Complexity Metrics
For
Small Business Software Applications
In
ACCESS Environment

By
Sana F. Abiad
B.Sc., American University of Beirut

PROJECT

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at the Lebanese American University
June 1999

[redacted]

Dr. Nash'at Mansour (Advisor)
Associate Professor of Computer Science
Lebanese American University

[redacted]

Dr. Ramzi Haraty
Assistant Professor of Computer Science
Lebanese American University

List of Tables

Acknowledgments

I would like to thank Dr. Mansour, for encouraging and supporting me the past four years, and Dr. Haraty for his help.

I would like to thank also Dr. Kabani for his advice and support through academic years spent at LAU.

Finally, I would like to thank my parents as well as my sister Sawsan for their encouragement, and especially my husband Hassan, not forgetting my son Ahmad.

Abstract

Without measurements (or metrics) it is impossible to detect problems early in the software process. 'Software Metrics' can serve as an early warning system for potential problems. In this project, we propose complexity metrics for small business applications developed using MS Access. These metrics can be used for estimating the effort and duration of a project's implementation phase. The proposed metrics are base on Function Point metrics with certain modifications that make them applicable to relational database languages.

Chapter 1

Introduction

'Software Metrics' is an important subject area within the more general domain of software engineering. While software engineering refers to the collection of activities concerned with turning software development and management into a discipline engineering activity, software metrics refer to anything within software engineering which has a quantifiable feel to it [Fenton 1993]. This includes measuring, predicting and improving costs, productivity, quality and complexity of software products.

Software metrics are usually classified into two types: product metrics and process metrics. Product metrics include the size of the product, its logical structure complexity, the function of the product, or a combinations of them. Process metrics include cost and effort estimation and quality control and assurance of a software project.

In this project, we concentrate on structural and complexity metrics. Structure measures and complexity metrics are often based on control flow and vocabulary [McCabe, 1976] [Halstead, 1977]. These metrics cannot be meaningfully applied to relational database languages since there is no flow graphs in such languages.

Many metrics have been proposed for large-scale procedural or object-oriented applications. To the best of our knowledge, no metrics have been reported for

small or medium scale business relational database applications, this why we started searching for such metrics. In this project, we propose new aggregate database metrics for small business applications developed in ACCESS environment It computes the effort and duration in Working Days, based on the assumption that this effort includes only the implementation work done on ACCESS. That is, it includes coding and testing.

This report is organized as follows: Chapter 2 is a survey of a number of complexity and structural metrics on procedural and object-oriented languages. In Chapter 3 we propose a number of metrics that are considered to be the base attributes in calculating cost and effort estimation of projects developed in Access. Chapter 4 proposes a new aggregate database metric that could be applied to small business applications developed in MS ACCESS. In Chapter 5 we apply the aggregate database metric on a number of ACCESS applications. Chapter 6 is the conclusion.

Chapter 2

Survey of Complexity and Structural Metrics

In this chapter, we make a survey of a number of complexity and structural metrics. Most of the structural metrics are based on procedural languages. However, complexity metrics are based on both, procedural and object-oriented languages. There are two metrics that estimate the effort and duration of a project, and they are independent of any language

2.1. Cyclomatic Number

McCabe [1976] has described a complexity measure approach for measuring and controlling the number of paths through a program. This complexity measure is defined in terms of basic paths that when taken into combination will generate every possible path.

Definition: The cyclomatic number V(G) of a strongly connected graph G .

$$V(G) = e - n + 2p \qquad [2.1]$$

e = edges

n = vertices

p = connected components .

The application of the above theorem will be made as follows: Given a program, we will associate with it a directed graph that has unique entry and exit nodes. Each node in the graph corresponds to a block of code in the program where the flow is sequential, and the arcs correspond to branches taken in the program.

The overall strategy will be to measure the complexity of a program by computing the number of linearly independent paths V(G), control the " size" of program by setting an upper limit to V(G) and use the cyclomatic complexity as the basis for testing methodology.

Several properties of cyclomatic complexity are stated below:

1) $V(G) \geq 1$

2) V(G) is the maximum number of linearly independent paths in G ; it the size of a basis set.

3) Inserting or deleting functional statements to G does not affect V(G) .

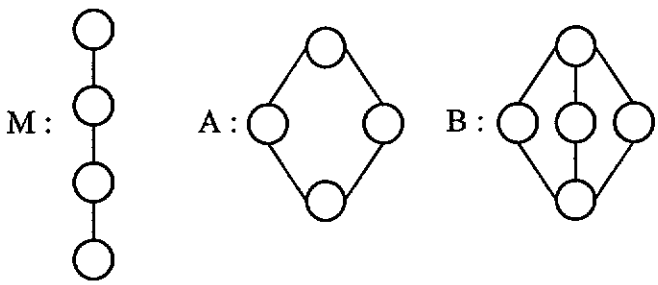
4) G Has only one path if and only if $V(G) = 1$

5) Inserting a new edge in G increases V(G) by unity

6) V(G) depends only on the decision structure of G

This complexity measure will control the structure of the program by counting the branches (i.e. if statement, loops, case,...).One application of this complexity measure was to set the upper cyclomatic complexity to 10 . When the complexity exceeded 10 and they had to either recognize and modularize subfunctions or redo the software. The intention was to keep the "size" of the modules manageable and allow for testing all the independent paths.

The role of *p* in the complexity calculation v = e - n + 2p will now be explained. Imagine a main program M and two called subroutines A and B having a control structure shown below:



$V(M \cup A \cup B) = V(M)+V(A)+V(B)$

Let us denote the total graph above with three connected components as $M \cup A \cup B$. Now, since p = 3 we calculate complexity as:

$V(M \cup A \cup B) = e - n + 2p = 13 - 13 + 2*3 = 6$

This method with $p \neq 1$ can be used to calculate the complexity of a collection of programs, particularly a hierarchical nest of subroutines. Notice however that $V(M \cup A \cup B) = V(M) + V(A) + V(B)$.

2.2. Heuristics for Computing Attribute Value of C ++ Program Complexity Metrics

In these metrics, we examine program complexity from three dimensional view points in object-oriented paradigm: Syntax dimension, inheritance dimension and interaction dimension [MiKim et al., 1996]

Any system (language, tool and methodology) is object-oriented if it supports all of the following four properties:

1) Data and procedures are combined in entities known as objects

2) Messages are used to communicate with these objects

3) Similar objects are grouped into classes

4) Data and procedures can be inherited within a hierarchy of classes

2.2.1. Attributes for syntax complexity

$$Sx(p) = f_1(IMC, NOM, NOCL, LCOM, UOC) \quad [2.2]$$

1) $IMC(M_i)$: Degree of internal method complexity of a method M_i

2) $NOM(C_i)$: Number of methods in a class C_i

3) NOCL(P) : Number of classes in a program P

4) $LCOM(C_I)$: Degree of lackness of cohesion in a class C_i

5) UOCL(P) : Ratio of used classes to defined classes in a program P

$0< UOC(P)\leq 1$

These five attributes correspond to the volume of program and effort of coding.

2.2.2. Attributes of Inheritance complexity

To evaluate the inheritance aspect of the program, we introduce the following attributes, which correspond to the degree of reuse by inheritance

$(IH(P) = f_2(DIT, NODC, NODA, DOR, NOD)$

1) $DIT(C_i)$: Depth of inheritance tree for class C_i

2) $NODC(C_i)$: Number of children of a class C_i

3) $NODA(C_I)$: Number of all inheriting ancestors of a class C_i

4) $DOR\ (C_I)$: Degree of reuse by inheritance ($0 \leq DOR(C_I) < 1$) for each C_I

5) NOD (P) : Number of disjoint inheritance trees in a program P.

