed, maintained and supported these systems are leaving the workforce leaving an estimated shortfall of developers in the near time. Migration of these legacy applications can be troublesome due poor documentation and estimating the sizes of the projects is nontrivial. Expert estimates are the most common method of estimation when it comes to software projects but the method is heavily relying on the experience, knowledge and intuition of the estimator. The use of a complementary estimation method can increase the accuracy of the estimation. This thesis constructs a meta model that combines enterprise architecture concepts with the COCOMO II estimation model in order to utilize the benefits of architectural overviews and tested models with the purpose of supporting the migration process. The study proposes a method combining expert cost estimation with model based estimation which increases the estimation accuracy. The combination method on the four project samples resulted in a mean magnitude of relative error of 10%.

Keywords: Software Development Cost Estimation, COCOMO II, Enterprise Architecture, P'AMF, ArchiMate, Software Migration Cost Estimation.
# Table of Contents

- List of figures: v
- List of tables: vi
- Abbreviations: vii

1. INTRODUCTION: 1
2. OBJECTIVE AND DELIMITATIONS: 2
    2.1 Purpose: 2
    2.2 Delimitations: 2
3. METHODOLOGY: 3
    3.1 Case study: 3
    3.2 Research process: 6
4. THEORY: 10
    4.1 Algorithmic cost modeling: 10
    4.2 Expert judgment: 13
    4.3 COCOMO II: 13
    4.4 Enterprise Architecture: 20
    4.5 Combination of model and expert judgment: 22
5. META MODEL IMPLEMENTATION: 24
    5.1 Modeling language: 24
    5.2 Concepts: 25
    5.3 Software Development Change Estimation view: 28
    5.4 Application sizing: 29
    5.5 Element sets: 30
    5.6 Change project cases: 31
6. DATA: 33
    6.1 Project A: 34
    6.2 Project B: 36
    6.3 Project C: 37
    6.4 Project D: 39
7. ANALYSIS: 41
    7.1 Input from project members: 41
    7.2 Calibration: 42
    7.3 Combination of model and expert judgment: 42
8. RESULTS: 43
    8.1 Validation: 43
    8.2 Accuracy: 44
9. DISCUSSION: 47
    9.1 Future research: 49
10. CONCLUSION: 51
11. REFERENCES: 52

Appendix A – KPA Questionnaire: 56
Appendix B – COCOMO II elicitation survey: 61
Appendix C – Scale factors and effort multipliers: 70
List of figures

Figure 1: Thesis outline........................................................................................................6
Figure 2: Meta model of the core concepts of ArchiMate (The Open Group, 2009)..........................................................................................................................21
Figure 3: Connection between Software Development Change Meta model, ArchiMate meta model and the view.................................................................25
Figure 4: Software Development Change meta model......................................................25
Figure 5: Software Development Change View with relationships.........................28
Figure 6: Slightly simplified version of project A’s model...........................................36
Figure 7: Calculation differences in COCOMO II platform factors and calculation model..................................................................................................................43
Figure 8: Sources of errors in case study.................................................................48
Figure 9: Integrated modeling error sources............................................................49
List of tables

Table 1: Degree of influence measures and their ratings. ........................................ 11
Table 2: Processing complexity characteristics..................................................... 12
Table 3: KPA Questionnaire weighting table. ...................................................... 15
Table 4: Process maturity rating selection table............................................... 16
Table 5: Example of effort divisor usage. ............................................................. 27
Table 6: Application function to service attribute mappings. ................................ 31
Table 7: Project A COCOMO II results. ............................................................ 35
Table 8: Project B COCOMO II results. ............................................................ 37
Table 9: Project C COCOMO II results. ............................................................. 39
Table 10: Project D COCOMO II results. ........................................................... 40
Table 11: Case study (M)MRE results with different methods without before calibration. .......................................................... 44
Table 12: The accuracy of the schedule estimation (TDEV). .............................. 45
Table 13: Case study (M)MRE results with different methods after calibration. .......................................................... 45
Table 14: The (M)MRE of the schedule estimation after calibration................. 46
Abbreviations

- CMM  Capability Maturity Model
- COCOMO  Constructive Cost Model
- COTS  Commercial off-the-shelf
- EAAT  Enterprise Architecture Analysis Tool
- EM  Effort Multiplier
- MMRE  Mean Magnitude of the Relevant Error
- MRE  Magnitude of the Relevant Error
- OCL  Object Constraint Language
- P²AMF  Predictive, Probabilistic, Architecture Modeling Framework
- SF  Scale Factor
1 INTRODUCTION

When having a software product portfolio spanning over hundreds of legacy systems, maintenance becomes a problem. Expensive hardware as well as lack of experienced developers in the environments drives the cost of maintenance each year. These legacy systems are often crucial to the businesses and cannot be phased out without a proper replacement (Bennet, 1995).

Even though new computing technologies have emerged on the market, a considerable amount of software still runs on legacy systems. It is estimated that around 200 billion lines of Cobol code are running in live operation and that 75% of the world’s business data are processed in Cobol (Datamonitor, 2008) (Barnett, 2005). Along with an estimated shortfall in Cobol developers in the 2015-2020 timeframe as the older generation leaves the workforce, it is imminent that migration from the legacy mainframes becomes a priority for many organizations (Barnett, 2005).

There are many difficulties involved in the migration process. Understanding the design and functionality the legacy systems may be troublesome due to the fact that many of these systems have poor, if any, documentation. Because of this, interaction from a system expert is often required (Bisbal et al., 1997). These experts need to analyze the old systems to create accurate requirement specifications regarding technical functionality to provide documentation for the developers and architects involved in the migration process.

Because of the importance of these systems the replacement often needs to suit both new business objectives while maintaining functionality for legacy systems that not yet have been migrated. These factors all come into play when estimating the cost and time of a migration software project. A case study made by Kitchenham et al. in 2002 showed that as much as 72% of the 145 maintenance projects in the survey used expert opinion as the method of estimating software development costs (Kitchenham et al., 2002). Another survey in 1997 showed that out of 26 industrial projects almost 81% were based on expert estimates (Molokken & Jørgensen, 2003).

One of the problems with expert estimates is that they can be strongly biased and misled by irrelevant information. This can lead to over-optimism and inaccurate estimations. This causes project over-runs and can be avoided with an unbiased estimation model (Kitchenham et al., 2002).

This thesis will propose a meta model based of the ArchiMate modeling language (The Open Group, 2009) combined with the Constructive Cost Model II (COCOMO II).
2 OBJECTIVE AND DELIMITATIONS

Large software development projects have a reputation of being hard to estimate resources for and information system managers have confirmed that cost estimation is an important concern (Lederer & Prasad, 1995). The objective of this thesis is to create a meta model for determining the time and cost for system migration and phase out projects in large IT portfolios that can be used as a complement to expert judgments. The purpose of the thesis is not to validate COCOMO II itself, as it already is a validated method for estimations, but rather create and validate the implementation of the meta model implementation. The scope of the thesis includes a validation of the implemented calculation model as well as a case study at a manufacturing company in Sweden.

2.1 Purpose

The purpose is to analyze the possibilities of including software change projects from an architectural view based on the ArchiMate modeling language together with COCOMO II. The aim is define the strength and weaknesses of the meta model approach as well as measuring the estimation accuracy regarding schedule time and cost of the model. As the purpose of the model is to serve as a complement to expert judgments a proposal of how to combine the two methods is also made.

The purpose can be clarified into the following questions:

- How can an estimation model be combined with meta modeling techniques to facilitate software development estimation processes?
- How can a software development estimation model complement existing expert elicitation methods?

2.2 Delimitations

Several delimitations have been made in order to reduce the scope thesis.

The perhaps most notable delimitation is that COCOMO II and ArchiMate is used as a baseline throughout the entire meta model design. The rationale behind the decision is that they both are fairly simple yet enable enough flexibility in the design.

COCOMO II suggests calibration to the specific organization in order to achieve the most accurate results. It supports two types of calibrations where the most comprehensive calibration has not been made because of the large effort needed to gather project data.

As the goal is to develop a meta model that enables software change estimation for development project, no research have been made to support commercial off-the-shelf (COTS) integration projects.
3 METHODOLOGY

This chapter describes the choice of methodology used in this thesis to gather, analyze and validate the required data.

The general methodology used in the thesis has been a case study at a manufacturing company in the automotive industry in Sweden. Sample projects have been gathered and project stakeholders have been interviewed for necessary information to both facilitate the meta model design process as well as providing data from the project samples.

3.1 Case study

Questions that involve “how many” or “how much” are according to (Yin, 2009) likely to favor survey methodology over case study methodology. The COCOMO II estimation model relies on many measurable metrics questionnaire used is fully structured. Thus, the survey methodology can be seen as a natural fit. Because some of the questions and statements in the questionnaire are rather diffuse it is important to verify that the respondents have the same understanding of the questions in order for them to be comparable, therefore the questions requires a rationale behind the decision. According to (Yin, 2009) case study methodology is useful when the researcher is searching for answers of the type “how” and “why”. With the COCOMO II elicitation these questions are generally not needed but as the purpose of this thesis is to develop models of the architecture in the systems as well, the survey format needs to be complemented. The survey and case study methodologies are not mutually exclusive meaning that for example the survey methodology can be used inside of a case study (Yin, 2009) and these two types of methodologies are used within this research.

Robson classifies research methodologies into four types of purposes: Exploratory, Descriptive, Explanatory and Improving. These types can be combined and are thus not mutually exclusive (Robson, 2002).

Exploratory case studies have the objective of finding out what is happening and to generate insights and hypothesis for new research. Explanatory studies seek explanations of situations or problems, often through casual relationships (Runeson & Höst, 2009).

According to Robson (2002) a plan for a case study should at least contain these following elements:

- Objective
- Description of the case
- Theory
- Research questions
- Methods
- Sample selection strategy
These six elements for the case study plan are described in the following subsections.

### 3.1.1 Objective
The objective of the study can be explanatory, descriptive, explanatory or improving (Runeson & Höst, 2009).

In the context of this research the objective is a combination of explanatory and exploratory. The exploratory part of the case study is to construct a new meta model and generate ideas for future research within the area. The explanatory part is conducted by creating models that describe the architecture and its relationships along with the calculation model.

### 3.1.2 Description of the case
The case study company is a large manufacturer in the automotive industry. The case itself is based on the previous record of software estimation in the company where the previous use of expert judgment needs to be complemented with another method to provide greater accuracy in the effort prediction process.

The case company has utilized a form of expert judgment to conduct effort estimations in the software development projects. The results of these estimations are very dependent on the knowledge and experiences of the estimator thus the accuracy of the estimations vary greatly. Because of this they have requested a method that can serve as a compliment of their existing method.

### 3.1.3 Theory
Gregor (2006) classifies four groups of theories. These groups are

- Analysis
- Explanation
- Prediction
- Explanation and prediction
- Design and action

The theoretical framework used in the case study consists of the COCOMO II model (Boehm et al., 2000) and the function point analysis theories (Albrecht & Gaffney Jr, 1983). The prediction group is a type of theory that predict without providing explanations (Hannay et al., 2007). Both of these theories (function point analysis & COCOMO II) are located within the prediction group. These include mathematical and probabilistic models as well as predictive models in software engineering. Further, it is especially stated that COCOMO II is a prediction theory (Hannay et al., 2007).

### 3.1.4 Research questions
The case research question is to validate the accuracy of the meta model implementation.

How does it compare to the company’s current methods?
How accurate is the model and how can it serve as a complement to expert judgments?

3.1.5 Methods
When collecting data it is important to use several sources of data in the case study in order to limit the effects of the interpretations of the sources to ensure high validity of the data (Runeson & Höst, 2009). Lethbridge et al. (2005) divides data collection techniques regarding software engineering into three different levels:

- **First degree**
  Direct contact with the subjects and data collection occurs in real time. This includes for example interviews, questionnaires, focus groups and observations.

- **Second degree**
  Indirect method where the researchers have access to the engineer’s environment. Once the data collection has been initiated the subjects can work normally while data is automatically collected. The technique is passive and do not require any direct contact with the subjects.

- **Third degree**
  This method works by looking at the engineer’s output and by-products. Outputs can for instance be source code, documentation and reports while the by-products are created in the working process and includes log files, change logs and output from build or configuration management tools.

As the collection method of COCOMO II has to be strict in order to get reproducible results, the first degree level of data collection is used in the case study. A survey based of the attributes and their descriptions in COCOMO II were used in the interviews. Interviews are made with several project members in several projects in order to get a reliable calibration of the model as well as a reliable validation.

3.1.6 Sample selection strategy
The case study consists of several software development projects where different types of migration projects have been selected. The projects studied must be closed in order to get the final cost of the project in both calendar time and person hours spent. A project is considered closed once the managers in the company have approved the final report written by the project manager where the total costs of the projects are included.

Projects that are not general software development projects such as commercial off-the-shelf integration projects have not been included in the study as they do not conform well to the COCOMO II model.

There are several strategies for choosing samples and case study companies. Flyvbjerg defines two different categories of selections: Random selection and Information-oriented selection. The first category is further defined into two subtypes, namely random sample and stratified sample. The latter category is further
defined into four different strategies for selection: Extreme/deviant, maximum variation cases, critical cases and paradigmatic cases (Flyvbjerg, 2006).

The particular case company was chosen due their interest in the problem area but the selection of the samples (i.e. projects for validation and calibration purposes) has to be selected on some premise. The selection method for the project samples have been based on stratified samples, as there are both practical constraints on the selections as well as limitations in the COCOMO model that have only been initially calibrated for projects larger than 2000 source lines of code (Boehm et al., 2000, p. 400). The practical constraint is the amount and type of information needed from the project members as they are responsible for providing data regarding the projects. Thus, the projects members need to be available at the company and the project has to have been finished recently in order for them to have fresh enough memories regarding the project.

