CHAPTER 6

AN ANALYSIS OF EXISTING SOFTWARE ESTIMATION TECHNIQUES

This chapter describes the series of techniques that are implemented in the hybrid tool. Several programs, with Graphic User Interfaces (GUI), have been created to calculate these metrics. These programs can be used as a stand-alone system to determine the effort needed to complete a software product by considering its attributes. The chapter concludes with a discussion of the metrics, which have been implemented for estimation.

6.1 CHARACTERISTICS OF THE SOFTWARE METRIC SYSTEMS

The study of software metrics covers all the basic concepts which are related for the implementation of the effort calculation of software, Galorath et al (2006). An estimation tool has been proposed which calculates the effort needed to perform the following activities of work: Requirements, Design, Coding and Testing. It has been extended by combining quality attributes with function point metrics with risk assessment strategy.

Detailed Study analyses have been done on the following:

- Specificity and completeness of Requirements
- Use Case Metrics
- Function Point Metrics, LOC Metrics, COCOMO Model Metrics
• Function Point Quality metrics
• Analysis of all the above metrics
• Effort and risk analysis estimation

6.2 REQUIREMENTS

This section presents a detailed overview of the software metrics which is used to measure the specificity and measure of the requirements, Brian Nixon (1998). This metrics is used during the inception phase of the unified process model and also this metric can be extended to the elaboration phase.

The quality of software requirements specification can be measured by determining two values. The first value is the specificity of the requirement. ‘Q₁’ refers to the specificity of the requirements. ‘Q₂’ refers to the completeness of the requirements. ‘Q₃’ refers to the non-functional requirements.

To determine the specificity (lack of ambiguity) of requirements equation 6.1 has been used.

\[
Q_1 = \frac{nu_i}{n_r}
\]  \hspace{1cm} (6.1)

Where:
\[
Q_1 = \text{Specificity of requirements.}
\]
\[
u_i = \text{number of requirements for which all reviewers had identical interpretations.}
\]
\[
n_r = \text{Total number of requirements, it is given by:}
\]
\[
f = \text{number of functional requirements}
\]
nn_f = number of non-functional requirements (n_r = n_f + nn_f)

The optimal value of Q_1 is one, if the requirements are not ambiguous, then the value should be closer to one. The lower the value, the more ambiguous the requirements are; and this will bring problems in the later phases. Therefore, the requirements need to be constantly reviewed and discussed by all team members until they all understand the requirements and agree to adhere to these requirements. Figure 6.1 clearly shows how the requirement metrics have been calculated.

![Figure 6.1 Requirement Metrics](image)

The completeness of functional requirements is given by equation 6.2.
Where

\[ Q_2 = \frac{n_u}{n_i \times n_s} \]  \hspace{1cm} (6.2)

Where

\( Q_2 = \) completeness of functional requirements only. This ratio measures the percentage of necessary functions that have been specified for a system, but doesn’t address nonfunctional requirements.

\( n_u = \) number of unique functional requirements,

\( n_i = \) number of inputs (all data inputs) defined or implied by the specification document.

\( n_s = \) number of scenarios and states in the specification.

The degree to which requirements have been validated have been considered.

\[ Q_3 = \frac{n_c}{n_c + nn_v} \]  \hspace{1cm} (6.3)

Where:

\( Q_3 = \) degree to which the requirements have been validated.

\( n_c = \) number of requirements that have been validated as corrected.

\( nn_v = \) number or requirements that have not yet been validated
6.3 DESIGN

Analysis of Function Points as a design software metrics which can be used during the design phase has been done. This metric was developed by Albrecht. It focuses on measuring the functionality of the software product according to the following parameters: user inputs, user outputs, user inquiries number of files and the number of external interfaces. Figure 6.2 depicts the calculation of unadjusted function point’s value.

![Software Metrics](image)

**Figure 6.2 Unadjusted Function Points**

Once the parameters are counted, a complexity (simple, average and complex) value is associated to each parameter. Figure 6.2 shows the implementation provided for unadjusted FP. A complexity adjustment value is added to the previous count. This value is obtained from the response to 14 questions related to reliability of the software product. The implementation of the reliability questions is shown in Figure 6.3 and
Figure 6.4. Figure 6.5 shows how the adjusted function point’s values are calculated. Equation 6.4 presents the estimation of Function Points.

\[ FP = \text{count total} \times [0.065 + 0.01 \times \sum F_i] \]  

(6.4)

Where:

- \( FP \) = Total number of adjusted function points
- \( \text{count total} \) = the sum of all user inputs, outputs, inquiries, files and external interfaces to which have been applied the weighting factor.
- \( F_i \) = a complexity adjustment value from the response to the 14 reliability questions.
Figure 6.4 FP Reliability Questions Part II

The FP metric is difficult to use because one must identify all of the parameters of the software product. Somehow, this is subjective, and different organizations could interpret the definitions differently. Moreover, interpretations could be different from one project to another in the same organization, and this could be different from one software release to another.
6.4 CODING

This section presents the software metrics appropriate to use during the implementation phase of the software design. The metrics presented in this section are: Defect Metrics and Lines of Code (LOC).

6.4.1 Estimation of Number of Defects

The following formula helps to get the estimation of the number of defects based on the Function Points of the system and is given below:

\[
\text{Potential Number of Defects} = \text{FP}^{1.25}
\]  

(6.5)

Where:

FP = Function Points
Figure 6.6 shows the calculation of potential number of defects in the software.