2.2.3. Attributes for interaction complexity

To evaluate the inheritance aspect of the program, we introduce five attributes that correspond to the degree of coupling. Especially CBI reflects " is kind of " relationship and UCL reflects " is part of relationship"

$ITP(P) = f_3(CBI, RFC, UCL, VOD, MPC)$ **[2.3]**

1) $CBI(C_I)$: Degree of coupling of inheritance in a class C_i

2) $RFC(C_i)$: Degree of response in a class C_i

3) UCL(C_i) : Number of classes used in a class C_i except for ancestors and children.

4) VOD(C_i) : Number of validation of the law of Demeter in a class C_i

5) MPC(C_i) : Number of send statements in a class C_i

2.2.4. Heuristics for attributes

Among fifteen attributes, NOM, NOCL, DIT, NODC, NODA and NOD can be computed based on the definitions.

To compute the values of the attributes

1) IMC(M_i) : $IMC(M_i) = \{(N_1 + N_2) * \log (n_1 + n_2)\} / (n_1 /2 + n_2 /2)$ **[2.4]**

n_1 : number of unique operators in the method M_i

n_2 : number of unique operands in the method M_i

N_1 : total number of appearances of operators

N_2 : total number of appearances of operands

2) UOC(P) : $UOC (P) = n_{uc} / n_{dc}$ **[2.5]**

n_{dc} : number of classes defined in a program P.

n_{uc} : total number of classes used in program P.

3) DOR(C_i) : for each class C_i, $DOR(C_i) = \sum_{k=1}^{r(Ci)} K/(t + t_r)$ **[2.6]**

t : total number of classes in a program P

$r(c_i)$: the reused number of each class (C_i) in the program P

t_r : total sum of all $r(c_i)$'s in the program P

4) CBI(C_i) : it is a degree of coupling of inheritance and indicates the degree how far the implementation of the class is dependent on the other classes inherited.

For each class C_i, $CBI(c_i) = S(c_i) * ih(c_i)$ **[2.7]**

$S(c_i)$: the sum of all IMC (M_j)'s in class C_i

$ih(c_i)$: number of classes which inherit class C_i

5) UCL(C_i) : it is computed for each class C_i in program P . Interaction among classes via message passing is performed by four kinds of operations in C_{++} . Let S denote a set of classes which don't have " is-a " relationship with class C_i

$UCL(C_i) = u_1 + u_2 + u_3 + u_4$ **[2.8]**

u_1 : compute the number of such classes that are in S and are defined as date member of class C_i .

u_2 : compute the number of such classes that are in S and are defined as date member of method M_j in class C_i .

u_3 : compute the number of such classes that are in S and are defined as pointer in class C_i .

u_4 : compute the number of such classes that are in S and are defined as pointer in method M_j .

6) MPC(C_i) : it is computed for each class C_i in program P. MPC is the number of statements which are sent from other classes

$$MPC(C_i) = \sum (s(m_{ij}) + r(m_{ij})) * v(m_{ij}) \quad \textbf{[2.9]}$$

Let M (C_i) be a set of member functions which are used in class C_i and are called from other classes.

$s(m_{ij})$: the number of "j" in each member function m_{ij} of $M(C_i)$

$r(m_{ij})$: the number of reserved words (if, switch, case, default, for, while, do-while) in each m_{ij} .

$v(m_{ij})$: the number of m_{ij} used by other classes .

2.3. Convexity and Independence in Software Metric Theory

In a cubic flowgraph F of order o, there are 3n+1 edges among 2n+1 nodes or vertices, one distinguished and named $X = X_F$ [Prather, 1996], the others:

a) Decision nodes have indegree = 1 and out degree = 2

b) Junction nodes have indegree = 2 and out degree = 1

c) The distinguished node has indegree = 1 and out degree = 1

Note: The prime (cubic) flowgraphs are those that cannot be built up non-trivially by sequencing or nesting. These have recently been characterized as the cubic flowgraphs that are triply connected without the removal of at least three edges.

Theorem 1 (Prather-Guilien)

Each cubic flowgraph has a unique decomposition

$F = S (P_1[F_{11} , F_{12} , ...], ..., P_r [P_{r1} , P_{r2}, ...])$

into a sequence of prime with maximal nested sub-flowgraphs F_{ij}

Theorem 2 (Fenton-Whitty)

Every prime flowgraph of order n is obtained by " graphing" a new decision-to-junction edge onto a prime of order n-1 . As a result, the prime cubic flowgraph can be effectively enumerated as an infinite union.

$$S = \bigcup_{n=0}^{\infty} S_n$$

$S = \{C\}$

$S_1 = \{ D_1 , D_2 \}$

$S_2 = \{ E_1 , E_2 , E_3 , E_4 , E_5 , E_6 \}$

S_n is the subclass of primes of order n

D_1 Dijkstra flowgraph for structured programming construct of selection

D_2 Dijkstra flowgraph for structured programming of repetitive

C is the circuit (of order zero)

2.3.1. Classification of structural software metrics

To refer to an isomorphism invariant function

m : cubic flowgraph $\longrightarrow$ natural numbers .

is a hierarchical (software) metric. j

(a) S-induction : $mS(x_1 , x_2 , ... , x_r) = m_s (mx_1 , mx_2 , ... , mx_r)$

(b) P-induction : $mP(y_1 , y_2 , ... , y_r) = m_p [e_j \longrightarrow my_j)$

For a certain numerical function and graphical algorithm m_s and m_P , we present a series of six illustrative examples, designed to exhibit the full range of applicability of the extended axiomatic system.

P : McCabe's cyclomatic measure

$$P_s (x_1 , x_2 , ... , x_r) = 1 + \sum_{x_i \geq 1} (x_i - 1) \qquad \textbf{[2.10]}$$

$$P_p [e_j \longrightarrow y_j] = 1 + op + \sum (y_j - 1) \qquad \textbf{[2.11]}$$

$y_j \geq 1$

μ : Prather's mu measure

$$\mu_s(x_1, x_2, ..., x_r) = \sum_{1} x_i \quad \textbf{[2.12]}$$

$$\mu_p[e_j \longrightarrow y_j] = 2^{op} \max\{y_j\} \quad \textbf{[2.13]}$$

λ : lines of (pseudo) code measure

$$\lambda_s(x_1, x_2, ..., x_r) = \sum_{1} x_i \quad \textbf{[2.14]}$$

$$\lambda_p[e_j \longrightarrow y_j] = 1 + \sum_{1} y_j \quad \textbf{[2.15]}$$

δ : depth of-nesting metric

$$\delta_s(x_1, x_2, ..., x_r) = \max\{x_i\} \quad \textbf{[2.16]}$$

$$\delta_p[e_j \longrightarrow y_j] = 1 + \max\{y_j\} \quad \textbf{[2.17]}$$

σ : localized size measure

$$\sigma_s(x_1, x_2, ..., x_r) = \max\{x_i\} \quad \textbf{[2.18]}$$

$$\sigma_p[\, e_j \longrightarrow y_j \,] = \max\{y_j\}\ [\, 1 + oP, y_j]$$ **[2.19]**

η : number of (simple)- path metric

$$\eta_s\,(x_1\,, x_2\,, \ldots..,\, x_r\,) = \prod_i x_i$$ **[2.20]**

$$\eta_p\,[\, e_j \longrightarrow y_j\,] = \sum \prod_I y_j\,;^{bij}$$ **[2.21]**

where bij =1 , je_j is on (simple) path Pi (else 0)

2.3.2. Theory of convexity

Our goal is to find a framework that will allow for combinations of (generalized) hierarchical metrics, in some useful yet mathematically sound environment where by individual researchers can give weight to various software engineering attributes in designing a metric that best serves their needs. The notion of a pseudo-metric **f** to be a real-valued function on flowgraph that comes about by forming a weighted sum.