### 3.2 Research process

The implementation of the meta model have been validated against these projects in order to measure the accuracy of the model as well as making sure that it calculates correct comparing with the COCOMO II model.

![Figure 1: Thesis outline.](image)

### 3.2.1 Literature review

The literature review is conducted with the purpose to describe the underlying theories of the meta model and to ensure that the meta model contains the most relevant theories. It is focused on the algorithmic cost modeling theories that make use of either function point estimates or source lines of code such as COCOMO II and the function point analysis (Boehm et al., 2000; Albrecht & Gaffney Jr, 1983). The literature review covers aspects of modeling theory as well as software engineering and forecasting methodology.

### 3.2.2 Elicitation

The elicitation phase makes use of different elicitation methods to collect required data for both the construction of the meta model as well as the application of the meta model.
The techniques used to elicit information are through the use of interviews and workshops with project stakeholders in order to obtain the attribute values needed for the COCOMO II model. The relevant attributes are the Scale Factors (SF) and Effort Multipliers (EM). They described in more detail in chapter 4.

The selection of the interviewees has been based on the following factors, in order:
1. Relevance (project knowledge)
2. Knowledge (domain knowledge)
3. Accessibility

The interviewees’ roles have been project managers, experienced system developers and architects.

The second factor in the list, knowledge, takes the domain knowledge into account. This is needed in order to get accurate estimation regarding the product specific attributes such as product complexity and the amount of required reusability.

The current project cost estimations that have been estimated through expert judgment were also collected for the purpose of measuring the differences of the models outcome and the expert’s estimations. These were collected through the final reports and validated in the interview so ensure that they contained the correct numbers. The final report has also been used in order to validate the information provided by the interviewees. It contains comments about the project, team cohesion, issues, etc.

3.2.3 Construction
The general outline of the meta model was created between the literature review and the elicitation stage.

The meta model construction involves mapping the concepts of the architectural framework to the COCOMO II model. The meta model was created using the Enterprise Architecture Analysis Tool and its calculation capabilities to set up calculation rules that conforms to the estimation model. (Dept. of Industrial information and control systems, 2012)

3.2.4 Analysis
The analysis consists of the empirical data is divided into two different activities.

The first activity is to analyze the empirics and map the targeted architecture onto the meta model concepts that have been developed. This is made by using the information provided by the reports and documentation as well as the information provided by the interviewees. This makes it possible to predict the efforts and schedules within the model as well as visually validating the modeled architecture with the interviewees. Both the calculations and the visualizations of the predictions are made through the Enterprise Architecture Analysis Tool developed by the department of Industrial Information and Control systems at KTH (Dept. of Industrial information and control systems, 2012).
The second activity is to calibrate the model with the empirical data. The purpose of the calibration is to make the model to fit the organization that is using it. In the perfect reality, a calibration would not be needed, but as it is difficult to standardize subjective factors between organizations such as various capabilities it can improve the predictive accuracy. The calibration method utilizes linear regression and the formulas are covered in the theory chapter. The calibration calibrated the multiplicative constants used in the algorithmic model and covers both the scheduling estimations and the effort estimations.

3.2.5 Validation
The validation consists of measuring the accuracy of the model. The accuracy will be measured by using the mean magnitude of the relative error (MMRE) and the magnitude of the relevant error (MRE) (Conte et al., 1985). The mean magnitude of the relevant error has been criticized for only measuring the standard deviation and that it would be necessary to measure the distribution of the variable \( z \) (\( z = \text{estimate/actual} \)) through for instance box plots (Kitchenham et al., 2001).

When comparing prediction models with MMRE there is a high probability that the model selected (i.e. the best model) will provide an estimate below the mean (i.e., will underestimate) compared to a model that predicts the mean. There is however no available universal replacements for MMRE proposed thus it have been continuously used (Foss et al., 2003).

As there have no new methods that successfully have replaced MMRE this thesis will use that to validate the results against the case study. The major problem of MMRE is when selecting the best model out of a set of models and since this thesis is not an evaluation of several methods, it is a suiting choice. The formula of how to calculate the MMRE is shown presented in formula 1.

\[
\text{MMRE} = \frac{1}{n} \sum_{i=0}^{n} MRE_i
\]

\[
MRE = \frac{|E - \hat{E}|}{E}
\]

Where \( E \) is the actual result and \( \hat{E} \) is the estimate.

A model has an acceptable accuracy level if 75% of the projects’ estimations are higher or equal to 75% (Conte et al., 1985). This is called the prediction quality (PRED) and has been used frequently when comparing models and methods within the area of software estimation (Lagerström et al., 2010; Kemerer, 1987). Despite the question marks regarding both PRED and MRE, researchers advocate that the measures should “... at very least be placed in an accessible, precise, and analyzable context” (Korte & Port, 2008).

The prediction quality formula is shown in formula 2 where \( n \) is the complete set of projects and \( k \) is the amount of projects that have greater or equal accuracy to \( q \).
Formula 3 shows the prediction quality formula fully expanded.

The acceptable accuracy level for the model can now be denoted $PRED(0.25) = 0.75$, meaning that a 75% of the project samples shall be within ±25% of the actual result.
4 THEORY

In this chapter the theory used in order to create the meta model is described. The theory for the model is influenced by COCOMO II and the meta model by Lagerström (Lagerström et al., 2010; Boehm et al., 2000). The COCOMO II relevant parts are described more into details while Lagerström’s work has served as an influence and guideline for the meta model construction.

There are claims that a combination of estimates from independent sources, preferable applying different approaches, will on average improve the estimation accuracy. This can be used to combine expert elicitation (i.e. the projects’ estimates) with the prediction from the model. Research have shown that combinations between model and expert estimates produces up to 16% above the best single decision (Blattberg & Hoch, 1990).

4.1 Algorithmic cost modeling

Algorithmic cost modeling makes use of mathematical formulas to estimate the project costs. These algorithms can depend on both subjective and objective attributes, such as personnel experience and complexity of the product. The most used formula for algorithmic cost modeling is shown in formula 4.

\[ \text{Effort} = A \times \text{Size}^E \times M \]  

(4)

\( A \) is a constant factor which depends on the organizations practices and the type of software to be produced. \( \text{Size} \) can be both estimated in source lines of code as well as approximated in function or application/object points. The exponent \( E \) is a constant, sometimes called the scale factor, which is used to scale the project depending on its size. This is to reflect the fact that the project’s cost does not increase linearly with the project size. The value of the constant \( E \) is usually between 1 and 1.5, where the latter would correspond to a large diseconomy of scale. (Sommerville, 2010).

4.1.1 Function point analysis

The method of using function points to estimate size of software development was created by Allan Albrecht (Albrecht, 1979) in the late 1970s. The general approach of function point estimates is to count the number of inputs, inquiries and outputs to be developed. The definitions of the measurements published in (Albrecht & Gaffney Jr, 1983) are shown in the following list.

- **External input types** - Counts unique user data or control inputs that enters the external boundary of the application which adds or changes data in a logical internal file type. The external input is considered to be unique if it, compared to another internal input type, have:
  - Different format
  - Different processing logic
• **External output types** - Counts unique user data or control output that leaves the application. The external output is considered to be unique if it, compared to the another external output types, has:
  - Different format
  - Different processing logic

• **Logical internal file types** - Counts major logical groups of user data or control information in the application. In the context of a database, count each logical group of data that the application *uses, generates or maintains.*

• **External interface file types** - Counts all files passed or shared between applications. Count all major logical groups of user data or control information that *enters or leaves* the application.

• **External inquiry types** - Counts each unique input or output combination where the input to the application generates an immediate output. The external inquiry is considered to be unique if it, compared to the other external inquiry types, has:
  - Different format in either input or output parts
  - Different processing logic

Once these have been counted, they are weighted according to a predefined table based on earlier collected statistics.

Depending on the complexity of the inputs, inquiries and outputs, they will receive additional weighting and the result of this produces the Unadjusted Function Points (UFP). The last step of the calculations is to consider the processing complexity in the application. This is done by estimating the degree of influence of the application on 14 different characteristics. These are used to develop an adjustment factor ranging from 0.65 to 1.35 (Albrecht & Gaffney Jr, 1983). The possible measures are listed in table 1.

<table>
<thead>
<tr>
<th>Influence</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not present or no influence</td>
<td>0</td>
</tr>
<tr>
<td>Insignificant influence</td>
<td>1</td>
</tr>
<tr>
<td>Moderate influence</td>
<td>2</td>
</tr>
<tr>
<td>Average influence</td>
<td>3</td>
</tr>
<tr>
<td>Significant influence</td>
<td>4</td>
</tr>
<tr>
<td>Strong influence</td>
<td>5</td>
</tr>
</tbody>
</table>

*Table 1: Degree of influence measures and their ratings.*
The list of the complexity characteristics used to for the processing complexity is shown in Table 2.

<table>
<thead>
<tr>
<th>ID</th>
<th>Characteristic</th>
<th>ID</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>Data Communications</td>
<td>$c_8$</td>
<td>Online Update</td>
</tr>
<tr>
<td>$c_2$</td>
<td>Distributed Functions</td>
<td>$c_9$</td>
<td>Complex Processing</td>
</tr>
<tr>
<td>$c_3$</td>
<td>Performance</td>
<td>$c_{10}$</td>
<td>Reusability</td>
</tr>
<tr>
<td>$c_4$</td>
<td>Heavily Used Configuration</td>
<td>$c_{11}$</td>
<td>Installation Ease</td>
</tr>
<tr>
<td>$c_5$</td>
<td>Transaction Rate</td>
<td>$c_{12}$</td>
<td>Operational Ease</td>
</tr>
<tr>
<td>$c_6$</td>
<td>Online Data Entry</td>
<td>$c_{13}$</td>
<td>Multiple Sites</td>
</tr>
<tr>
<td>$c_7$</td>
<td>End User Efficiency</td>
<td>$c_{14}$</td>
<td>Facilitate Change</td>
</tr>
</tbody>
</table>

Table 2: Processing complexity characteristics.

The processing complexity is then calculated using formula 5. In the formula, $c_{i,j}$ is the characteristic $c_i$ multiplied with its influence rating $j$ in Table 2.

$$ PC = 0.65 + 0.01 \sum_{i=0}^{14} c_{i,j} $$  \hspace{1cm} (5)

The final function point (FP) measure can then be calculated through the formula:

$$ FP = PC \times UFP $$  \hspace{1cm} (6)

To determine the work needed for software development change project with an existing code base, formula 6 is proposed by Albrecht (Albrecht & Gaffney Jr, 1983).

$$ E = (UFP_{add} + UFP_{change}) \times PC_{after} + UFP_{del} \times PC_{before} $$  \hspace{1cm} (7)

$UFP_{add|change|del}$ is the unadjusted function points of the functionality added, changed and deleted in the project. $PC_{before|after}$ is the processing complexity before the change project started and the estimated complexity in the to-be state, when the project have finished.

Function point based estimation is useful because it is based on information that is available early in the project life cycle. The processing complexity formula is according to Boehm inconsistent with his experience in the development of the Cocomo II model. The reason for this is due to the fact that each of the 14 characteristics in the function point analysis method can only give a maximum of 5% contribution to the estimated effort. Because of this, COCOMO II instead makes
use of the unadjusted function points and applies its own reuse factors, effort multipliers and scale factors (Boehm et al., 1995).

### 4.2 Expert judgment

By utilizing the knowledge and experience from experts with extensive experiences in similar projects, expert judgments can be used to estimate the costs of a software project. Expert judgment is often used when there is little data to obtain and the requirements are vague. One of the most common methods of the expert judgment is Delphi (Khatibi & Jawawi, 2011). With Delphi, a group of experts are asked to provide an estimate and reasons for their estimates as well as responding to the other expert’s estimates and reasons. This feedback can then be used by the experts to revise their estimates. Then the process can iterate if there are revisions made. The whole process is completed once the estimates have converged and an approval has been made (Khatibi & Jawawi, 2011; Green et al., 2007).

Expert judgment might be unreliable when experts are misled by irrelevant information. An early estimate based on little information strongly biases the re-estimation, although the estimators had been told to not make use of the early estimate of the input (Jørgensen & Sjoberg, 2001). Because of this it is a good idea to combine the model with the expert judgment to avoid misled experts.

### 4.3 COCOMO II

Boehm et al. (2000) selected five factors that describe economies or diseconomies of scale in software projects. This is based on the theory that depending on these variables, the productivity in the project can increase or decrease as it gets larger. If the scale factor would be one, the productivity would be linear. If the scale factor would be higher or lower, the productivity in the project will be estimated to be of exponential value, either increasing or decreasing depending on the variables. The formula that Boehm stated is shown in equation 8. $SF_j$ is the scale factor $j$ ranging from 1 to 5.

The factors in COCOMO II are rated with the options Very Low, Low, Nominal, High, Very High and Extra High. Some factors do not utilize the whole scale meaning that for instance the ratings “Very High” and “Extra High” holds the same value for the factor. A low rating does not necessarily mean that the effort will increase and corresponding, a high rating does not necessarily mean that the effort will decrease. The whole list of the attribute values can be seen in Appendix C.

#### 4.3.1 Scale Factors

The scale factors are listed and described in the following subsections. $B$ is a constant derived from gathered data from finished projects.
4.3.1.1 Precedentedness
Precedentedness reflects the previous experiences of this type of project. If the organization is completely familiar within this system domain, the value should be extra-high. If the organization has no previous experience, it should be very low. (Boehm et al., 2000, p. 33)

4.3.1.2 Development Flexibility
The development flexibility factor reflects the degree of flexibility in the development process. If the need for software conformance with pre-established requirements or external interfaces is basic, the flexibility would be extra high. On the other hand, if the need for the conformance is high the flexibility would be considered to be very low (Boehm et al., 2000, p. 33).