![Software Metrics]

Figure 6.6 Estimating the Potential Number of Defects

6.4.2 Lines of Code

The Lines of Code (LOC) metric specifies the number of lines that the code has. The comments and blank lines are ignored during this measurement. The LOC metric is often presented on thousands of lines of code (KLOC) or source lines of code (SLOC).

LOC is often used during the testing and maintenance phases, not only to specify the size of the software product but also it is used in conjunction with other metrics to analyze other aspects of its quality and cost. Figure 6.7 shows the implementation of LOC.
Several LOC tools are enhanced to recognize the number of lines of code that have been modified or deleted from one version to another. Usually, modified lines of code are taken into account to verify software quality, comparing the number of defects found to the modified lines of code.

Many LOC tools are available to recognize the lines of code generated by software tools. Often these lines of code are not taken into account in final count from the quality point of view since they tend to overflow the number. However, those lines of code are taken into account from the developer’s performance measurement point of view.

Figure 6.7 Lines of Code (LOC)
6.5 EXISTING ESTIMATION TECHNIQUES

This section presents software metrics that could help to know the quality of the system during all phases of the project in regards to the Effort, Time and Cost of it.

6.5.1 COCOMO II

COCOMO II is the new version of the Constructive Cost Model for software effort, cost and schedule estimation (COCOMO). COCOMO II adds the capability of estimating the cost of business software, OO software, software created via evolutionary or spiral model and other new trends (for example commercial off the shelf applications –COTS).

![Software Metrics](image)

*Figure 6.8 COCOMO II Effort*
The market sectors, which are benefited the most from COCOMO II, are: 1) the ‘Application Generators’ (Microsoft, Lotus, Novell, Borland, etc.), 2) the ‘Application Composition’ sector that focuses on developing applications which are too specific to be handled by prepackaged solutions, and 3) the ‘Systems Integration’ sector, which works on large scale, highly embedded systems (for example: EDS and Andersen Consulting, etc.). The COCOMO II model for the Application Composition sector is based on Object Points. Object Points count the number of screens and then it will report back on the application a weighted factor which can be of three levels of complexity: simple, medium and difficult. Figure 6.8 shows this implementation.

The COCOMO II model for the Application Generator and System Integrator sectors is based on the Application Composition model (for the early prototyping phase), and on the Early Design (analysis of different architectures) model and finally on the Post-Architecture model (development and maintenance phase).

In COCOMO II metric, the effort is expressed in Person Months (PM). Person Months (also known as man-months) is a measure that determines the amount of time one person spends working in the software development project for a month.

The number of PM is different from the time the project will take to complete. For example, a project may be estimated to have 10 PM but have a schedule of two months. Equation 6.6 defines the COCOMOII effort metric.
\[ E = A \times (KLOC)^B \]  \hspace{1cm} (6.6)

Where:

- \( E \) = Effort
- \( A \) = A constant with value 2.45
- \( KLOC \) = Thousands of Lines of Code
- \( B \) = factor of economies or diseconomies (costs increases), given by equation 6.7.
  \[ B = 1.01 + (0.01) \times (\sum S_f) \]  \hspace{1cm} (6.7)

Where:

- \( S_f \) is the weight of the following Scale Factors (SF): Precedentedness (how new is the program to the development group), Development Flexibility, Architecture/Risk Resolution, Team Cohesion, and Process Maturity (CMM KPA’s).

### 6.5.2 Statistical Model

Another approach to determine software effort has been provided by C. E. Walston and C.P. Felix of IBM. They used equation 6.8 to define the effort in its statistical model.

\[ E = a \times L^b \]  \hspace{1cm} (6.8)

Where:

- \( E \) = Effort
- \( a \) = a constant with value: 5.2
- \( L \) = Length in KLOC
- \( b \) = a constant with value: 0.91
Walston and Felix also calculated the Nominal programming productivity in LOC per person-month as defined by equation 6.13.

\[ P = \frac{L}{E} \]  \hspace{2cm} (6.9)

Where

\begin{align*}
P &= \text{Nominal programming productivity in LOC per person-month} \\
L &= \text{Length in KLOC} \\
E &= \text{Effort from equation 6.10}
\end{align*}

The implementation of this metric is shown in Figure 6.9

![Software Metrics](image)

**Figure 6.9 Statistical Model Effort**
6.5.3 Halstead Metric for Effort

Halstead has proposed an effort metric to determine the effort and time which is given by equations (6.10), (6.11) and (6.12):

\[ E = \frac{V}{L} \]  \hspace{1cm} (6.10)

Where:

\[ E = \text{Effort in mental discriminations needed to implement the program.} \]

\[ V = N \log_2 n \]

Where:

\[ V = \text{program volume} \]
\[ N = \text{program length} \]
\[ n = \text{program vocabulary} \]

\[ L = \frac{V^*}{V} \]  \hspace{1cm} (6.11)

Where

\[ L = \text{program Level} \]

\[ V^* = \text{potential volume given by equation 6.11.} \]

\[ V^* = (N_1^* + N_2^*) \log_2 (n_1^* + n_2^*) \]  \hspace{1cm} (6.12)

Halstead’s metrics include one of the most complete sets of mathematical combinations of software metrics and even though most people recognize that they are interesting; not many companies follow them due to the reason that it is hard to convince people about change solely on the basis of these numbers.