$f = \sum_{I=1}^{k} a_i m_i$ where each m_i is a generalized hierarchical metric **[2.22]**

the weights a_i are required to satisfy the properties $a_i \geq 0$ and

$$\sum_{I=1}^{k} a_i = 0$$

For each individual flowgraph G this means that f is evaluated as

$$f(G) = \sum_{I=1}^{k} a_i m_i (G) \qquad \textbf{[2.23]}$$

If f_1 , f_2 , ..., f_k are pseudo-metric, we can define a convex combination of f_1, f_2 , ..,f_k as

$$f = \sum_{I=1}^{k} a_i f_i \quad (\text{with } a_i \geq 0 \text{ and } \sum a_i = 1)$$

and again we compute

$$f(G) = \sum_{I=1}^{k} a_i f_i(G) \text{ , for each individual flowgraph G.}$$

2.4. Measuring Program Structure With Inter-Module Metrics

A structure metric could help locate poorly structured parts in a project and serve as an indicator when and where a restructuring effort may be necessary [Amman et al., 1996]. Moreover, structure measures and complexity metrics are often based on control flow or vocabulary.

However in the object-oriented paradim, control flow metrics can not meaningfully be applied to measure the structure. Therefore, the four structure metrics that we are going to discuss are based on the principle of vocabulary hiding and generally measure a form of cohesion. These metrics use a different approach in which all distinct identifiers are counted including type names, exception names and field names as well as procedure and variable names. These metrics differ from

existing structure metrics since they are not based on flowgraphs, but on a comparison between the sets of visible and used vocabulary. At last, these metrics are aimed at programming languages with explicit module constructs and strict import/export conventions in the form of interfaces and import declarations.

Definitions

Given a project P , let

$Int(P)$ = set of interfaces

$Imp(P)$ = set of implementations

$Mod(P) = Int(P) \cup Imp(P)$ (set of modules)

For any $i_i \in Int(P)$, $m_j \in Mod(P)$, let

E_i = set of entities exported by i_i

V_j = set of entities visible in m_j (including local entities)

U_j = set of entities used in n_j (including local entities)

LV_j = set of local entities visible in m_j

LU_j = set of local entities used in m_j

Define

$V_{ij} = E_i \cup V_j$ (entities exported by i_i , visible in m_j)

$U_{ij} = E_i \cap V_j$ (entities exported by i_i , used in m_j)

$I_j = \{i_i \in Int(P) \mid V_{ij} \neq 0 \}$ (interfaces imported by m_j)

$M_i = \{m_j \in Mod(P) \mid V_{ij} \neq 0 \}$ (clients for i_i)

For any $m_j \in Mod(P)$, $e \in V_j$,let

S_j = set of procedures in m_j

Us_{ej} = set of procedures in m_j in which entity e is used .

For any $S_k \in S_j$, i_i e Int(P) , let

SV_{jk} = set of entities used in procedures S_k in m_j

SV_{jk} = set of entities visible in procedures S_k in m_j

Define

$SU_{ijk} = E_i \cap SV_{jk}$

$LV_{jk} = LV_i \cap SV_{jk}$

$LU_{jk} = LU_j \cap SU_{jk}$

2.4.1. Usage / Visibility Metric

It measures what proportion of the entities imported from one or more interfaces are actually used in the target module. The usage/visibility metric is based on the idea that the more external entities that are visible from given point in a program, the more complex the program is. UVR_{ij} (usage visibility ratio) value

measure the proportion of the external entities which are used in the target module with respect to only one external module.

$$\gamma m_j \in Mod(P), \gamma i_i \in Int(P) \text{ if } V_{ij} \neq \varnothing$$

$$UVR_{ij} = \frac{|U_{ij}|}{|V_{ij}|} \qquad [2.24]$$

Normally, we would like to know the usage/visibility ratio of a module with respect to all imported interfaces. The simple measure is an unweighted average as shown below.

$$\gamma m_j \in Mod(P), \text{ if } I_j \neq \varnothing$$

$$SUVR_j = \frac{\sum_{i_i \in I_j} UVR_j}{|I_j|} \cong \frac{\sum_{i_i \in I_j} \frac{|U_{ij}|}{|V_{ij}|}}{|I_j|} \qquad [2.25]$$

However, it may be often more appropriate to weigh imported interfaces depending on their size and the usage visibility for one module is best represented by

$$WUVR_j = \bigcup_{i_i \in I_j} U_{ij} / \bigcup_{i_i \in I_j} V_{ij} \qquad [2.26]$$

To calculate the usage/visibility ratio for a project consisting of a set of modules:

$$UVR = \frac{\sum_{m_j \in Mod(P)} WUVR_j}{|Mod(P)|} \qquad [2.27]$$

2.4.2. External Cohesion Metric

The external cohesion metric is calculated for "server" modules, i.e. interfaces. It measures what proportion of the visible entities in the target interface is actually used in clients which import the target interface. Thus, the hypothesis is that if an interface is well designed, a large portion of it will be used if it is imported .

$$\gamma i_i \in Int(P), \text{ if } M_i \neq \varnothing$$

$$EC_i = \frac{\sum_{m_j \in M_i} UVR_{ij}}{|M_i|} \qquad [2.28]$$

We compute an unweighted aggregate measure for the external cohesion. It takes into account all interfaces of a target project

$$\text{For any P, f } (Int(P) \neq \varnothing) \wedge (\exists\, i_i \in Int(P) \mid M_i \neq \varnothing)$$

$$EC = \frac{\sum_{i_i \in Int(P)} EC_i}{|Int(P)|} \qquad [2.29]$$

2.4.3. Module Cohesion Metric

The module cohesion measures the ratio of procedures in which an imported entity is used compared to the total number of procedures in a module, i.e. it measures the extent to which entities from imported interfaces are used in different parts of the target module. The hypothesis is that if any entity is used in most parts or procedures of the target modules, the target module is suitable structure for the imported entity to be used in.

The module cohesion with respect to one of the imported interface

$$\gamma m_j \in Mod(P),\ \gamma i_i \in Int(P), \text{ if } V_{ij} \neq \varnothing$$

$$MC_{ij} = \frac{\sum_{e \in U_{ij}} \frac{|US_{ej}|}{|S_j|}}{|V_{ij}|} \qquad [2.30]$$

which is equivalent to

$$MC_{ij} = \frac{\sum_{s_k \in S_j} \frac{|SU_{ijk}|}{S_j}}{|U_{ij}|}$$

to compute the module cohesion with respect to all imported interfaces

$\gamma m_j \in Mod(P)$, if $I_j \neq \varnothing$

$$SMC_j = \frac{\sum_{i_i \in I_i} MC_{ij}}{|I_j|} \qquad [2.31]$$

$$MC_j = \frac{\sum_{s_k \in S_j} \frac{|\bigcup_{i_i \in Int(p)} SU_{ijk}|}{S_j}}{|\bigcup_{i_i \in I_j} U_{ij}|}$$

The module cohesion can also be easily aggregated for a multi-module project as an unweighted average.

$$MC = \sum_{m_j \in Mod(P)} \frac{MC_j}{|Mod(P)|}$$

2.4.4. Internal Cohesion Metric

The internal cohesion measures the proportion of the number of procedures in a given module in which a local target entity is used compared to the total numbers of procedures in the target module (.i.e. if a global item is used, it ought to be used in a large number of procedures).

$\gamma m_j \in Mod(P)$, if $LU_j \neq \varnothing$

$$MC_{ij} = \frac{\sum_{sk \in S_j} \frac{|LU_{jk}|}{|LV_{jk}|}}{|S_j|} \qquad [2.32]$$

The internal cohesion for an entire project can be computed as a simple average For any P,

$$\text{if } (\text{Mod}(P) \neq \varnothing) \wedge (\exists m_j \in \text{Mod}(P) \mid LU_j \neq \varnothing)$$

$$IC = \frac{\sum_{m_i \in \text{Mod}(P)} IC_j}{|\text{Mod}(P)|} \qquad [2.33]$$

2.5. Cocomo Model

The constructive Cost Model (cocomo) was developed by Barry Boehm and published in 1981 [Boehm, 1981] . He developed an easy to understand model that predicts the effort and duration of a project based on input relating to the size of the resulting systems and a number of cost drivers that Boehm believes affect productively.