4.3.1.3 Architecture / Risk Resolution
The architecture or risk resolution factor reflects the extent of the risk analysis carried out. If the risk analysis is thorough and detailed, extra high would be used. If there is little or no analysis, very low should be used. The factor is a result of a consolidation of two factors (Design thoroughness by Product Design Review & Risk Elimination by Product Design Review) from the Ada COCOMO model. It is consolidated to form a more comprehensive definition of the factor (Boehm et al., 2000, p. 34).

4.3.1.4 Team Cohesion
Team cohesion reflects the status of the interactions within the development team in the project. If the development team knows each other well and have worked together before without any communication problems, extra high should be used. If the team has very difficult interactions, very low should be used (Boehm et al., 2000, p. 34).

4.3.1.5 Process Maturity
According to Bradford (1997) the software process maturity is significantly affecting the software development efforts. The same research also shows that the PMAT factor has higher statistical significance as an exponential variable than as a multiplicative variable, proving that the process maturity can be capable of reducing the diseconomy of scale.

The process maturity is a scale factor in COCOMO II that rates how mature the software development process is. The attribute is connected to the SEI Capability Maturity Model (CMM). If the organization does not have a defined CMM for their processes, the Key Process Area questionnaire (KPA) can be used instead (Boehm et
al., 2000). The full questionnaire can be seen in appendix A but an overview of the topics covered is shown in the list below (Boehm et al., 2000, pp. 36-40).

- Requirements Management
- Software Project Planning
- Software Project Tracking and Oversight
- Software Subcontract Management
- Software Quality Assurance
- Software Configuration Management
- Organization Process Definition
- Training Program
- Integrated Software Management
- Software product Engineering
- Intergroup Coordination
- Peer Reviews
- Quantitative Process Management
- Software Quality Management
- Defect Prevention
- Technology Change Management
- Process Change Management

If based on the KPA questionnaire, the calculation of the Equivalent Process Maturity Model (EPML) is computed as five times the average compliance of the KPAs of a project. The “Does not apply” and “Don’t know” selections are excluded. The weighting is shown in table 3 and the equation is shown in equation 9.

<table>
<thead>
<tr>
<th>Selection</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Almost always</td>
<td>100%</td>
</tr>
<tr>
<td>Frequently</td>
<td>75%</td>
</tr>
<tr>
<td>About half</td>
<td>50%</td>
</tr>
<tr>
<td>Occasionally</td>
<td>25%</td>
</tr>
<tr>
<td>Rarely if ever</td>
<td>1%</td>
</tr>
<tr>
<td>Does not apply</td>
<td>Excluded</td>
</tr>
<tr>
<td>Don’t know</td>
<td>Excluded</td>
</tr>
</tbody>
</table>

Table 3: KPA Questionnaire weighting table.
It should be noted that \( n \) is the number of KPAs that are included, thus all of the “Does not apply” and “Don’t know” should not be counted.

The final PMAT rating is then selected according to table 4

<table>
<thead>
<tr>
<th>PMAT rating</th>
<th>Maturity Level</th>
<th>EPML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Low</td>
<td>CMM Level 1 (lower half)</td>
<td>0</td>
</tr>
<tr>
<td>Low</td>
<td>CMM Level 1 (upper half)</td>
<td>1</td>
</tr>
<tr>
<td>Nominal</td>
<td>CMM Level 2</td>
<td>2</td>
</tr>
<tr>
<td>High</td>
<td>CMM Level 3</td>
<td>3</td>
</tr>
<tr>
<td>Very High</td>
<td>CMM Level 4</td>
<td>4</td>
</tr>
<tr>
<td>Extra High</td>
<td>CMM Level 5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4: Process maturity rating selection table.

### 4.3.2 Cost drivers

COCOMO makes use of a term called cost drivers in its calculations. The cost drivers are the 17 effort multipliers that affect the software development project in either positive or negative way. They are selected in the range from {very low, low, nominal, high, very high} and have specified values depending on the attribute. The nominal value will however always has the value assigned to 1 meaning that the attribute have will have no effect on the effort estimation (Boehm et al., 2000). The cost drivers are divided into four categories: product factors, platform factors, project factors and personnel factors. They have each a different set of factors within their category.

The nominal value of a cost driver is 1, meaning that it does not affect the effort, neither positively nor negatively.

#### 4.3.2.1 Product factors

There are several factors that can affect the effort required to develop a software product. The COCOMO model uses five factors to accomplish this.

- **Required Software Reliability (RELY)**
  Measurement to which degree the software product has to function over a period of time (Boehm et al., 2000, p. 41).

- **Database Size (DATA)**
  This factor captures the effect large test data requirements may have on software development. It is important to consider because of the effort required to gather and generate the test data that will be used to verify or test the applications (Boehm et al., 2000, p. 41).

- **Product Complexity**
The complexity is one of the major cornerstones that affect the effort required in development. It is divided into five areas: control operations, computational operations, device-dependent operation, data management and user interface management operations. These areas are then subjectively weighted and the final factor can according to the COCOMO II.2000 constants boost the effort required to as much as 74% higher (Boehm et al., 2000).

- Developed for Reusability (RUSE)
  The theory behind the RUSE factor is that the higher requirements of reusability of an application, the higher the effort for development will be. The range goes from no reuse to the highest, across multiple product lines. (Boehm et al., 2000, p. 42)

- Documentation Match to Life-Cycle needs
  Depending on the required documentation of the product developed, additional costs may occur. If the documentation has to be very excessive for the life cycle needs it can boost the required effort with up to 24% (Boehm et al., 2000, p. 42).

### 4.3.2.2 Platform factors

The platform factors are referring to the target infrastructure hardware and software such as virtual machines (Boehm et al., 2000).

- Execution Time Constraint (TIME)
  If there is an execution time constraint upon the developed software and the target infrastructure it should be described here. The ratings are described as percentage of consumed execution time resources, ranging from <50 % to 95% of available execution time. (Boehm et al., 2000, pp. 45-46)

- Main Storage Constraint (STOR)
  The main storage constraint rating represents main storage constraints for a software system. The rating ranges from the nominal value of <50% of available storage to extra high that represents 95% of the available storage (Boehm et al., 2000, p. 46).

- Platform Volatility (PVOL)
  The software running on the infrastructure services used can increase the development effort in the case that they are often changing (thus volatile). The more the platform is changing, the higher the costs will be. This is measured in numbers of major and minor changes over a period of time. (Boehm et al., 2000, p. 46)

It is however questionable if the TIME and STOR attributes are still relevant due to the increase in both processing capacity and storage capacities (Boehm et al., 2000).

### 4.3.2.3 Personnel Factors

Next to the size of the development project, the personnel have the strongest influence when predicting the effort required of the project. COCOMO uses two
different types of personnel: Analysts and Programmers. The analysts are the people working on requirement elicitation and high-level design. The capabilities of these two roles are determined by the efficiency and thoroughness as well as the ability to communicate and co-operate (Boehm et al., 2000, pp. 47-49).

- **Analyst Capability (ACAP)**
  Ranges in percentiles, where the 90th percentile of analysts will decrease the cost of the development with 29%.

- **Programmer Capability (PCAP)**
  Determined in the same way as the ACAP but the 90th percentile of programmers lower the costs of 24%, which indicates that the programmers are very important but not as important as the analysts.

- **Personnel Continuity (PCON)**
  The personnel continuity measure the turnover of the personnel: from 3% to 48%.

- **Applications Experience (APEX)**
  This rating depends on the level of experience the project teams developing the software have with the same type of applications.

- **Platform Experience (PLEX)**
  This rating depends on the level of experience on the platform used to develop the software. The platform includes graphic user interfaces, databases, networking and distributed middleware.

- **Language and Tool Experience (LTEX)**
  Rates the experience of the development environment such as the programming language and tools for the life cycle. Tools for requirement analysis, configuration management, project planning and control should all be included in this factor.

### 4.3.2.4 Project factors

The project factors account for specific factors that are specific to the project. These factors include location of the development team, schedule compression and usage of modern software tools (Boehm et al., 2000, pp. 49-51).

- **Use of Software Tools (TOOL)**
  The usage of software tools factor describes how well the software tools are integrated with the development process. The simplest tools would be editors with no life cycle support whereas the best tools integrates with the processes and is mature.

- **Multisite Development (SITE)**
  This factor is a combination of two factors. The first factor, Site Collocation (SITECOL) describes the location of the development team, ranging from international to fully collocation. If the team is distributed, the second factor, Site Communication (SITECOM), describes how well the team can...
communicate. The communication ranges from phone to interactive multimedia.

- Requirement Development Schedule (SCED)
  The SCED rating measures the constraints on the schedule for the project team developing the software. This is rated as the percentage of schedule stretch-out or acceleration compared to the nominal schedule requiring a given amount of effort. Stretch-outs do not add or decrease effort. A compressed schedule does however increase the effort needed up to 43% for the lowest rating which is 75% of the nominal schedule.

4.3.3 Schedule estimation
COCOMO II supports estimation of calendar time between the Life Cycle Objective (LCO) and Initial Operational Capability (IOC) phases in the MBASE/RUP development model (Boehm et al., 2000, pp. 57-58). The formula for schedule estimation is based on the effort of the project excluding the schedule effort multiplier ($PM_{NS}$). The schedule compression is taken into account by multiplying the effort with the compression/expansion percentage. $B$, $C$ and $D$ are constants. The multiplicative constant $C$ can be calibrated as well as the exponent $B$. $E$ is the effort scaling factor which is shown in equation 7.

$$TDEV = C \times PM_{NS}^{D + 0.2 \times (E - B)} \times \frac{SCED\%}{100}$$

(10)

$$C = 3.67$$
$$D = 0.28$$
$$B = 0.91$$

4.3.4 Calibration
Calibrating COCOMO II to the organization typically results in better estimates. The calibration can involve consolidation or elimination of redundant parameters, adding cost drivers that are not yet existing in the model or calibrating the model to existing project data. (Boehm et al., 2000, p. 175)

In this research, the last tailoring method will be used. There are two different ways of calibrating COCOMO II to existing project data:

- Multiplicative constant $A$
- Multiplicative constant $A$ and the baseline exponent $B$

It is recommended to have at least 5 data points for calibration of the multiplicative constant $A$ and at least 10 data points for the multiplicative constant $A$ and the baseline exponent $B$. The local calibration usually improves the prediction accuracy because of the subjective factors in the model such as Analyst Capability, Programmer Capability, Platform Experience, etc. Further, the life cycle activities in the projects covered by COCOMO II may differ from the ones in the particular organization (Boehm et al., 2000, p. 175).
The calibration is made by log-transforming the COCOMO II effort equation.

\[
A = e^{\beta_0} \\
B = \beta_1 \\
\ln(\text{Effort}) = \beta_0 + \beta_1 \cdot \ln(\text{Size}) + \beta_2 \cdot SF \cdot \ln(\text{Size}) + ... + \beta_6 \cdot SF_5 \cdot \ln(\text{Size}) + \beta_7 \cdot \ln(\text{EM}_1) + \beta_8 \cdot \ln(\text{EM}_2) + ... + \beta_{22} \cdot \ln(\text{EM}_{16}) + \beta_{23} \cdot \ln(\text{EM}_{17})
\] (11)

To compute the constant A the equation can be altered to:

\[
\ln(\text{Effort}) = [\beta_1 \cdot \ln(\text{Size}) + \beta_2 \cdot SF_1 \cdot \ln(\text{Size}) + ... + \beta_6 \cdot SF_5 \cdot \ln(\text{Size}) + \beta_7 \cdot \ln(\text{EM}_1) + \beta_8 \cdot \ln(\text{EM}_2) + ... + \beta_{22} \cdot \ln(\text{EM}_{16}) + \beta_{23} \cdot \ln(\text{EM}_{17})] - \beta_0
\] (12)

Where \( \beta_1 = B = 0.91 \) in COCOMO II.

In the same technique, the schedule prediction formula can be log-transformed in order to calibrate the multiplicative constant \( C \).

The COCOMO II calibration uses a multiplicative error structure which minimizes the sum of squared residuals in log space (equation 13). The reason for this it to remove asymmetry and reduce the undue influence of projects in the tail of the distribution, such as very large or very small projects (Boehm et al., 2000, p. 180).

\[
\sum_{i=0}^{n} \left( \ln(p_{\text{accurate}}) - \ln(p_{\text{estimate}}) \right)^2
\] (13)

4.4 Enterprise Architecture

Enterprise architecture analysis has emerged during the last decade. Different frameworks have been used to analyze various attributes in enterprises and their IT architectures. Change project is a common type of project in an enterprise, thus including cost estimation for change projects with enterprise architecture could appeal to architects.

Research in the area has proposed a framework of enterprise architecture analysis using ArchiMate and a computational model “The predictive, Probabilistic, Architecture Modeling Framework” (P²AMF), which is based on Object Constraint Language (OCL) (Johnson et al., 2013). Through P²AMF, the framework can enable calculation of entities in the model. This framework will be the basis of the meta model used to enable COCOMO II estimations.

A meta model in general defines the modeling concept that then can be used to describe the models (Klint et al., 2005). Creating a meta model can be seen as the task of creating a language which has the capabilities to describe relevant aspects of a subject that is of interest for the future users of the created models (Höfferer, 2007).
The reason to base the meta model on ArchiMate is because it keeps the number of entities down, while capturing the substantial number of concepts that need to be modeled (Franke et al., 2013).