2.5.1. Basic Cocomo Model

The basic Cocomo Model provides a quick, early, rough estimates of software effort and costs but its accuracy is limited because of its lack of factors which have significant effect on the amount of effort required to develop a software project. The basic Cocomo model is applicable to the small and to the medium size projects developed in a familiar software development environment.

There are certain definition which describes the basic Cocomo model :

- The primary cost driver is the number of delivered source instructions (DSI) developed by the project . (i.e. any line containing source statements, data, format statements and job control languages)
- The period covered by Cocomo cost estimates start at the beginning of the product design phase and ends at the end of the integration and test phase.

- A Cocomo man-month (MM) consist of 152 hours of working time.

- Cocomo estimates assume that the project will enjoy good management and that the requirement specification will not change after the requirement phase.

There are three main modes of software development which yields different cost estimates. Another point that may be considered is that a large software project may contain several subprojects operating in different modes. These modes are :

2.5.1.1. The Organic Mode

In the organic mode, relatively small software teams develop software in a highly familiar environment. Most people connected with the project have extensive experience in working with related systems within the organization and have a thorough understanding of how the system under development will contribute to the organization objectives.

Organic mode projects are relatively small in size (not more than 50 KDSI)

The effort and schedule equations are :

$$MM = 2.4 (KDSI)^{1.05} \quad \textbf{[2.34]}$$

$$TDEV = 2.5 (MM)^{0.38} \quad \textbf{[2.35]}$$

2.5.1.2. The Semidetached Mode

The semidetached mode of software development represents an intermediate stage between the organic and the embedded modes. This means an intermediate level of the project characteristics or a mixture of the organic and the embedded characteristics. The size of a semidetached mode product usually extends to 300 KDSI.

The effort and schedule equations are :

$$MM = 3.0 (KDSI)^{1.12} \quad \textbf{[2.36]}$$

$$TDEV = 2.5 (MM)^{0.35} \quad \textbf{[2.37]}$$

2.5.1.3. The Embedded Mode

An embedded mode software project has to operate within tight constraints, it must operate within a strongly coupled complex of hardware, software, regulations and operational procedures.

The effort and schedule equations are :

$$MM = 3.6 (KDSI)^{1.2} \quad \textbf{[2.38]}$$

$$TDEV = 2.5 (MM)^{0.32} \quad \textbf{[2.39]}$$

2.5.2. The Intermediate Cocomo Model

In addition to the single predictor variable (size in delivered source instructions) used by the Basic Cocomo model, the Intermediate incorporates an additional 15 predictor variables (see Table 2.1)

An intermediate Cocomo software development effort estimate begins by generating a nominal effort estimate. Then this nominal effort is adjust by applying multipliers determined from the project's rating with respect to the other 15 cost driver attributes. The nominal effort is multiplied by the product of all 15 effort multipliers to obtain the final effort.

The intermediate Cocomo scale factor for the three software development modes are :

Development Mode	Nominal effort equation	Schedule equation
Organic Mode	$(MM)_{nom} = 3.2\ (KDSI)^{1.05}$	$TDEV = 2.5\ (MM)^{0.38}$
Semidetached Mode	$(MM)_{nom} = 3.0\ (KDSI)^{1.12}$	$TDEV = 2.5\ (MM)^{0.35}$

Table 2.1 Intermediate COCOMO Software Development Effort Multipliers

	Rating					
Cost drivers	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.15	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual Machine volatility		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.05	
Personal attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Programmer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

2.6. Function Point Analysis

Function Point Analysis sizes an application from an end-user perspective, it is totally independent of all language considerations [Dreger, 1989]. FPA actually and reliably evaluates :

- The business value of a system to user
- Project size, cost and development time.
- MIS programmer productivity and quality.
- Maintenance modification and customization effort.
- Feasibility of in-house development.

The formulas are as follow [Dredereg, 1989] :

FP= (total unadjusted function points) * (0.65 + (0.01*total degree of influence)**[2.40]**

The total unadjusted function points expressed in the equation will be discussed in Business Function and the total degree of influence will be discussed in adjustment factors.

2.6.1. Business Functions

Business functions made available to the end user, are identified and then organized into groups. When designing a new system it should be analyzed according

to the order of the function as we will see below, then each business function is classified by it's level of complexity (simple, average or complex) Function points are counted as they are logically not physically implemented. Moreover, they may be overlapping .

The business functions are:

2.6.1.1. Outputs

Outputs are items of business information processed by the computer for the end user. Each unique data or control output procedurally generated that leaves the application boundary should be counted . An output is considered unique if :

1- It has a different format or

2- It has the same format as another output but requires different processing logic.

After counting the number of outputs, they should be classified according to the number of files referenced or accessed and the number of data items referenced .

2.6.1.2. Inputs

Inputs are items of business data sent by the user to the computer for processing and to add, change or delete something. Each unique user data, or control input that enters the application boundary and also updates a logical internal file, data set, table or independent data item should be counted. An input is considered unique if:

1- It has a different format or

2- It has the same format as another input but requires different processing logic.

After counting the number of inputs, they should be classified according to the number of files referenced or accessed and the number of data items referenced.

2.6.1.3. Inquiries

Inquiries are looking for a specific data into a database or master file that requires an immediate response and perform no update functions. An inquiry is considered unique if:

1- It has a different format from others in either its input or output position or

2- It has the same format , both input and output, as another inquiry but requires different processing logic in either.

After counting the inquiries, they should be classified as follows: First, class if input and output individually as discussed before. Second, compare the numeric complexity value of one portion with other and select the greater.

2.6.1.4. Files

Files are data stored for an application, as logically viewed by the user. Each major logical group of user data or control information maintained entirely within the

application boundary should be counted. Logical internal files consist only of those logical data sets, files or tables which perform all capture, generation, use, maintenance and storage function entirely within the application boundary and are somehow available to users by way of outputs, inquiries, inputs and / or interfaces. (Transaction files and print files are not counted)

After counting the number of files, they should be classified according to the number of data item actually required by the application and either the number of record formats within the file or the number of logical relationship in which this file participates to meet application requirements.

2.6.1.5. Interfaces

Interfaces are data stored elsewhere by another application but used by the one under evaluation. Each major logical file or another logical group of user-approved data or control information within the application boundary that is sent to, shared with, or received from another application should be counted as one interface. Interfaces can represent any of the following situations:

1- Data or control information passed from file A to file B.

2- Date or control information is shared between file A and file B .

These interfaces will be classified by complexity according to the number of data item referenced and the number of record formats within the file or the number

of logical relationships in which the interface file participates to meet application requirements.

2.6.2. General Application Characteristics

The physical environment influences are provided by 14 adjustment factors which represent applicable elements of the physical system, they vary from one environment to another and adjust the value of the logical system accordingly.
These influences are classified by giving each a weight from 0 to 5 . These factors are:

1- Data communications

2- Distributed data or processing

3- Performance objectives

4- Heavily used configuration

5- Transaction rate

6- On-line data entry

7- End user efficiency

8- On-line update

9- Complex processing

10- Reusability.

2.7. The Theory of Software Science

The theory of software science as presented by Halstead in his 1977 monograph *"Elements of Software Science"* and summarized in [Shen et al. 1983]. A computer program is considered in software science to be a series of tokens which can be classified as either "operators" or "operands". All software science measures are functions of the counts of these tokens. The basic metrics are defined as :

η_1 = number of unique operators

η_2 = number of unique operands

N_1 = total occurrences of operators

N_2 = total occurrences of operands

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program, is defined as

$$\eta = \eta_1 + \eta_2 \qquad \textbf{[2.41]}$$

The length of the program in terms of the total number of tokens used is

$$N = N_1 + N_2 \qquad \textbf{[2.42]}$$

Another measure for the size of the program, called the volume:

$$V = N * \log_2 \eta \qquad \textbf{[2.43]}$$

The unit of measurement of volume is the common unit for size - "bits". It is the actual size of a program in a computer if a uniform binary encoding.

Since an algorithm may be implemented by many different but equivalent programs, a program that is minimal in size is said to have the potential volume V^*. Then the program level is defined by

$$L = V^* / V \qquad \textbf{[2.44]}$$

The value of L ranges between zero and one, with L = 1 representing a program written at the highest possible level (i.e. the minimum size). The inverse of the program level is termed difficulty

$$D = 1 / L \qquad \textbf{[2.45]}$$

The effort required to implement a computer program increases as the size of the program increases. It also takes more effort to implement a program at a lower level. Thus the effort in software science is defined as

$$E = V / L = D * V \qquad \textbf{[2.46]}$$

The unit of measurement of E is "elementary mental discriminations". The following are the set of hypotheses presented by Halstead.

2.7.1. Length Equation

The first hypothesis of software science is that the length of a well-structured program is a function only of the number of unique operators and operands. N is the predicted length of the program:

$$N = \eta_1 \log_2\eta_1 + \eta_2 \log_2\eta_2 \approx \eta \log_2(\eta / 2) \qquad [2.47]$$

This may be considered valid in a statistical sense. Such relationships are common in experimental science dealing with human subjects.

2.7.2. Potential Volume

As defined earlier, a program that implements an algorithm and it is minimal in size is said to have the potential volume V^*, and if the desired function is already defined in the programming language or its subroutine as a "built-in" procedure. Then, the vocabulary of this program consist of two operators and η_2^* operands. One operator is the name of the procedure, the other operator is a grouping symbol needed to separate the list of parameters from the procedure name. Thus,

$$V^* = (2 + \eta_2^*) \log_2(2 + \eta_2^*) \qquad [2.48]$$

η_2^* = number of input / output parameter for this procedure

2.7.3. Program Level (Difficulty) Estimator

An alternative formula which estimates the level is defined as:

$$\hat{L} = \frac{1}{\hat{D}} = \frac{2}{\eta_2} * \frac{\eta_2}{N_2} \qquad [2.49]$$

An intuitive argument for this formula is that programming difficulty increases if additional operators are introduced (η_2 / 2 increases) and if an operand is used repetitively (N_2 / η_2 increases).

2.7.4. Programming Time

The programming time T of a program in seconds is

$$T = E / S \qquad 5 \leq S \leq 20 \qquad [2.50]$$

E = effort and its unit of measurer is "number of elementary mental discriminations"

S = "Stroud number", it ranges between 5 and 20, and is normally set to 18

This formula can be used to estimate programming time when a given problem is solved by a single, proficient, concentrating programmer writing a single-module program.

Function_oriented systems typically have a page of code. OO methods typically have less than six lines, and there are many smaller components. Numbers in an experience database from function-oriented projects will not be directly comparable to OO results.

- No case statements:

  Well-designed OO systems don't use case statements. In fact, Smalltalk doesn't even have a case statement. This further skews OO versus traditional system complexity.

- Fewer IF statements

  The use of inheritance to have single-minded self-managing objects lead to less checking of types of objects to decide what actions to take.

Good OO designs will make the historical measurements of complexity less useful. Other measurements that have meaning for OO code are needed, at a minimum as supplements to the traditional measures.

It is proposed that a complexity measurement that looks at the number and types of message send in a method be the basic measurement of complexity. Therefore, the following assigned weights are used to compute method complexity:

• API calls	5.0
• Assignments	0.5
• Binary expressions (Smalltalk)	2.0

or arithmetic operators (C^{++})

- Keyword messages (Smalltalk) 3.0
  or messages with parameter (C^{++})
- Nested expressions 0.5
- Parameters 0.3
- Primitive calls 7.0
- Temporary variables 0.5
- Unary expressions (Smalltalk) 1.0
  or messages without parameters (C^{++})

2.9. Cognitive Complexity Model

The exact mental process involved in debugging, extending or simply understanding a section of code is chunking [Henderson-Sellers, 1996]. Chunking involves recognizing groups of statements and extracting from them information that is remembered as a single mental abstraction. These chunks are then further chunked together into larger chunks and so on, forming a multilevel, aggregated structure over several layers of abstraction. However, there is also evidence that programmers perform a type of tracing, which involves scanning quickly through a program either forward or backward, in order to identify relevant chunks. These two processes of chunking and tracing both have implications for software complexity.

Chunks may be classified as elementary or compound. Elementary chunks consist only of sequentially self-contained statements. Compound chunks are those that contain other chunks within them.

In mathematical terms, the difficulty of solving a programming inquiry focused on the ith chunk is given by the complexity C_i

$$C_i = R_i + \sum_{j \in N} C_j + \sum_{j \in N} T_j \qquad \textbf{[2.52]}$$

R_i = difficulty of understanding the immediate (ith) chunk

C_j = difficulty in moving from the original top level down to the current (ith) chunk, that difficulty occurs from both chunking and tracing.

T_j = the difficulty of tracing a particular dependency.

Chapter 3

Suggested Complexity Metrics

The main point that that we will discuss in the following chapter is: *"What metrics make a program more complex than another developed in Ms Access 97?"* The idea is to start looking at all the steps that any relational database programmer (using Access 97) will go over to finish a database project. Then, consider the steps that differ in timing from one project compared to another. These steps are considered to be the major attributes in calculating the effort and duration of a project.

The major steps that we will concentrate on are the following:

1. Setting the Tables.
2. Setting the Relationships.
3. Defining Queries.
4. Setting Indexes.
5. Preparing Forms.
6. Preparing Reports.

3.1. Setting the Tables

It includes defining the tables; fields that define these tables and if needed add some properties on tables or fields. The suggested complexity metrics at this step are the following: -

- Number of Tables
- Number of Fields / Table
- Number of Properties / Field
- Number of Table Properties

3.1.1. Number of Tables

The base of any relational database project is the tables and the relations between them. The numbers of tables will determine the size of a project (small, medium, or large). If we consider a programmer that has to set a big number of entities, and another with a fewer number of entities, the first one will eventually spend more time in performing this task.

3.1.2 Number of Table Properties

In MS Access, for each table you can set a number of properties, which helps in defining the overall picture of the table. Setting some of these properties will enforce overall restriction on the table, and hence increase complexity of setting this table. The table properties are the following:

- Description — to provide information about the table.
- Validation Rule — you can use the Validation Rule to specify a rule for entering data into a record, field or control. When data is entered that violates this Validation Rule property, you can use Validation Text to specify the message to be displayed to the user.
- Validation Text — The error message that appears when you enter a value prohibited by the Validation Rule.
- Filter — you can use the filter property to specify a subset of records to be displayed when a filter is applied to a query, form or table.
- Order By — You can use the Order By property to specify how you want to sort records in a form, query, report or table.
- Orientation — By default it is Left-To-Right.

3.1.3. Number of Fields /Table

As we have described above that there is no relational database system without tables and relations between them, and there is no tables without fields defining them.

The fields are the major component of any table, and each table should contain a number of fields. The number of fields is also a measure of the effort and time a programmer may spend on setting a certain table. As the number of fields increases, then also the time and effort of a programmer increases.

3.1.4. Number of Properties / Field

For each field defined in MS Access you can set a number of properties, some of these properties are set by default. At this stage of translating the ER diagram, you can set the properties of the fields. Field properties can be divided into two groups. The first group of properties (Format, Input Mask, Caption, Default Value) will help in defining the overall picture of the field. The second group of properties (Validation Rule, Validation Text, Required, Indexed) will enforce some restrictions on the field as a whole. Setting some of these properties will increase the complexity, the time and effort of a programmer. The properties are the following:

- Format — Format allows you to customize the way datatypes are displayed and printed. Usually either the Format or Input Mask of a field is filled out when you want to control the display of a field.
- Input Mask — it is similar to Format in that it controls the way you display data. The difference between Format and Input Mask is that the Input Mask can control how the data is entered.
- Caption — Caption functions in the exact same way as it does in the Design view of a table. The caption is the text that appears in the label of a field.