### 4.4.1 ArchiMate

ArchiMate is a simple framework intentionally resembling the Unified Modeling Language (UML) (The Open Group, 2009). The reason of using ArchiMate as the basis of graphical notation framework is due its generality, making it possible to extend existing meta models with change project estimation as well as providing a solid ground for future adaptions.

The ArchiMate language consists of three core concepts, namely the *active structure*, *passive structure*, and *behavioral elements*. The passive structure elements are elements on which behavior is performed while the active structure is the entity performing the behavior. These concepts are then specialized in each of the three layers specified in ArchiMate (The Open Group, 2009):

- **Business layer**
  - Offers products and services to external customers
- **Application layer**
  - Supports the business layer with application services which are realized by software applications.
- **Technology layer**
  - Infrastructure services needed to run applications, realized by computers, communication hardware and system software.

The ArchiMate language allows inheritance for specialized properties. This means that the specialized concept inherits the parent concept while other restrictions or properties may apply.

![Figure 2: Meta model of the core concepts of ArchiMate (The Open Group, 2009).](image-url)
The Predictive, Probabilistic Architecture Modeling Framework is a generic framework for system analysis (Johnson et al., 2013). It is based on OCL and used to describe expressions in the Unified Modeling Language (UML). The framework is fully implemented in the Enterprise Architecture Analysis Tool (EAAT) (Ullberg et al., 2010; Johnson et al., 2007; Buschle et al., 2011). The framework has been utilized to calculate the formulas in the COCOMO II model accordingly. In the previous implementations of COCOMO the estimated values have been specified as absolute values. These values are all estimated by at least one person and the possibility of applying a probability distribution for major indicators would be more appropriate. The framework allows this and for instance if an application component A is to be developed and its size, measured in source lines of code, is estimated to be about 20 kSLoC, the estimator can also approximate the probability distribution for the component. It could be defined in P²AMF as for example the normal distribution:

$$P(\text{myApplicationComponent.size}) = \text{Normal}(20000,2000)$$

The end result of this would be that the algorithmic formula used in the model would have a probability distribution indicating the probable cost range of the project rather than a specific mean value. This, in combination with the ArchiMate language, provides a strong basis for an analysis framework for cost estimation.

4.5 Combination of model and expert judgment

The combination of using both algorithmic models and expert judgment can be beneficial according to research by Blattberg & Hoch (1990). The authors suggest using 50% model and 50% manager forecast which in their model leads to a 70:30 split between model and intuition. Their ideas lead to the interesting point of what combination of model and judgment would be most beneficial in terms of software development with the COCOMO II model.

The best combination of two models on a certain project can be described as the function which minimizes the function according to a prediction accuracy function:

$$\min\{Acc(D_A, D_B, D_{real}, Ratio)\}$$  \hspace{1cm} (14)

Where $$D_A$$, $$D_B$$ denotes the predictions of model A and model B while $$D_{real}$$ is the outcome of the project. Ratio is the ratio assignment between model A and B which Blattberg & Hoch (1990) suggest to be 50:50. However, the research by Blattberg & Hoch is not specifically targeting COCOMO II and it is possible that the overall prediction accuracy can increase by altering this ratio. Developing this function, it can be generalized to minimize the accuracy of a set of N projects with the possibility of using constraints $$c_i, c_{i+1}, \ldots, c_n$$:

$$\min\{\sum_{i=0}^{N} Acc(D_{A_i}, D_{B_i}, D_{real_i}, Ratio) \leq c_i\}$$  \hspace{1cm} (15)

The accuracy function can be implemented as the mean relative error giving the function:
Further it can be adapted to ensure predictive quality with the constraint variables. Using $c_i, c_{i+1}, \ldots, c_n \leq K$ it can be ensured that if a possible solution exists, all the projects are at least within $K$ accuracy, thus they have $PRED(K) = 100$. 

\[
\text{Acc}(D_A, D_B, D_{real}, Ratio) = \frac{|Ratio \times (D_A - D_B) - D_B|}{D_{real}} \tag{16}
\]
5 META MODEL IMPLEMENTATION

According to the theory of the COCOMO II model, the final cost in person-month (PM) can be calculated as:

\[ PM = A \times Size^E \prod_{i=1}^{17} EM_i \]  

(17)

The purpose of the meta model implementation is to provide a general way of doing cost and schedule estimations in software development projects. In order to support the model has to support migrations, modifications and newly added software. This also includes being able to use both function points as size input as well as source lines of code.

5.1 Modeling language

ArchiMate is in general used to describe the layers in enterprises’ architectures and to for example show which applications are used in which business processes. ArchiMate is tailored for describing as-is and to-be scenarios. While this is not a problem for most attributes in COCOMO II, there are some factors that are project specific and do not conform to this method. To cope with this, a second meta model will be introduced that handles the project specific factors. These two meta models are then combined in a final viewpoint where the cost analysis can be made and displayed. The most obvious project specific factors from COCOMO II are the “Project factors”. These factors include schedule compression, site communications and collocation, and the usage of software tools within the project. The other part of the project specific meta model is the Scale-factors which include proceededness (of the software to be developed), development flexibility, process maturity, team cohesion and architecture/risk resolution. It may be argued that the process maturity, which is measured through the Capability Maturity Model (CMM) or through the KPA questionnaire, is not project specific and therefore should be placed in the ArchiMate based meta model. There are two major reasons why this has not been done:

- At the case study company, the projects’ processes tended to be driven in different ways depending on the project manager. Because of this, the questionnaire resulted in different rating for different projects, disproving the idea of a uniform process maturity rating for all projects.

- The process maturity rating should be rated at the time when the project starts (Boehm et al., 2000). This means that the process maturity is supposed to be rated for each project, thus it is project specific.

However, if the modeler or the company has the same process maturity, it can be modeled accordingly. This is only an extension that allows for project specific maturity levels.
In order to combine ArchiMate with COCOMO II there is a need to define some new concepts. It is handled by using different meta models and combining them through a view that connects the two different and otherwise incompatible meta models. This way there is no need to alter the definitions of the ArchiMate concepts but rather use them in a new way in the Software Development Change Estimation view.

5.2 Concepts

There are several concepts in the ArchiMate that are used in the meta model. Some of them have been slightly altered to enable new concepts to connect them with.
5.2.1 **Software Development Process**
A Software Development Change process is a business process that constitutes a process where software is changing. The process can only be connected to projects and have the “One-To-Many” relationship; meaning that one process can be used by many projects but a project can only have one process. There’s only one attribute in this class: the Process Maturity. The rationale for this decision is that an organization may share the same process for several projects, thus it would be useful to only have one entity that changes. In the case that the process maturity is unique for a project it can still be modeled with a new instantiation of the class. Process Maturity is the attribute found in the scale factors in COCOMO (PMAT) and has the same functionality as in COCOMO.

5.2.2 **Software Development Project**
New software change projects are modeled from the Software Development Project class. This class has the attributes from the COCOMO project factors. These factors are considered to be project specific and do not fit to the “as-is” or “to-be” oriented ArchiMate language. Because of this, the project is instantiated in the custom meta model.

The projects’ attributes are:
- Usage of Software Tools (TOOL)
- Multisite Development (SITE)
- Required Development Schedule (SCED)
- Precedentness (PREC)
- Development flexibility (FLEX)
- Team Cohesion (TEAM)
- Architecture/Risk resolution (RESL)

The idea of this approach is to abstract the concept of scale factors and effort multipliers for the modeler and combining the thematically same attributes as close to each other as possible.

The projects can be connected to one or more Personnel objects. If the project is connected to several instances of Personnel, the mean of the values are used in the calculations. It does also connect to the class “Activity” which constitutes an activity within a project. A project can have one or more activities while an activity can only be assigned to one project.

5.2.3 **Activity**
The Activity class does not have any user specified inputs. It only serves as an aggregator of “Change”-object to give the modeler possibilities to model the change project in a complete way. The Activity can assign several changes and the Activity Cost attribute is the accumulated values (i.e. sum) of the change cost values in the Change-objects.
5.2.4 Change

The change class is perhaps the most important concept in the meta model. It is the basis of the software change projects and illustrates the changes in the architecture. It is connected to a project activity but also to one or more effort divisors. In the figure of the Software Development Change View it might seem confusing that it has three different relations with the effort divisor where none of them have a “one or more” relation. This is solved through the use of invariants which ensure that either one of these cases is modeled:

- Relation with one or more effort divisors through the “Change New” connector
- Relation with one or more effort divisors through the “Change To” connector and one or more effort divisors through the “Change From” connector.

The first case enables scenarios of creation and modification of services, functions and components. The second case enables migrations with a FROM and TO-state.

The change class has one value that is to be specified by the modeler and that is Change Size. Change Size can be either specified in unadjusted function points (UFP) or source lines of code. In the case where UFPs are used, the gearing factor of the component determines and the estimate of the amount of lines of code to be produced and this is then used in the calculations.

The Change Cost attribute if calculated as in the COCOMO formula with support for what the original author describes as “Multiple Module Effort Estimation” (Boehm et al., 2000).

The Time To Develop (TDEV) attribute is calculated as the schedule estimation formula is described section 4.3.1.

5.2.4.1 Effort Divisor

An effort divisor is a class that supports the functionality of splitting a change size amongst several components. If there is only one component affected by the change the effort divisor is not needed. It is however needed in when several components are affected by a software change. The effort divisor splits the change size as assigned by the modeler. An example could be that when a change that is estimated to be 5 KSLOC is affecting 3 components and the divisors are assigned ¼, ¼ and ½, the result would be as shown in Table 3.

<table>
<thead>
<tr>
<th>Component</th>
<th>Divisor</th>
<th>Assigned change</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>¼</td>
<td>1250 SLOC</td>
</tr>
<tr>
<td>B</td>
<td>¼</td>
<td>1250 SLOC</td>
</tr>
<tr>
<td>C</td>
<td>½</td>
<td>2500 SLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 1</td>
</tr>
</tbody>
</table>

Table 5: Example of effort divisor usage.
The result of this division is that the factors applied to each Component will be weighted by the divisor, leaving the modeler with the possibilities of adding divisors where large changes are made that affect Components with different characteristics, such as very complex components and very simple components at the same time.

5.3 Software Development Change Estimation view

The constructed view contains the 17 effort multipliers as well as the 5 scale factors in a combination. The view is color coded to differentiate between the three ArchiMate layers as well as the software development change meta model:

- Red – Business layer
- Green – Application layer
- Yellow – Infrastructure layer

The view combines the ArchiMate meta model with the software change meta model (blue entities).

Figure 5: Software Development Change View with relationships.
5.4 Application sizing

In a change project, the new application will have a size attribute that needs to be estimated. There are several approaches to estimate the size such as the function point analysis and source lines of code. Depending on the type of project different approaches can be used. The meta model supports both the use of function points and source lines of code. The size of an application service is calculated as the sum of its application functions sizes. The formula used is the following

$$ ApplicationService.\text{Size} = \sum_{i=0}^{n} ApplicationFunction.\text{Size}_i $$  \hspace{1cm} (18)

5.4.1 Software migration using Source lines of code estimation

When migrating an entire application service or component, the source lines of code estimation of the as-is application can be used as an estimation of the size of the to-be application (Boehm et al., 2000).

$$ Change_{\text{Size}} = \frac{Size_{\text{old}}}{GearingFactor_{\text{old}}/GearingFactor_{\text{new}}} $$  \hspace{1cm} (19)

In formula 18, we see that the Size of the new application can be estimated by the size of the old application using the gearing factors to refine the estimations based on the programming language. This leads to the full formula:

$$ ApplicationService.\text{Size} = \sum_{i=0}^{n} \frac{ApplicationFunction.\text{Size}_i}{ApplicationFunction.GF(ApplicationService.GF)} $$  \hspace{1cm} (20)

5.4.2 Sizing using Function Points

The basis of the Gearing Factor is to transform estimations regarding source lines of code to a measure that can be compared across programming languages. The Gearing Factor value of 1 is an equivalent of 1 unadjusted function point. This makes it possible to combine estimates with source lines of code and function points. The difference compared to the traditional function point analysis by Albrecht (Albrecht & Gaffney Jr, 1983) is that COCOMO II has its own way of adjustment using the effort multipliers instead of processing complexity measures. Using formula 19, a function can be estimated using unadjusted function points as its size with a gearing factor of 1 and then transformed to its equivalent source lines of code in the language used for the application. The same formula can be applied to estimate sizes of components by substituting the ApplicationService with ApplicationFunction and ApplicationFunction with ApplicationComponent.

Because of the nature of migration projects much of the requirements of the application may already be present. In the cases where the purpose of the migration
only is to change technology stack; the new system will in general have the same requirements as the old system. In this case the basis of the system size can be made through the old application size. However, when a new application is to be developed, it may be hard to derive the estimate of source lines of code. There are also other benefits of using function points instead of source lines of code, such as the ability to predict non-coding defects such as requirements and specification defects (Jones et al., 2011).

5.5 Element sets

There are several effort multipliers in COCOMO II that correspond to specific entities in a model. These multipliers will in this meta model be connected once for each usage.

5.5.1 Infrastructure Service

The platform factor multipliers Platform volatility, Storage and Time Constraints are factors that each is unique to a specific environment or platform. To exemplify, Application Function A may use two different platforms: one system for the database DB1 and another platform for DB2. On the first platform, one set of constraints may apply and another set may apply for the second platform. Thus, in the model, both platforms are instantiated and shown, and the average of these two platforms will be the resulting platform factor for the application function. This is shown in the formula below:

\[
ApplicationFunction_PlatformFactor = \frac{1}{n} \sum_{i=0}^{n} \sum_{j=0}^{2} InfrastructureService_{i,j}
\] (21)

In the formula, \( InfrastructureService_{i,j} \) corresponds to the \( j \):th factor (Platform volatility, Storage, Time Constraints) of the Application Functions \( i \):th Infrastructure Service.