- Default Value — a value that is entered automatically in this field for new records.
- Validation Rule — you can use the Validation Rule to specify a rule for entering data into a record, field or control. When data is entered that violates this Validation Rule property, you can use Validation Text to specify the message to be displayed to the user.
- Validation Text — The error message that appears when you enter a value prohibited by the Validation Rule.
- Required — require data entry in this field? By default it is set to No.
- Allow Zero Length — Allow zero length in this field? By default it is set to No.
- Indexed — you can use it to set an index on a single field. An index speeds up queries on the indexed field as well as sorting and grouping by.

3.2. Setting the Relationships

Defining tables is an essential step in any relational database project, but there is a need to relate all the defined tables. Here comes the role of setting the relationships, which includes relating the tables, selecting the type of the relations and setting the properties of each relationship. The suggested metrics are the following:

- Number of Relationships
- Type of Relationships

- Properties of Relationships

3.2.1. Number of Relationships

The number of relationships will indicate if the tables in the ER-diagram are related or not, it is dependent on the number of tables. As the number of tables increases, then the number of relations also increases. It is an important metric because it is an indication of the complexity of the system.

3.2.2. Type of Relationships

There are three different types of relationships available in MS Access and each of a certain complexity. The type of relationship indicates how each table is associated to the related table. The ER-diagram could contain any set of relationships. The relationships are the following:

1. One-to-One: it is used when a lot of information has to be kept for each record, usually in numerous fields.
2. One-to-Many : which reflects a hierarchy of details, with a parent record that contains information about a group of items and each item having details stored in a separate table.
3. Many-to-One: which reflects a hierarchy of details.

3.2.3. Properties of Relationships

You can define properties for any relationship in your diagram. The relationship properties will enforce restrictions on the related table. These properties will increase the complexity of a system, since they should be reviewed in any change of the design of the system. The properties are the following:

- Enforce Referential Integrity — It ensures that relationships between records in related tables are valid. This prevents the user from accidentally deleting or changing related data.
- Cascade Update Related Fields — With enforce referential Integrity selected, there are several user events that need to be handled. One such event is a change of the field on the "one" side. With Cascade Related Fields selected, this change is propagated to the "many" side, so the records in the detail table are not left orphaned.
- Cascade Delete Related Fields — The other event that needs to be considered is the deletion of the parent record. When Cascade Delete Related Records is selected, the deletion of the record on the "one" side forces the deletion of the records on the "many" side.

3.3. Defining Transactions

Transactions actually perform a number of different functions. They can be used to preview, add, edit or delete data, as well as perform calculations. They can also be used as a record source for a form or report. For these reasons, transactions are considered to be the brain of relational database system.

Every object in MS Access has properties, and transactions are objects in MS Access. You can set query properties to the query object as a whole or on the individual objects within a query. Later, you will see a detailed description of each query property. Having more properties in one query will increase the time to set or maintain this query.

In Access, there are a finite number of transactions that are available. One type of these transactions may be more complex to set or maintain than another; hence the type of the transactions that are used in a certain project may increase or decrease the complexity of a project.

To conclude the metrics that affect the time that a programmer may spend on this phase, which is defining transactions you have to consider the following metrics:

- Number of Transactions
- Number of Properties / Transaction

- Type of Transactions

3.3.1. Properties of a Transaction

There is a set of properties for each transaction, you can set some, none or all. Some properties are set by default. The properties are the following:

- Description: it is a field of 255 characters, in order to have a description of the query.
- Output All Fields: by default it is set to No. Setting this feature to Yes means that every single field on the QBE (Query By Example) grid is visible when the query runs.
- Top Values: by default it is set to All. This option enables you to query the top 5, 25, or 100 records in the query, or the top 5 percent or 25 percent. It actually calculates the top number or top percentage on the leftmost sorted field.
- Unique Values: This option enables you to return only records that are unique. By default it is set to No.
- Unique Records: This option enables you to get the unique records based on all fields in the underlying data source, not just those fields present in the QBE grid. By default it is set to No.
- Run Permission: This option enables you to give an end user authorization to perform an action that he or she would otherwise not be allowed to execute.
- Source Database: The default for this option is (current). With this property you can specify, in a string expression, an external database where the tables or queries for the current query remain.

- Source Connect Str: This option works in conjunction with the Source Database item. It is the name of the application in which the external database was created.
- Records Locks: This feature is used for multi-user systems. It determines how the records are locked while the query is being executed by a user. This item has only three options: No Locks (the default), All Records, and Edited Record.
- Recordset Type: This option allows you to decide whether the data available to a form or report will be a dynaset or a snapshot. Dynaset is the default and when set; you can edit bound controls based on a single table or tables with one-to-one relationship. While setting snapshot, no tables or the controls bound to their fields can be edited.
- ODBC Timeout: This item is only important when the database is or will be networked. It sets how long this workstation checks the network connection for response before telling the user that the network station is no longer connected to the network. The default is 60 seconds.
- Filter: This line is automatically filled in for you after specification have been made to the query during a run.
- Order By: This line is filled in for you by Access when the query is run and specifications have been made.
- Max Records: This option determines the maximum number of records returned by an ODBC database. The default is 0, means that it will process all records.

3.3.2. Type of Transaction

A finite number of transactions are available. Eventhough several different variations of transactions may be created, there are only two types of transactions: transactions that are acted on, or action queries, and transactions that are viewed, or select queries. Now you will see a list and description of the different types of transactions available in Access.

I - Select Queries

Select queries are those in which the data resulting from the query is viewed. There are two types of SELECT queries: simple and crosstab.

- **Simple Select**

Simple select queries are the most common of all the queries. They enable you to extract data from different tables and view it. Limited manipulations can be made with simple select queries.

They can calculate sums, averages, counts, and other types of totals on one or more tables.

- **Crosstab**

The second type of SELECT query is the crosstab query. It is somewhat like a simple select query in that it can calculate sums, averages, counts, and other types of totals on one or more tables. It differs from a simple select query in that it displays information not only the left column of the datasheet, but also across the top. A crosstab query is similar in appearance to a Pivot Table: it has row headings as well as column heading. When the crosstab query is selected, Access automatically adds a Total row and a Crosstab row to the QBE grid.

The purpose of a crosstab query is to perform a consolidation process on sections of data that intersect and to give summary information on that data. The data appears in a two-dimensional array or matrix with mathematical operations being performed at each intersection. Basically, a crosstab is the method by which a relational database can display data in a spreadsheet-like manner.

A crosstab has three major parts: Row Headers, the Column Header, and the Value. The Value is simply the result to be calculated at the intersection. At least one field or function must be chosen for the Column Header, the Row Header, and the Value. Up to three Fields can be chosen for the Column Header and the Value. Moreover, you can add a grand total column to the resulting query.

II - Action Queries

Action queries move or change data. Through Access action queries, you can make changes to tables of data, action query makes these changes in one operation. Only four types of action queries exist: MAKE TABLE, UPDATE, APPEND, and DELETE.

- **MAKE TABLE Query**

MAKE TABLE queries actually do exactly what their name implies. They make tables based on one or more other tables by utilizing all or part of the data from each preexisting table. There are several reasons why a developer would use the MAKE TABLE query:

- Allow for creating tables to export data to other Access databases.
- To create a query of historical information, it can retain all the information on a table or groups of tables up to a specific date.
- It can be used for creating a backup copy of a table.
- It can accelerate the performance of forms and reports that are based on multitable queries.

- **UPDATE Query**

UPDATE queries enable you to change data on a global scale. Executing an UPDATE query is uneventful. The records being changed don't fly past the screen.

The only way to tell whether the update has taken place is to open the table where the data being updated.

Usually, UPDATE queries based on one table are executable. If the UPDATE query contains two tables with a one-to-one relationship, it usually executes. If the UPDATE query is based on three or more tables having a one-to-many relationship, it won't work.

- **APPEND Query**

It adds a group of records from one or more tables to the end of one or more other tables. The APPEND query simply takes the new records and adds them to the end of an existing table.

- **DELETE Query**

The DELETE query enables you to delete records from one or more tables in a single action. As helpful as this capability can be, it can also be very dangerous. If the DELETE query is based on one table, the records that match the criteria are deleted when the query is run.

- **Parameter Queries**

Parameter queries give the user more flexibility. When a parameter query is run, Access display a dialog box prompting the user for more information.

Parameters can be set on just about all queries, including action and select queries. Parameters should be used liberally.

- **Union Queries**

Union queries combine the results of two or more queries. To create a union query, you must write it in code. In a union query, all the fields being combined must have the same data type, with one exception: a number field can have a corresponding Text field. By default, the UNION operator unions all rows in both statements, into a DISTINCT recordset (Recordset object represents the records in a table or the records that result from running a query. You use Recordset object to manipulate data in a database at the record level). If you want your dataset to return duplicate results from the combining of the two SELECT statements, you need to use the UNION ALL operator.

There is no theoretical limit to how many SELECT statements you can union together. The only limit is the internal amount of memory that Access has available to process queries.

3.4. Setting Indexes

Two things that affect the performance of transactions are the design of the database schema and the selection of indexes that are used on each table. Although in many cases changing the schema may provide the greatest performance improvements, it is often too costly to change an application to reflect the schema change. This means that the only option we are left with that can make an impact is to use indexes. By using an index on a table, the query now gains assistance from the index to sort the results.

The main tasks involved in deploying indexes are analyzing query requirements and deciding which indexes are and are not necessary to provide the desired performance. Usage of indexes therefore provides the best way to tune database tables and queries for data access.

User can improve data access in your Access database by following these basic guidelines for indexing:

- All database tables must have a primary key. It provides the fastest path to locate any record when its identifying properties are known in advance.

- Fields that are not part of the primary key and are frequently used for selecting records should be indexed.
- Do not "automatically" add indexes other than the primary key to a table. Indexes cost computer time and power to maintain, and it is possible to over-index a table.
- Absorb one index into another whenever possible.
- All records that are accessed for a single record SELECT, UPDATE or DELETE should be addressed by the primary key.
- Records that are selected as part of a frequently used single table list or grid should be accessed through an index.
- Each query that involves the joining of four or more tables should be carefully analyzed for indexing and joining.
- When two non-related tables are joined based on common elements, both tables should be indexed based on these same common elements.

3.5. Preparing Forms

Forms should constitute the fundamental interaction between the Access database and the end user. Forms guide the user through the operation of the application, protect the underlying data from accidental damage, and provide a level of security control if necessary.

Simple forms can be created through Access wizards. By taking the developer through the initial steps of form generation, wizards provide the initial skeleton upon which to build. You can do a big number of modifications to the resulting form, and these changes are the main metrics at this step. The number of modifications will imply if this form is more complex than another, therefore the metric is:

- Number of modifications

The modifications to a form are the following:

- Adding controls to a form: Controls are graphical objects that allow the user to view data or perform an action. Controls include:

    - Text Boxes: it is used to display and edit text or numeric values, it can be bound to a field in the database.
    - Label Control: it is used to display textual information.
    - Combo Boxes and List Boxes: they provide an easy way for a user to make a selection, as well as an alternate method of displaying rows from a database.
    - Option Groups: they are a good way to display limited multiple choices to users.
    - Tab Control: You use the Tab Control to separate the controls on a form into multiple pages, it is similar to a traditional card file.

- Link OLE documents to a form: You can add objects created in other Windows applications to an Access form. For example, you can add a picture created with Microsoft Paint, or a worksheet created with Microsoft Excel, or a document created with Microsoft Word.
- Control Alignment and placement
- Moving and Sizing Control
- Text Alignment: it affects how the text is aligned in the control.
- Colors: They make forms easier to read and understand.
- Borders: Several properties affect the borders of controls and form. They are border style, border width and border color.
- Effects: it enables you to specify how the control or section appears.
- Changing Text Fonts and Sizes: Special fields or labels can be denoted by different font sizes.
- Changing a Control to another: As you are designing forms, you may sometimes need to change a control to another type.
- Unbound Controls: They are used to convey information to the users or receive it from them, but is not linked to a field in the database.

    - Command Buttons: When you insert a command button, the wizard set it to do many of the most popular actions.
    - Calculated Controls: They use expressions to derive their data, rather than direct user or database field input.

- Adding Shapes and Lines to Forms: You can make forms more readable by breaking up sections using shapes and lines.

- Adding Graphics to a Form: Using graphics is another means of creating an impact on drawing attention to a form. Access supports two methods for adding graphics to a form. The first, you can add an overall graphic background. The second is to use the image control, which enables you to place many graphics on the form and position them wherever you want.

3.6. Preparing Reports

Printing a report from Access is often the final result of the database effort. No matter how great a user interface is, printed output is more comprehensible to most people. Even though reports can be simply rows and columns of text, people have high expectations for how a report will look. Forms and reports are very similar in how they manipulate controls, sections, and properties.

Simple reports can be created through Access wizards. By taking the developer through the initial steps of report generation, wizards provide the initial skeleton upon which to build. You can do a big number of modifications to the resulting report, and these changes are the main metrics at this step. The number of modifications will imply if this report is more complex than another, therefore the metric is:

- Number of modifications

The modifications to a report are the following:

- Moving and sizing controls
- Report customization with properties: numerous properties must be set when a report is created.
  - OrderBy Property: this property orders the report in the manner specified.
  - Caption Property: This entry will be displayed in the report's title bar, and will be used for establishing a hyperlink to this report.
  - Picture Properties: There are five picture properties that when manipulated can create various creative effects.
  - HasModule Properties: It is used to tell Access whether the report has a module page (code behind the report).
- Expressions: The key role of a good report is displaying data in an understandable fashion.
  - Concatenating Fields for a Text Box
  - Turning a Yes/No Field into Text
  - Using If...Then Statements with the IIF Function: The IIF, or immediate if, function performs a simple If...Then test. The arguments for this function have three parts: the Test, True value, and False value. In the Test portion, write a test against a field value. If that test passes, it displays the contents of True; otherwise, it displays the contents of False.
  - Turning the Display Off a Field if the Value is Null: If a customer on an invoice report doesn't get a discount, you don't display the control for Discount or its label.
  - Retrieving Records through Dlookup Statements: When the report requires a record from a foreign table, and a subreport is not justified, the Dlookup statement is a solution.

- Combining two or more Text Fields into one (ex. Address Block).
- Referencing an Open Form on the Report: Sometimes a report is run from a button the user presses on a form. The data on that form can be used in the report without queries or Dlookup.
- Adding Up Report Data with the Sum Function: You can use a wizard to insert the Sum Function into the Report Footer automatically.

• Embedding Hyperlinks into Access Reports: When a hyperlink is added to a report, it cannot be used directly as a clickable item, but the hyperlink will print on a sheet of paper as underlined text.

• Programming in reports: This includes the actions that any report could perform if programmed the following events:

- OnOpen: Occurs when the report is opened but before the report is previewed or printed.
- OnActivate: When the report receives focus and becomes the active window.
- OnDeactivate: When the report loses focus, which means that you have selected another form, query, or report.
- OnNoData: Occurs after Access has formatted a report for printing but with an empty recordset.
- OnPage: Occurs after Access formats a page of a report for printing but before the page is printed.
- OnError: Occurs when a run-time error is produced in Access.
- OnClose: Occurs when the report is closed and removed from the screen.

3.7. Summary

In summary, this chapter covered the most important steps a programmer may pass while developing a project in MS Access. The time factor is the main criteria used to determine the major attributes in calculating the effort and duration of Access projects.

Chapter 4

Aggregate Complexity Metrics

In this chapter, we are proposing DBP from which effort (in person-days) can be estimated. We apply the suggested metrics discussed in the previous one on a number of different applications. Throughout our work, we have faced a number of difficulties that urged us to find new metrics. This metric will combine all the suggested metric in a relevant way and will be referred to as Database Points.

Given a project developed in Ms Access, the DBP (Database Points) will evaluate the following:

- The business value of the system to the user.
- Project size, cost and development time.
- Maintenance, modification and customization effort.