5.5.2 Application Function

An application function is in ArchiMate defined as a representation of a group of internal behaviors of an application component (The Open Group, 2009). The intentions in the meta model are the same but with the added clarity that there might be several application components that can be assigned to an application function. Application functions are virtual and only show internal behavior. If the functions are supposed to be exposed to end users they should realize application services. The Application Function concept has two evidential properties: database size (DATA) and documentation match to life-cycle needs (DOCU). These properties lie within the application function because their concerns regard groups of components rather than just single components. In case a single component needs a specific DOCU or DATA rating, an additional application function can be instantiated that only connects to the specific component.
The size of an application function is determined by the sum of the unadjusted function points (UFP) of all application components. The UFP can be calculated by dividing the source lines of code by the gearing factor of the application component.

\[
ApplicationFunction.Size = \sum_{i=0}^{n} \frac{ApplicationComponent_i.Size}{ApplicationComponent_i.GF/ApplicationFunction.GF}
\]  

This calculation allows an application function size to be calculated even though it uses different programming languages.

The application function entities have two evidential properties: Product Complexity and database size.

5.5.3 Application Service
The application services are the end services that are used within the organization. These services are realized by one or more application functions, thus these services have attributes that are aggregates of the application functions properties. The mapping is specified in the table below:

<table>
<thead>
<tr>
<th>Function (Many)</th>
<th>Service (One)</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Complexity</td>
<td>ProductComplexity</td>
<td>Average</td>
</tr>
<tr>
<td>Data base size</td>
<td>DataBaseSize</td>
<td>Average</td>
</tr>
<tr>
<td>Size</td>
<td>Size</td>
<td>Sum</td>
</tr>
<tr>
<td>PlatformFactor</td>
<td>PlatformFactor</td>
<td>Average</td>
</tr>
</tbody>
</table>

Table 6: Application function to service attributes mappings.

5.5.4 Application Component
An application component is in the ArchiMate 1.0 specification defined as a modular, deployable and replaceable part of a system (The Open Group, 2009). The component is the key concept in the meta model as it constitutes real parts of the software. The component concept holds the COCOMO attributes “Required Reusability”, “Complexity” and “Size”. It also holds the additional “Gearing Factor” attribute to enable migration calculations.

5.6 Change project cases
The meta model supports 5 different cases of a change project. These range from full service migrations to specific component modifications:

1. Application Service Migration (full migration)
2. Application Function Migration (partial migration)
3. Application Component Migration (partial migration)
4. Application Service Modification
5. Application Function Modification (one or more components involved)
6. Application Component Modification

The six cases are described below. They all have the same characteristic resulting in that components always are subject of changes and while services and functions necessarily don’t have to change. In this context a modification can also involve adding new services, functions or components.

As services and functions are virtual objects, meaning that they have no real existence in the software, only working as an abstraction layer, all changes have the effect of altering the components in the architecture. With this it is meant that changes can be applied to these virtual objects but the real calculations are propagated down in the chain to the components and the components of the virtual objects are used as the basis for the calculations.

5.6.1 Case 1 - Application Service Migration
In the case of full migrations, the Change object should be connected to both an “as-is” application service as well as a “to-be” application service, utilizing the “to-be” attributes as programming language and platform factors while converting the size of the “as-is” application to the new programming languages’ gearing factors. If the migration is dividing the old application service into smaller parts and several new application services would be needed, the concept of the “Effort divisor” is needed to divide the effort to the different services.

The application service has to be connected to application functions and the modeler has to decide how much of the effort each function represents in order for the model to calculate the person-month effort with as correct factors as possible. If there is only one function or if the modeler decides that the functionality is equally split amongst the functions, the average of the factors is used.

5.6.2 Case 2 - Application Function Migration
When an application function is to be migrated, there has to be an existing application service that the migrated function can realize.

There are two ways of migration of an old application function:
- By connecting it to one or more existing functions and adapt them with the changes needed for the migration
- By creating one or more new application functions

The migrated function can either be connected to an existing function (thus modify an already existing) or a new function can be created.
Since functions are connected to services, there have to be existing application services for the functions that are created or modified. An application function in the existing application is either created or modified to adapt to this new functionality and the change is connected to both the adapting/new application function as well as the old application function that is migrated away from. The old function is used as the effort basis (source lines of code or function points) and the new application function has its attributes such as platform factor calculated by its usage of infrastructure. In the case that a function is migrated to more than one application function, it is dividing the effort equally to the new functions. The functions also need to be connected to one or more components because of the factors. In the case there is more than one component for a function, an effort divisor can be used to divide the effort needed to migrate or modify the function for the components.

5.6.3 Case 3 - Application Component Migration
A component is migrated to one or more new components. A change object can use effort divisors in order to divide the effort for the migration. For the calculations, the old components size is used while for the new component one has to specify the complexity and required reusability factors. One might argue that the complexity does not change but there are circumstances when the component to be developed can utilize new libraries, techniques and frameworks to decrease the complexity but also cases where complexity has increased due to performance reasons or similar.

5.6.4 Case 4 - Application Service Modification
When modifications of an application service are needed it can be handled in two ways. Changes can be made to the application service itself, using effort divisors to distribute the effort to the functions. The two other ways of doing changes to an application service is through case 5 and case 6.

5.6.5 Case 5 - Application Function Modification
Modifications to functions can be made by connecting the Change-object to the function itself and the effort divisors can be used to distribute the effort to two or more components. If one component is assigned to the function there is no need for effort divisors.

5.6.6 Case 6 - Application Component Modification
The last case is when changes are made at component level. Component modifications are the changes that are closest to the reality as the components are the real software while functions and services are of virtual nature to enable a layer of abstraction.

6 DATA
The data points used to validate and calibrate the model are projected as having been closed during the last 6 months and satisfy the constraint of having > 2000 SLOC produced in the project. The data have been collected through interviews with
project managers, developers and architects in the projects. The participants have answered the questions needed to select the different ratings for each of the factors in COCOMO II. Project reports have been used to validate the information elicited and as a source of project calendar time and the effort (person-hours) to complete the project.

Appendix A was used as the basis for querying the process maturity attribute because the organization did not have a CMM level for its development process.

Appendix B was used in order to elicit the information needed for the COCOMO II related factors as input on how the visual model should look. An example of how a model of a software development change project can look like can be seen in figure 6, which represents a slightly (visually) modified version of the change project. The degree of detail in the model is up to the modeler to decide but in these projects a high level of abstraction has been used. The reasons for this is to not deviate to far from the original COCOMO II model in order to simplify the validation process, i.e. compare with COCOMO II calculation output. Another reason is due to the fact that it is more time consuming to go into a higher level of detail when the modeler (in this case the author) does not have a complete understanding of the systems.

6.1 Project A

The purpose of the project was to construct a new system that handles documents related to production parts and their life cycle needs. The system is used by suppliers to the manufacturing company. The system was implemented as a module of an already existing platform. The scope of the system changed during the project as the complexity of certain functionality was higher than expected. This caused a half year delay of the project. The whole project took 12 700 man hours distributed over a period of 28 months in calendar time. The implementation phase was during 15 of these months. The project manager changed once and the information about the project were gathered by interviews with the last project manager as well as one of the developers.

In general, the project was a successful project. Most things worked well according to the customers of the project and the communication with the project members were overall good.

The project team members were collocated in the same office and the communication with the customer occurred both through video link and in person meetings.

According to the project participants some of the requirements with the project’s stakeholders were not always clear and the scope changed because of this. One of the comments was that “... the decisions regarding time and cost is taken according to a waterfall principle but development is done in iterations. Scope and cost has to be revised continuously.”. Stakeholders did also have deviating opinions regarding the solution which caused delay in development and re-implementations.

The overall conclusion is that the project manager believes that it was a successful project despite the estimations that were made prior to the project start. The team consists of highly experienced members.
6.1.1 Sizing
The calculations of source lines of code of the newly developed system were made in two steps. There were two main changes in the project: a new web application and modifications to an existing application. The web application consisted of about 33 KSLOC written in the .NET platform. The modification part of the system was a little harder to get the accurate estimate of source lines of code changes. The development was made in a 4th generation language but the trouble of counting came with the development method. The methods of development involved copying whole functions of code from already existing code while performing minor changes to each such function. This resulted in that 20 lines of code function could be added to the software while only one of the lines was actually “programmed”. According to the model the goal is to measure only the amount of intellectual work put into the development of the software (Boehm et al., 2000). With this in mind, the produced source lines of code into the modification were estimated to be 50 KSLOC. Thus, in total, the project produced 83 KSLOC.

6.1.2 Result

<table>
<thead>
<tr>
<th>Scale Factors</th>
<th>Rating</th>
<th>Effort Multipliers</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>HIGH</td>
<td>RELY</td>
<td>NOMINAL</td>
</tr>
<tr>
<td>FLEX</td>
<td>NOMINAL</td>
<td>DATA</td>
<td>HIGH</td>
</tr>
<tr>
<td>RESL</td>
<td>NOMINAL</td>
<td>CPLX</td>
<td>NOMINAL</td>
</tr>
<tr>
<td>TEAM</td>
<td>HIGH</td>
<td>RUSE</td>
<td>NOMINAL</td>
</tr>
<tr>
<td>PMAT</td>
<td>NOMINAL</td>
<td>DOCU</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

Factors:

\[
E = 1.0763 \\
EM_{project} = 0.88 \\
EM_{product} = 1.05 \\
EM_{platform} = 1.01 \\
EM_{personel} = 0.30 \\
EM_{tot} = 0.28
\]

Actual

\[
PM_{act} = 83.55 \\
TDEV_{act} = 15 \\
SLoC_A = 50000 \\
SLoC_B = 33000
\]

Table 7: Project A COCOMO II results.
6.2 Project B

This project was initiated for the purpose of replacing an old application with a new one running on the company’s standardized platform with included support and development agreements. The old application was based on old technology and
could not run on modern PC’s such as the ones based on the x64 architecture. The program is used to determine variables of the propeller shaft used in the produced vehicles. It is only used by the experts in the area and the old application did only run on one PC. Overall, the project was deemed successful. Deviations in the project schedule happened due to the complexity in the algorithms that were implemented to do the calculations. The project utilized the software development method working iteratively in sprints with demonstrations to customers after each of the sprints (Schwaber & Sutherland, 2011). This was one of the reasons why the communication between the project team and the stakeholders were deemed as good. The project had an 18% overrun of the estimated budget due to deviations between the old application and the new application. These deviations increased the scope of the project, thus caused the increase in time.

6.2.1 Sizing

The sizing in this project was straight forward as it only consisted in migrating one web application. The project resulted in 5.5 KSLOC lines of code developed on the .NET platform.

<table>
<thead>
<tr>
<th>Scale Factors</th>
<th>Rating</th>
<th>Effort Multipliers</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>NOMINAL</td>
<td>RELY</td>
<td>LOW</td>
</tr>
<tr>
<td>FLEX</td>
<td>LOW</td>
<td>DATA</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>RESL</td>
<td>HIGH</td>
<td>CPLX</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>TEAM</td>
<td>VERY HIGH</td>
<td>RUSE</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>PMAT</td>
<td>HIGH</td>
<td>DOCU</td>
<td>VERY HIGH</td>
</tr>
</tbody>
</table>

Factors:

\[
E = 1.0582 \\
EM_{project} = 0.774 \\
EM_{product} = 2.23 \\
EM_{platform} = 1 \\
EM_{personnel} = 0.32 \\
EM_{tot} = 0.55 \\
PM_{act} = 17.42 \\
TDEV_{act} = 8 \\
SLoC = 5500
\]

Table 8: Project B COCOMO II results.

6.3 Project C

The purpose of project C was to redesign the earlier platform to enable the handling of conversions, software sales and updates for the software in the vehicles in the
same platform and application. This included saving historical information about the software in one place as well as making sure that the system has the capacity to handle the current as well as near future needs. The development was made in the .NET platform containing several web services that are accessible and can be reused by other applications and systems. The project involved designing new architectures and was regarded to be of high complexity due to complex control operations, database management and performance requirements. The communication in the team was mostly in the same building but when that was not possible, video links were used for communication when needed. The project itself was finished on time but there was a budget overrun of the estimated 6100 hours to the final 8060 which gives a MRE of almost 25%. Scrum was used as the general development process while some of the changes in the system’s integration were made as system change requests and delivered to external teams. These requests were followed up at project meetings that were kept regularly every week. A dedicated team tester tested the functionality iteratively during the development and the elimination of bugs was being handled as soon as they were found. In the project report it can be read that the way of working within the project were deemed as very good and that it should not be changed for future projects.

6.3.1 Sizing
The source lines of code produced by the project were in total 23 KSLOC. The three largest components in the system were the ConversionOrder component (~4.5 KSLOC), software update component (~4 KSLOC) and the report system (~3 KSLOC).

The calculations of the source lines of code were made by the static code analyzer in Visual Studio.
<table>
<thead>
<tr>
<th>Scale Factors</th>
<th>Rating</th>
<th>Effort Multipliers</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>VERY LOW</td>
<td>RELY</td>
<td>HIGH</td>
</tr>
<tr>
<td>FLEX</td>
<td>NOMINAL</td>
<td>DATA</td>
<td>HIGH</td>
</tr>
<tr>
<td>RESL</td>
<td>NOMINAL</td>
<td>CPLX</td>
<td>HIGH</td>
</tr>
<tr>
<td>TEAM</td>
<td>VERY HIGH</td>
<td>RUSE</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>PMAT</td>
<td>NOMINAL</td>
<td>DOCU</td>
<td>NOMINAL</td>
</tr>
</tbody>
</table>

Factors:

\[
E = 1.1026 \\
EM_{project} = 0.67 \\
EM_{product} = 1.68 \\
EM_{platform} = 1.11 \\
EM_{personnel} = 0.43 \\
EM_{tot} = 0.55
\]

Actual

\[
PM_{act} = 53.03 \\
TDEV_{act} = 14 \\
SLoC = 23000
\]

Table 9 Project C COCOMO II results.

6.4 Project D

Project D is the smallest of the four projects in the sample. The project was initially estimated to take ~630h which at a later stage in the project were updated to ~920h. The final effort needed was 1210 man-hours. The project was officially closed after 11 months in schedule time but was released in production after only 7 months.

It was initiated because of earlier research of the application portfolio in the company. The result of the research indicated that three systems used the same type of information and it would be beneficial in terms of maintenance and support to merge these into one system instead. The project also involved development of new web services to enable access and reuse of the information in the database for other systems.

The budget of the project was exceeded in the project because of two reasons.

1. Work on integration with other systems was not included.
2. Unplanned performance testing was need.

The unplanned performance testing occurred due to latency issues when working through firewalls.
Deviations and bugs were resolved during the development while issues that were classified as “nice to have” did not alter the planning since they were added to a feature list that are prioritized and planned for future releases.

6.4.1 Sizing

The project had its purpose to replace three applications which handled similar information. The new application was created to support collaboration with other services as well as a web interface for searching and administrative purposes.

The old system had approximately 4600 source lines of code in the service codebase and the web parts consisted of about 3900 source lines of code, both written in C#.

The new system had two components that together replaced the three other applications. The new components consisted of a service component and a web layer, consisting of about ~4100 lines of code respectively ~5000 lines of code, written in C#.

It should however be noted that the newly developed application has been adding functionality in the web layer which makes the comparison between the source lines of code slightly unreliable.

The total lines of code in the old systems were 8500 SLoC and the new application 9100 source lines of code – an increase by ~12%.

<table>
<thead>
<tr>
<th>Scale Factors</th>
<th>Rating</th>
<th>Effort Multipliers</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>LOW</td>
<td>RELY</td>
<td>NOMINAL</td>
</tr>
<tr>
<td>FLEX</td>
<td>NOMINAL</td>
<td>DATA</td>
<td>HIGH</td>
</tr>
<tr>
<td>RESL</td>
<td>NOMINAL</td>
<td>CPLX</td>
<td>LOW</td>
</tr>
<tr>
<td>TEAM</td>
<td>VERY HIGH</td>
<td>RUSE</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>PMAT</td>
<td>NOMINAL</td>
<td>DOCU</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

Factors:

\[
\begin{align*}
E &= 1.09 \\
EM_{project} &= 0.67 \\
EM_{product} &= 1.27 \\
EM_{platform} &= 1 \\
EM_{personell} &= 0.31 \\
EM_{tot} &= 0.27 \\
\end{align*}
\]

Actual:

\[
\begin{align*}
PM_{act} &= 7.95 \\
TDEV_{act} &= 8 \\
SLoC &= 9100 \\
\end{align*}
\]

Table 10: Project D COCOMO II results.
7 ANALYSIS

The study gathered data from four different projects. The subjects of the interviews have been briefed with a presentation of the model, how it works why the data is important to the study.

The thesis requires analysis of the data elicited from the project members. It does however also require analysis regarding the data that is needed for the calibration as well as the combination of the model and expert judgment. In these phases the most accurate figures regarding person-hours spent in the project needs to be known as well as the calendar time spent in the project.

7.1 Input from project members

To ensure that a specific rating was as correct as possible, the interviewees had to justify the decision by describing why a specific factor should have the assigned rating. Some of the ratings require less interpretation such as the personnel continuity (PCON) which can be calculated instead.

The multisite factor got assigned to very high on all of the four projects. This is because the fact that the case study company utilizes the latest technology to increase efficiency in communication.

The KPA questionnaire (Appendix A) is time consuming and will in these relatively small projects not alter the efforts greatly. Because of this, the author decided to not go into detail describing the several areas but instead give a broader description of the areas. This resulted in the process maturity ratings of three nominal values and one high value.

When the product factors such as complexity were discussed, the interviewees had time to go through each of the columns listed in the Appendix B’s application complexity table.

In the case of insecure interviewees or when they were in doubt, the selections have consistently been rounded closer to the nominal value. For instance, if the interviewees don’t know whether the factor should be high or nominal, the nominal value has been chosen; completely according to the COCOMO II model (University of Southern California, 2000).

The toughest question to get an accurate answer from has been the software sizing question. How many source lines of code have been produced? The definition of a source line of code used is the suggested by Boehm (2000, p. 77-81). Even though the definition exists, the size might not be tracked and when the project have modified software rather than been creating new, it can be confusing to know what to count. The key guideline here have been to together with the project members estimate the amount of intellectual created code and exclude parts that have been copied from other projects.

The input from the project members that was required to utilize the meta model implementation in order to create the models do not alter the end results per se, but rather the graphical end result.
7.2 Calibration

The calibration of the model was made with the USC COCOMO II software tool developed by the University of Southern California (2000) which implements and supports the linear regression technique described in the theory chapter (Boehm et al., 2000, pp. 177-178). The case study company tends to run several rather small projects instead of one large project, reducing the effect of the scale factors such as process maturity. The size of the projects ranges from the smallest of ~1200 man hours to the largest of almost 16 000 man hours.

It is eminent that the case study company is conforming rather well to the pre-defined constants in COCOMO II for the two largest projects (~8000h and ~1600h) while the smaller projects are not equally accurate. However, the schedule estimation is rather accurate for all four projects.

It is recommended to have at least 5 data points for calibration of the multiplicative constant A and at least 10 data points for the multiplicative constant A and the baseline exponent B (Boehm et al., 2000, p. 175). As this case study only involved 4 projects, only the multiplicative constant A was calibrated.

As the effort of the projects (person months) should not include requirements but should include software development, management, configuration management and quality assurance. The case study company have a separated IT company that is wholly controlled and owned by the company but operates as an own company. The decision was made to only count the hours spent by this IT provider.

This might seem unnecessary but as the customers of the projects are located within the same company it is necessary to separate them clearly, to properly align the model with the case company.

7.3 Combination of model and expert judgment

Previous research has shown that it can be beneficial to use an estimation technique combining 50% of an expert judgment with 50% of the prediction model result (Blattberg & Hoch, 1990).

The development process used at the case study company forces two estimations at different phases in the project. In the initiation of the project a first estimate is made which is very rough and the purpose of the estimate is to provide the management with some overall figures. If the cost of the project would be much higher than expected, it could be canceled. The second estimate is typically a more informed prediction and is more accurate. The estimates used in the combinations are the second estimates.

The constraints on the combination give different results on the best combination match. If the requirement is that all projects’ estimates shall be within 25% of the real efforts (i.e. \( \text{PRED}(0.25) = 100 \)), the expert estimates yield a higher ratio. On the other hand, if the goal is to achieve the lowest possible mean magnitude of relative error, the model yields the highest ratio. Microsoft Excel Solver has been used for the optimization using the nonlinear optimization algorithm Generalized Reduced Gradient (GRG2) (Microsoft, 2013).
8 RESULTS

8.1 Validation

The implementation of the meta model is validated against the USC COCOMO II tool (University of Southern California, 2000).

The validation was performed by using the same attribute values in the model calculation and in the USC COCOMO II tool and then comparing the results to ensure that the results are the same.

The calculation model results prove that the model produces the same results if the logical model is the same in the USC COCOMO II tool. However, if the modeler decides to utilize some of the benefits of the visual presentations and for instance separate the platform factors in COCOMO II for different infrastructure services, the model can slightly deviate from the USC COCOMO II tool. The reason is simply that in COCOMO II the platform factors for one application or module is the subjective average of the all platform services utilized. The meta model supports the modeler to include and model the platform services independently, assigning the platform factors for each of these. The result of Application function B’s platform factor is thus the average of Service A, Service B and Service C.

This is not an error in the calculation model but merely a necessary adaptation of the logic to enable flexibility in the modeling.

![Diagram](image)

Figure 7: Calculation differences in COCOMO II platform factors and calculation model.
8.2 Accuracy

Even before calibration the model conforms rather well to the data gathered. The two largest projects, project A and C are within the predictive quality margin of 25% (16% and 4%).

Project B is not estimated accurately and has a MRE of 44%. The model underestimates the effort needed for the project which partly may be because of the additional effort needed due to the problems found in the old application that was migrated from.

Compared to the expert estimates the model produces competitive estimates. In the table the mean relevant error has been computed with four different measures. These are the model and expert estimates as well as two combinations of them. The two combinations are the result of the optimal combination between model and expert estimates for the specific purpose. Optimal predictive quality (Opt. Pred) ensures that all projects are within 25% of the real effort outcome. The optimal mean relevant error (Opt. MRE) uses the combination that gives the lowest average MRE for the projects.

<table>
<thead>
<tr>
<th>Project</th>
<th>Measured in hours</th>
<th>MRE</th>
<th>Measured in hours</th>
<th>MRE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>14794</td>
<td>9700</td>
<td>12700</td>
<td>16%</td>
</tr>
<tr>
<td>B</td>
<td>1494</td>
<td>2140</td>
<td>2648</td>
<td>44%</td>
</tr>
<tr>
<td>C</td>
<td>7748</td>
<td>6500</td>
<td>8060</td>
<td>4%</td>
</tr>
<tr>
<td>D</td>
<td>1315</td>
<td>918</td>
<td>1209</td>
<td>9%</td>
</tr>
</tbody>
</table>

**Table 11:** Case study (M)MRE results with different methods before calibration.

The optimal predictive quality (Opt. Pred) is using 24% model and 76% expert. The optimal MRE (Opt. MRE) is using 59% model and 41% expert.

Table 11 shows that heading for the optimal predictive quality in the model would lower the mean magnitude of relevant error, while the optimal MRE achieves a very good mean magnitude of relevant error. From the result it also can be seen that by combining the expert judgments with the model both increases the predictive quality as well as MMRE. This is in line with the previous research (Blattberg & Hoch, 1990).

The accuracy of the schedule estimation can be seen in table 12. The mean MRE of 10.8% without calibration is promising. Project A were a little more off than the others.
### 8.2.1 Calibration

The calibration of the multiplicative constants for the effort estimation and schedule estimation were calibrated using the USC COCOMO II.2000.4 tool (University of Southern California, 2000).

The calibration resulted in an increased value of the two constants. The multiplicative constant $A$ used in the effort estimation increased from 2.94 to 3.23 whereas the multiplicative constant $C$ increased from 3.67 to 3.97. As it shows in table 13, the calibration yields a lower MMRE for the model estimation. This is because the calibration is minimizing the sum of squared residuals in log space rather than the magnitude of relative error (MRE). The optimal predictive quality (Opt. pred) was achieved using 31% model and 69% expert while the optimal MRE (Opt. MRE) was achieved by using 46% model and 54% expert.

<table>
<thead>
<tr>
<th>Project</th>
<th>Model</th>
<th>Actual</th>
<th>MRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12.56</td>
<td>15</td>
<td>16.2%</td>
</tr>
<tr>
<td>B</td>
<td>7.44</td>
<td>8</td>
<td>6.9%</td>
</tr>
<tr>
<td>C</td>
<td>12.83</td>
<td>14</td>
<td>8.3%</td>
</tr>
<tr>
<td>D</td>
<td>7.26</td>
<td>8</td>
<td>9.26%</td>
</tr>
</tbody>
</table>

**Table 12: The accuracy of the schedule estimation (TDEV).**

<table>
<thead>
<tr>
<th>Project</th>
<th>Measured in hours</th>
<th>MRE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model</td>
<td>Estimate</td>
</tr>
<tr>
<td></td>
<td>Model</td>
<td>Expert</td>
</tr>
<tr>
<td>A</td>
<td>16250</td>
<td>9700</td>
</tr>
<tr>
<td>B</td>
<td>1640</td>
<td>2140</td>
</tr>
<tr>
<td>C</td>
<td>8510</td>
<td>6500</td>
</tr>
<tr>
<td>D</td>
<td>1445</td>
<td>918</td>
</tr>
</tbody>
</table>

**Table 13: Case study (M)MRE results with different methods after calibration.**

After the calibration the MMRE in the schedule estimation decreased from almost 11% to 3.4% as shown in table 14.
<table>
<thead>
<tr>
<th>Project</th>
<th>Model</th>
<th>Actual</th>
<th>MRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>13.99</td>
<td>15</td>
<td>6.7%</td>
</tr>
<tr>
<td>B</td>
<td>8.29</td>
<td>8</td>
<td>3.7%</td>
</tr>
<tr>
<td>C</td>
<td>14.31</td>
<td>14</td>
<td>2.2%</td>
</tr>
<tr>
<td>D</td>
<td>8.09</td>
<td>8</td>
<td>1.1%</td>
</tr>
<tr>
<td><strong>Mean MRE</strong></td>
<td></td>
<td></td>
<td>3.4%</td>
</tr>
</tbody>
</table>

Table 14: The (M)MRE of the schedule estimation after calibration.
9 DISCUSSION

The results of the case study validates that the implementation of the COCOMO II model in the Enterprise Architecture Analysis Tool works as predicted and that the model estimates on par with the managers in the case study company.

The combination between model and manager estimates achieves performs far better than single selections of model or manager estimations. Without calibration, optimal MMRE strategy achieved a MMRE of 12% with PRED(.25) = 75%. When adding the constraint of PRED(.25) = 100%, the MMRE rose to 18% which was slightly better than the manager estimates (22%) and on par with the model (18%).

The calibration gave a higher MMRE for the model which may seem quite surprising but since the calibration does not minimize the MMRE but rather the sum of squared residuals in log space, this can happen. Even though the MMRE with the calibrated constants increased the combination approach achieved even higher accuracy, resulting in a MMRE of 14% while being constrained by PRED(0.25) = 100%.

The optimal MRE was achieved by using 46% model + 54% estimate and had PRED(0.28) = 100%.

The result shows that COCOMO II in itself is fairly accurate but when combined with expert estimates the results are very promising. In the reviewed projects, the expert estimates are based on the respective project’s manager. The method of estimating is ad-hoc based and heavily relies on the experience of the specific project manager. The accuracy of the manager estimates could further increase if a more structured approach such as the Delphi method was utilized.

The project input was gathered through interviews with project stakeholders such as developers and project managers. The input is subject to loss of quality because of the author’s role as a middle man. In a real estimation process, someone with insight in the project should be responsible for the modeling and gathering of data for the model. During the case study, the author served as the modeler and thus some information might have been misinterpreted or lost. Figure 8 shows how the case study was performed. The analyst (the author in this case) was responsible for both making assumptions of the information and interpreting the data elicited. The project managers and developers also make assumptions regarding for instance complexity, team cohesion, risk management, etc. The project managers’ and developers’ assumptions and estimations are necessary but if they are responsible for performing the modeling, the analyst does not have to make further assumptions and can thus avoid a potential loss in quality of information.
Figure 8: Sources of errors in case study.

Figure 9 shows a proposed solution where the analyst only has responsibility for the further development of and adjustments to the meta model. This increases the effort needed during the estimation phase for the project managers and developers but will add value through a more accurate estimate.

Since the project managers are also responsible for the expert estimates, it is crucial that the expert estimate is produced before the model estimate is calculated. The reason for this is that if the order was the opposite, the model estimate could bias the expert estimate and the combination of model and manager estimate might then not be as accurate (Kitchenham et al., 2002).

As the schedule estimation did not have any recorded estimation (expert estimation) the combination approach could not be utilized. However, the accuracy of the schedule compression seems to be good enough. Even before calibration, there is only one project that have a somewhat low accuracy (Project A; 16.2%) but it is still well below the predictive quality margin of 25%. After calibration, the mean MRE became 3.4% which is a really promising result. It should once again be noted that four project samples are not enough evidence but can rather be seen as an indication.

During the interviews a common comment was that the source lines of code could not possible be an accurate measure of development costs. Even though the fact is that it may be an accurate measure, it is nonetheless an extremely hard figure to estimate in the beginning of a project. The function point estimates are probably a better choice as input. The drawback is the other way around: it is easier to collect produced source lines of code rather that function points if they have not been identified beforehand. A potential solution to this dilemma is to start measuring the software requirements in function points. These function points should include the
product factors (RUSE; RELY, DATA, DOCU) to ensure that the weighting can be applied. Once the function point has been completed the amount of time spent on the function point can be complemented. When the database is sufficiently populated the model can be calibrated against the data.

![Diagram](image-url)

**Figure 9: Integrated modeling error sources.**

It is important to make sure that the attitudes regarding the selections are consistent within the organization. Even if they are consistently wrong the calibration will fix this and the estimations will be accurate. For instance, if an organization would decide to use a custom definition of what a source line of code is and include comments in the definition, the calibration would eliminate the differences and the results would be as accurate, assuming that the edit of the definition have a constant behavior.

### 9.1 Future research

This thesis has outlined several research areas that might be interesting to further explore. This study created a meta model that was based on COCOMO II. Another solution would be to create a model that is tailored for use in enterprise architecture analysis. This would include more subjective measurements such as static code analysis methods measuring for instance product complexity. This could for example be the Halstead complexity measure or the Cyclomatic Complexity by Thomas McCabe (Halstead, 1977; McCabe, 1976).

As the results from the combination of the model and the expert judgments had great accuracy, it would be interesting to see this method with a greater project
sample size. The sample size of 4 is too low to draw any conclusions but the results are promising. Since the combination only were made on the effort (person-hours) and not the schedule, it would also be interesting to see whether the same approach yields better results regarding the schedule too. This was not possible to study in this thesis as there were no expert estimates of the calendar time.

9.1.1 Commercial off-the-shelf integration projects
During the research it was apparent that the model was not sufficient for several projects, such as those that are not development projects per se but rather integrate commercial off-the-shelf (COTS) products instead of aiming to develop the whole application in-house. The COCOMO II model does not suit this kind of project but there are possibilities of adding support for this by utilizing the research in the COnstructive COTS (COCOTS) integration cost model (Abts et al., 2000).
10 CONCLUSION

This study has successfully implemented a meta model based of COCOMO II that can utilize modeling techniques for estimation purposes. The meta model is implemented in P²AMF using EAAT. The sample projects had accuracy in the same level as the experts estimates (MMRE ~20%). It should be noted that the meta model uses the same calculation model as COCOMO II and have no increase nor decrease in accuracy.

The study has however shown that the COCOMO II model can benefit from a combination with expert/manager estimates to produce estimations with higher accuracy than any of the methods alone. The combination that gave the lowest MRE used 46% model estimate and 54% expert estimate yielding a mean MMRE of only 10%.
11 REFERENCES


University of Southern California, 2000. *COCOMO II Cost Driver and Scale Driver Help*. [Online]
Available at:
http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html
[Accessed 15 May 2013].

University of Southern California, 2000. COCOMO II Downloads. [Online] Available at:
http://csse.usc.edu/csse/research/COCOMOII/cocomo_downloads.htm
[Accessed 10 April 2013].

University of Southern California, 2000. COCOMO II Model Definition Manual. [Online] Available at:
[Accessed 14 May 2013].

Appendix A – KPA Questionnaire

1. **Almost always** is checked when the goals are consistently achieved and are well established in standard operating procedures (over 90 percent of the time).

2. **Frequently** is checked when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances (about 60 to 90 percent of the time).

3. **About half** is checked when the goals are achieved about half of the time (about 40 to 60 percent of the time).

4. **Occasionally** is checked when the goals are sometimes achieved, but less often (about 10 to 40 percent of the time).

5. **Rarely if ever** is checked when the goals are rarely if ever achieved (less than 10 percent of the time).

6. **Does not apply** is checked when knowledge is obtained about the project or organization and the KPA, but KPA does not apply to the circumstances.

7. **Do not know** is checked when uncertainties about the KPAs are present.
### Key Process Areas (KPA)

<table>
<thead>
<tr>
<th>Requirements Management</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td>- System requirements allocated to software are controlled to establish a baseline for software engineering and management use.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>- Software plans, products and activities are kept consistent with the system requirements allocated to software.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Project Planning</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Software estimates are documented for use in planning and tracking the software project.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>- Software project activities and commitments are planned and documented.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>- Affected groups and individuals agree to their commitments related to the software project.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Project Tracking and Oversight</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Actual results and performances are tracked against the software plans</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>- Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>- Changes to software commitments are agreed to by the affected groups and individuals.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Subcontract Management</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The prime contractor and the subcontractor agree to their commitments to each other.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>- The prime contractor tracks the subcontractor’s actual results and performance against its commitments.</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>
## Key Process Areas (KPA)

<table>
<thead>
<tr>
<th>Software Quality Assurance (SQA)</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td>• SQA activities are planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
</tr>
<tr>
<td>• Affected groups and individuals are informed of software quality assurance activities and results.</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
</tr>
<tr>
<td>• Noncompliance issues that cannot be resolved within the software project are addressed by senior management.</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
<td>☐️</td>
</tr>
</tbody>
</table>

| Software Configuration Management (SCM) | | | | | | | |
| • SCM activities are planned. | | | | | | | |
| • Selected work products are identified, controlled, and available. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |
| • Changes to identified work products are controlled. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |
| • Affected groups and individuals are informed of the status and content of software baselines. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |

| Operational Process Definition | | | | | | | |
| • A standard software process for the organization is developed and maintained. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |
| • Information related to the use of the organization’s standard software process by the software projects is collected, reviewed, and made available. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |

<p>| Training Program | | | | | | | |
| • Training activities are planned. | | | | | | | |
| • Training for developing the skills and knowledge needed to perform software management and technical roles is provided. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |
| • Individuals in the software engineering group and software related groups receive the training necessary to perform their roles. | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ | ☐️ |</p>
<table>
<thead>
<tr>
<th>Key Process Areas (KPA)</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integrated Software Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The project's defined software process is a tailored version of the organization’s standard software process.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The project is planned and managed according to the project's defined software process.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Software Product Engineering</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The software engineering tasks are defined, integrated, and consistently performed to produce the software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Software work products are kept consistent with each other.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Intergroup Coordination</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The customer's requirements are agreed to by all affected groups.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The commitments between the engineering groups are agreed to by the affected groups.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The engineering groups identify, track, and resolve intergroup issues.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Peer Reviews</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Peer review activities are planned.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Defects in the software work products are identified and removed.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Quantitative Process Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The quantitative process management activities are planned.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The process performance of the project’s defined software process is controlled quantitatively.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- The process capability of the organization’s standard software process is known in quantitative terms.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Key Process Areas (KPA)**

<table>
<thead>
<tr>
<th>Software Quality Management</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>The project's software quality management activities are planned.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Measurable goals of software product quality and their priorities are defined.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Actual progress toward achieving the quality goals for the software products is quantified and managed.</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
</tbody>
</table>

**Defect Prediction**

<table>
<thead>
<tr>
<th>Defect Prevention activities are planned.</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common causes of defects are sought out and identified.</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Common causes of defects are prioritized and systematically eliminated.</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
</tbody>
</table>

**Technology Change Management**

<table>
<thead>
<tr>
<th>Incorporation of technology changes is planned.</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>New technologies are evaluated to determine their effect on quality and productivity.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Appropriate new technologies are transferred into normal practice across the organization.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
</tbody>
</table>

**Process Change Management**

<table>
<thead>
<tr>
<th>Continuous process improvement is planned.</th>
<th>Almost Always</th>
<th>Frequently</th>
<th>About half</th>
<th>Occasionally</th>
<th>Rarely if ever</th>
<th>Does not apply</th>
<th>Don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participation in the organization’s software process improvement activities is organization wide.</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>The organization’s standard software process and the project's defined software processes are improved continuously.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
</tbody>
</table>
Appendix B – COCOMO II elicitation survey

Appendix B is a modified reprint of the COCOMO II Cost Driver and Scale Driver Help manual (University of Southern California, 2000).
<table>
<thead>
<tr>
<th>RELY</th>
<th>Required Software Reliability</th>
<th>This is the measure of the extent to which the software must perform its intended function over a period of time. If the effect of a software failure is only slight inconvenience then RELY is low. If a failure would risk human life then RELY is very high.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOL</td>
<td>Usage of Software Tools</td>
<td>The tool ratings ranges from simple edit and code (very low) to integrated lifecycle management tools (very high)</td>
</tr>
<tr>
<td>SITECOL</td>
<td>Site: Collocation</td>
<td>The selection depends on the developers physical location ranging from wholly international to fully collocated (basically beside each other)</td>
</tr>
<tr>
<td>SITECOM</td>
<td>Site: Communications</td>
<td>The selection depends on how the communication between the developers are made, ranging from phone and mail to interactive multimedia (video chat, etc.)</td>
</tr>
<tr>
<td>SCED</td>
<td>Required Development Schedule</td>
<td>This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. Accelerated schedules tend to produce more effort in the later phases of development because more issues are left to be determined due to lack of time to resolve them earlier. A schedule compress of 74% is rated very low. A stretch-out of a schedule produces more effort in the earlier phases of development where there is more time for thorough planning, specification and validation. A stretch-out of 160% is rated very high.</td>
</tr>
<tr>
<td>ACAP</td>
<td>Analyst Capability</td>
<td>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. The rating should not consider the level of experience of the analyst; that is rated with AEXP, PEXT and LTEX. Analysts that fall in the 15th percentile are rated very low and those that fall in the 95th percentile are rated as very high.</td>
</tr>
<tr>
<td>PCAP</td>
<td>Programmer Capability</td>
<td>Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate. The experience of the programmer should not be considered here; it is rated with AEXP. A very low rated programmer team is in the 15th percentile and a very high rated programmer team is in the 95th percentile.</td>
</tr>
<tr>
<td>AEXP</td>
<td>Applications Experience</td>
<td>This rating is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team’s equivalent level of experience with this type of application. A very low rating is for application experience of less than 2 months. A very high rating is for experience of 6 years or more.</td>
</tr>
<tr>
<td>PEXP</td>
<td>Platform Experience</td>
<td>The Post-Architecture model broadens the productivity influence of PEXP, recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.</td>
</tr>
<tr>
<td>LTEX</td>
<td>Language and Tool Experience</td>
<td>This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. Software development includes the use of tools that perform requirements and design representation and analysis, configuration management, document extraction, library management, program</td>
</tr>
<tr>
<td><strong>PCON</strong></td>
<td>Personnel Continuity</td>
<td>The rating scale for PCON is in terms of the project's annual personnel turnover: from 3%, very high, to 48%, very low.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>DATA</strong></td>
<td>Data Base Size</td>
<td>This measure attempts to capture the affect large data requirements have on product development. The rating is determined by calculating D/P. The reason the size of the database is important to consider it because of the effort required to generate the test data that will be used to exercise the program.</td>
</tr>
<tr>
<td><strong>RUSE</strong></td>
<td>Required Reusability</td>
<td>This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects. This effort is consumed with creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications.</td>
</tr>
<tr>
<td><strong>DOCU</strong></td>
<td>Documentation match to life-cycle needs</td>
<td>Several software cost models have a cost driver for the level of required documentation. In COCOMO® II, the rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs. The rating scale goes from Very Low (many life-cycle needs uncovered) to Very High (very excessive for life-cycle needs).</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>Execution Time Constraint</td>
<td>This is a measure of the execution time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource. The rating ranges from nominal, less than 50% of the execution time resource used, to extra high, 95% of the execution time resource is consumed.</td>
</tr>
<tr>
<td><strong>STOR</strong></td>
<td>Main Storage Constraint</td>
<td>This rating represents the degree of main storage constraint imposed on a software system or subsystem. Given the remarkable increase in available processor execution time and main storage, one can question whether these constraint variables are still relevant. However, many applications continue to expand to consume whatever resources are available, making these cost drivers still relevant. The rating ranges from nominal, less than 50%, to extra high, 95%.</td>
</tr>
<tr>
<td><strong>PVOL</strong></td>
<td>Platform Volatility</td>
<td>&quot;Platform&quot; is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system, and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. This rating ranges from low, where there is a major change every 12 months, to very high, where there is a major change every two weeks.</td>
</tr>
<tr>
<td><strong>CPLX</strong></td>
<td>Product complexity</td>
<td>Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Select the area or combination of areas that characterize the product or a sub-system of the product. The complexity rating is the subjective weighted average of these areas.</td>
</tr>
<tr>
<td>Project factors</td>
<td>Very Low</td>
<td>Low</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td>TOOL</td>
<td>edit, code, debug</td>
<td>simple, frontend, backend CASE, little integration</td>
</tr>
<tr>
<td>SITECOL</td>
<td>International</td>
<td>Multi-city and Multi-company</td>
</tr>
<tr>
<td>SITECOM</td>
<td>Some phone, mail</td>
<td>Individual phone, FAX</td>
</tr>
<tr>
<td>SCED</td>
<td>75% of nominal</td>
<td>85%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Personnel factors</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACAP</td>
<td>15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
</tr>
<tr>
<td>PCAP</td>
<td>15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
</tr>
<tr>
<td>PCON</td>
<td>48% / year</td>
<td>24% / year</td>
<td>12% / year</td>
<td>6% / year</td>
<td>3% / year</td>
<td></td>
</tr>
<tr>
<td>AEXP</td>
<td>2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
<tr>
<td>PEXP</td>
<td>2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 year</td>
<td></td>
</tr>
<tr>
<td>LTEX</td>
<td>2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 year</td>
<td></td>
</tr>
<tr>
<td>Name:</td>
<td>Used By Applications:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform factors</td>
<td>Very Low</td>
<td>Low</td>
<td>Nominal</td>
<td>High</td>
<td>Very High</td>
<td>Extra High</td>
</tr>
<tr>
<td>TIME</td>
<td>50% use of available execution time</td>
<td>70%</td>
<td>85%</td>
<td>95%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOR</td>
<td>50% use of available storage</td>
<td>70%</td>
<td>85%</td>
<td>95%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVOL</td>
<td>major change every 12 mo.; minor change every 1 mo.</td>
<td>major: 6 mo.; minor: 2 wk.</td>
<td>major: 2 mo.; minor: 1 wk.</td>
<td>major: 2 wk.; minor: 2 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product factors</td>
<td>Very Low</td>
<td>Low</td>
<td>Nominal</td>
<td>High</td>
<td>Very High</td>
<td>Extra High</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>-----</td>
<td>---------</td>
<td>------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>RUSE</td>
<td>None</td>
<td></td>
<td>Across project</td>
<td>Across program</td>
<td>Across product line</td>
<td>Across multiple product lines</td>
</tr>
<tr>
<td>RELY</td>
<td>slight inconvenience</td>
<td>low, easily recoverable losses</td>
<td>moderate, easily recoverable losses</td>
<td>high financial loss</td>
<td>risk to human life</td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>DB bytes/Pgm SLOC &lt; 10</td>
<td>10 D/P &lt; 100</td>
<td>100 D/P &lt; 1000</td>
<td>D/P 1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOCU</td>
<td>Many life-cycle needs uncovered</td>
<td>Some life-cycle needs uncovered</td>
<td>Right-sized to life-cycle needs</td>
<td>Excessive for life-cycle needs</td>
<td>Very excessive for life-cycle needs</td>
<td></td>
</tr>
<tr>
<td>CPLX</td>
<td>Control Operations</td>
<td>Computational Operations</td>
<td>Device-dependent Operations</td>
<td>Data Management Operations</td>
<td>User Interface Management Operations</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Very Low</td>
<td>Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IFTHENELSEs. Simple module composition via procedure calls or simple scripts.</td>
<td>Evaluation of simple expressions: e.g., $A = B + C \times (D - E)$</td>
<td>Simple read, write statements with simple formats.</td>
<td>Simple arrays in main memory. Simple COTS-DB queries, updates.</td>
<td>Simple input forms, report generators.</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Straightforward nesting of structured programming operators. Mostly simple predicates.</td>
<td>Evaluation of moderate-level expressions: e.g., $D = SQRT(B^{**2} - 4 \times A \times C)$</td>
<td>No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.</td>
<td>Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.</td>
<td>Use of simple graphic user interface (GUI) builders.</td>
<td></td>
</tr>
</tbody>
</table>
### Change management process

#### Feature - Precedentedness

<table>
<thead>
<tr>
<th>Feature</th>
<th>Very Low</th>
<th>Nominal / High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organizational understanding of product objectives</td>
<td>General</td>
<td>Considerable</td>
<td>Thorough</td>
</tr>
<tr>
<td>Experience in working with related software systems</td>
<td>Moderate</td>
<td>Considerable</td>
<td>Extensive</td>
</tr>
<tr>
<td>Concurrent development of associated new hardware and operational procedures</td>
<td>Extensive</td>
<td>Moderate</td>
<td>Some</td>
</tr>
<tr>
<td>Need for innovative data processing architectures, algorithms</td>
<td>Considerable</td>
<td>Some</td>
<td>Minimal</td>
</tr>
</tbody>
</table>

#### Feature – Flexibility

<table>
<thead>
<tr>
<th>Feature</th>
<th>Very Low</th>
<th>Nominal / High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need for software conformance with pre-established requirements</td>
<td>Full</td>
<td>Considerable</td>
<td>Basic</td>
</tr>
<tr>
<td>Need for software conformance with external interface specifications</td>
<td>Full</td>
<td>Considerable</td>
<td>Basic</td>
</tr>
<tr>
<td>Premium on early completion</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

*PDR = Product Design Review that validates that the architectural design meets the requirements.
*LCA = RUP Life Cycle Architecture: Is the architecture stable, is the construction plan sufficiently detailed, etc.

#### Characteristic – Architecture/Risk Resolution

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR or LCA.</td>
<td>None</td>
<td>Little</td>
<td>Some</td>
<td>Generally</td>
<td>Mostly</td>
<td>Fully</td>
</tr>
<tr>
<td>Schedule, budget, and internal milestones through PDR or LCA compatible with Risk Management Plan</td>
<td>None</td>
<td>Little</td>
<td>Some</td>
<td>Generally</td>
<td>Mostly</td>
<td>Fully</td>
</tr>
<tr>
<td>Percent of development schedule devoted to establishing architecture, given general product objectives</td>
<td>5</td>
<td>10</td>
<td>17</td>
<td>25</td>
<td>33</td>
<td>40</td>
</tr>
<tr>
<td>Percent of required top software architects available to project</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>Tool support available for resolving risk items, developing and verifying architectural specs</td>
<td>None</td>
<td>Little</td>
<td>Some</td>
<td>Good</td>
<td>Strong</td>
<td>Full</td>
</tr>
<tr>
<td>Level of uncertainty in Key architecture drivers: mission, user interface, COTS, hardware, technology, performance.</td>
<td>Extreme</td>
<td>Significant</td>
<td>Considerable</td>
<td>Some</td>
<td>Little</td>
<td>Very Little</td>
</tr>
<tr>
<td>Number and criticality of risk items</td>
<td>&gt; 10 Critical</td>
<td>5-10 Critical</td>
<td>2-4 Critical</td>
<td>1 Critical</td>
<td>&gt; 5 Non-Critical</td>
<td>&lt; 5 Non-Critical</td>
</tr>
<tr>
<td>Characteristic – Team Cohesion</td>
<td>Very Low</td>
<td>Low</td>
<td>Nominal</td>
<td>High</td>
<td>Very High</td>
<td>Extra High</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>-----</td>
<td>---------</td>
<td>------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>Consistency of stakeholder objectives and cultures</td>
<td>Little</td>
<td>Some</td>
<td>Basic</td>
<td>Considerable</td>
<td>Strong</td>
<td>Full</td>
</tr>
<tr>
<td>Ability, willingness of stakeholders to accommodate other stakeholders' objectives</td>
<td>Little</td>
<td>Some</td>
<td>Basic</td>
<td>Considerable</td>
<td>Strong</td>
<td>Full</td>
</tr>
<tr>
<td>Experience of stakeholders in operating as a team</td>
<td>None</td>
<td>Little</td>
<td>Little</td>
<td>Basic</td>
<td>Considerable</td>
<td>Extensive</td>
</tr>
<tr>
<td>Stakeholder teambuilding to achieve shared vision and commitments</td>
<td>None</td>
<td>Little</td>
<td>Little</td>
<td>Basic</td>
<td>Considerable</td>
<td>Extensive</td>
</tr>
</tbody>
</table>

- **Precedentedness**: If the product is similar to several that have been developed before, then it is high.
- **Development flexibility**: The more flexible the requirements, schedules, interfaces, etc., the higher the rating
- **Architecture / Risk resolution**: Captures the thoroughness of definition and freedom from risk of the software architecture used for the project
- **Team Cohesion**: Accounts for sources of turbulence and extra effort due to difficulties in synchronizing the project’s stakeholders: users, customers, maintainers, interfaces and others.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>thoroughly unprecedented</td>
<td>largely unprecedented</td>
<td>somewhat unprecedented</td>
<td>generally familiar</td>
<td>largely familiar</td>
<td>thoroughly familiar</td>
</tr>
<tr>
<td>FLEX</td>
<td>rigorous</td>
<td>occasional relaxation</td>
<td>some relaxation</td>
<td>general conformity</td>
<td>some conformity</td>
<td>general goals</td>
</tr>
<tr>
<td>RESL</td>
<td>little (20%)</td>
<td>some (40%)</td>
<td>often (60%)</td>
<td>generally (75%)</td>
<td>mostly (90%)</td>
<td>full (100%)</td>
</tr>
<tr>
<td>TEAM</td>
<td>very difficult interactions</td>
<td>some difficult interactions</td>
<td>basically cooperative interactions</td>
<td>Largely cooperative</td>
<td>Highly cooperative</td>
<td>Seamless interactions</td>
</tr>
<tr>
<td>PMAT</td>
<td>CMM Maturity Questionnaire or formula CMM level.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C – Scale factors and effort multipliers

This appendix is a reprint of the scale factors and effort multipliers published in Boehm (2000).

### Scale Factors

<table>
<thead>
<tr>
<th>Driver</th>
<th>VL</th>
<th>L</th>
<th>N</th>
<th>H</th>
<th>VH</th>
<th>XH</th>
<th>Productivity Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>6.20</td>
<td>4.96</td>
<td>3.72</td>
<td>2.48</td>
<td>1.24</td>
<td>1.33</td>
<td></td>
</tr>
<tr>
<td>FLEX</td>
<td>5.07</td>
<td>4.05</td>
<td>3.04</td>
<td>2.03</td>
<td>1.01</td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td>RESL</td>
<td>7.07</td>
<td>5.65</td>
<td>4.24</td>
<td>2.83</td>
<td>1.41</td>
<td>1.39</td>
<td></td>
</tr>
<tr>
<td>TEAM</td>
<td>5.48</td>
<td>4.38</td>
<td>3.29</td>
<td>2.19</td>
<td>1.10</td>
<td>1.29</td>
<td></td>
</tr>
<tr>
<td>PMAT</td>
<td>7.8</td>
<td>6.24</td>
<td>4.68</td>
<td>3.12</td>
<td>1.56</td>
<td>1.43</td>
<td></td>
</tr>
</tbody>
</table>

### Effort Multipliers

<table>
<thead>
<tr>
<th>Driver</th>
<th>RELY</th>
<th>DATA</th>
<th>CPLX</th>
<th>RUSE</th>
<th>DOCU</th>
<th>TIME</th>
<th>STOR</th>
<th>PVOL</th>
<th>ACAP</th>
<th>PCAP</th>
<th>PCON</th>
<th>APEX</th>
<th>PLEX</th>
<th>TOOL</th>
<th>SITE</th>
<th>SCED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.82</td>
<td>0.92</td>
<td>1.00</td>
<td>1.1</td>
<td>1.26</td>
<td>1.29</td>
<td>1.63</td>
<td>1.63</td>
<td>1.49</td>
<td>2.00</td>
<td>1.76</td>
<td>1.51</td>
<td>1.51</td>
<td>1.40</td>
<td>1.53</td>
<td>1.43</td>
</tr>
</tbody>
</table>

Productivity is calculated in different ways depending on if the factor is a scale factor or effort multiplier.

For Scale Factors:

\[ P_{RSF_n} = \frac{(100)^{0.91+(0.01+SF_{n_{max}})}}{(100)^{0.91}} \]

For Effort Multipliers:

\[ P_{REM_n} = EM_{n_{max}} \]