4.1. Applying the Suggested Metrics

The verification of any metric is by applying it on a set of different projects, implemented by different programmers. There was available a set of applications that are of various sizes. These applications were designed for various institutions: restaurants, insurance companies, hotels, LAU listening center, LAU audio visual center, car companies, etc. Different programmers with different background and experience wrote these applications using MS Access. This was helpful in our work,

since we need to know the duration a set of programmers may need on different applications of various difficulty.

The primary step in applying the suggested metrics was to take each application, count the metrics and try to apply a formula that combines them all. There were certain applications that took a lot of time to count their metrics and then to apply the formula.

At this stage, and after applying the suggested metrics an all the available applications, there was a set of problems that hindered us from using the suggested metrics as a metric for relational database projects using MS Access. The problems are the following:

1. There is huge number of metrics suggested at each stage. At each stage we had around four to six metrics.
2. A lot of time is needed at each stage in order to count the suggested metrics. For example, the number of fields, properties for each field.
3. Some of these metrics were directly related to other metrics. For example, the number of tables and the number of fields are directly related and they could be combined in one category.
4. Some of these metrics are not that important in determining the time and effort of the programmer. For example, the number of indexes is not that important since they are counted by properties of the field.

4.2. Categories of the Database Metric

After facing all these problems, the attitude was to search for a smaller set of metrics that combine all the suggested metrics but in an easier way. Then to attach them in one formula that would be easy and direct to apply.

While analyzing the available projects and trying to apply the suggested metrics, I have seen that these metrics come into five categories: Tables, Relationships, Transactions, Forms and Reports. These categories combine all the suggested metrics and will be the base of DBP (Database points) that determines the duration and effort of any Access project.

These categories could be classified as Simple, Average, or Complex according to its complexity. The factors that determine the category's complexity will be discussed through the explanation of each category.

4.2.1. Tables

It is the number of different tables that are set in any database. Setting these tables include defining the properties of each table, setting all the fields in the tables, and defining properties for each field. The factors that influence the classification of this category are the following:

- Number of Fields / Table: it is the number of fields defined in each table.

- Properties / Field: it is the type of properties that are defined for each field. Some properties will increase the complexity of this category more that other properties.
- Table Properties: it is the type of table properties that are defined that increase the complexity of this category. Some properties will increase the complexity than others.

Through our survey on the set of applications, we were able to classify tables as simple, average or complex. The factors that influenced the classification were listed above. Most of the tables that we encountered were of Simple or Average class. The simplest way of setting any table is to set a small number of fields, not greater than twenty fields. Each field can have a set of properties that influence only the way this field is displayed. For each table, you should set at least one key field.

The more complex way of setting a table is to add the number of fields (to be greater than thirty) or to add properties that influence the validation of each field and table. As a result, the complex way of setting a table will be everything else other than those classified as simple or average.

Any table could be classified as *Simple* if it satisfies all of the following:

- Number of fields: The number of fields in the table should be less than or equal to 20.
- Properties / Field: you can set one or more of these properties or none: Format, Input Mask, Caption, Default Value, Indexed.

- Properties / Table: you should not set any property.

Any table could be classified as *Average* if it satisfies any of the following:

- Number of fields: The number of fields in the table should be greater than 20 and less than or equal to 30.
- Properties / Field: You can set any of the properties that were described in the *Simple* class added to it one of the following: Validation Text, Validation Rule.
- Properties / Table: You should set one or more of the following properties: Validation Rule, Validation Text.

Any table could be classified as *Complex* if it satisfies any of the following:

- Number of fields : The number of fields could be any number greater than 30.
- Properties / Field: All of the field properties should be set.
- Properties / Table: You can set any of the properties that were described in the *Average* class added to it one of the following: Filter, Order By, Orientation.

4.2.2. Relationships

It is the number of distinct relations connecting any two tables defined in the ER-diagram. Defining a relation between two tables include defining the fields that make the relation, setting the type of the relationship (one-to-one, one-to-many, many-to-one) and setting the properties that define these relations. The factors that influence the classification of this category are the following:

- Type of relationship: it is the type that defines the relationship. Any relationship defined in MS Access could be one of the following: One-to-One, One-to-Many, Many-to-One.
- Properties of relationship: it is the type of properties that are defined for each relationship. Some properties will increase the complexity of this category more than others.

The relationships that we encountered were of different complexity. The majority of relationships that were simple to set are One-to-Many with no properties. The more complex class to set was any type of relationship with enforcing integrity property to be set. The last class will be any type of relationship with all properties set.

Any relationship could be classified as *Simple* if it satisfies all of the following:

- Type of relationship: it could be any of the following: One-to-Many, Many-to-One.
- Properties of relationship: you should not set any property.

Any relationship could be classified as *Average* if it satisfies all of the following:

- Type of relationship: it could be any of the following: One-to-Many, Many-to-One, One-to-One.

- Properties of relationship: you should set Enforce Referential Integrity.

Any relationship could be classified as *Complex* if it satisfies all of the following:

- Type of relationship: it could be any of the following: One-to-One, One-to-Many, Many-to-One.
- Properties of relationship: you should set all of the following: Enforce Referential Integrity, Cascade Update Related Fields, Cascade Delete Related Fields.

4.2.3. Transactions

It is the number of distinct transactions defined in the database. Any transaction could retrieve, update, or delete data from one or more tables. Defining a transaction includes defining the type of the transaction, the tables that are needed to retrieve data, and setting properties of each transaction. The factors that influence the classification of this category are the following:

- Type of the transaction: In MS Access, there are a big number of transactions. These transactions could be categorized into : select transactions or action transactions
- Properties of transactions: it is the type of properties that are defined for each transaction. Some properties will increase the complexity of this category more than other properties.

In the set of applications that we encountered, there were a large number of different transactions. The first set, which is the majority of these transactions, was simple select transactions. Moreover, the easiest to set are those transactions that are concerned with retrieving data (such as Select, Crosstab), and those concerned with modifying recordset (Update, Append and Delete). Concerning transaction properties, the easiest are those concerned with the number of records to be modified or retrieved.

The second set of transactions that we encountered is composed of Make Table and Parameter Query. The complex set of transactions will be everything else that was not mentioned in the Simple or Average class.

Any transaction could be classified as *Simple* if it satisfies all of the following:

- Type of transaction: The transaction could be one of the following: Select, Crosstab, Update, Append, Delete.
- Properties of the transaction: you can set none, one or more of the following: TopValues, Output All Fields, Unique Values, Unique Records.

Any transaction could be classified as *Average* if it satisfies all of the following:

- Type of transaction: it could be any of the type of classes defined in the *Simple* class or one of the following: Make Table Query, Parameter Queries.
- Properties of the transaction: you can set any of the properties defined in the *Simple* class added to it any of the following: Run Permission, Source Database, Source Connect Str, Records Lock, Recordset Type, ODBC.

Any transaction could be classified as *Complex* if it satisfies all of the following:

- Type of transaction: it could be any of the transactions defined in the *Average* class or one of the following: Union queries, Nested Queries.
- Properties of transaction: you can set any of the properties defined in the *Average* class added to it any of the following: Filter, Order By, Max Records.

4.2.4. Forms

It is the number of distinct forms that are defined in any database. Setting a form includes describing the query or table that acts as the source for the data to be retrieved, displayed or added. It also includes describing function keys, adding effects, adding shapes and lines, adding graphics. The factor that determines the classification of this category is the type of modifications done to any form.

Forms could be created using wizards or without wizards. The majority of forms that we encountered and the easiest way of setting a form is to move controls (if using wizards) or moving controls. These controls are concerned with the way the fields are displayed. The more difficult way of setting a form is concerned with adding special effects to the form (shapes, lines, etc.). The most difficult way (which is the Complex class) of setting a form will be everything else that is not mentioned in the Simple or Average class.

Any form could be classified as *Simple* if it satisfies one of the following:

- Adding controls to a form: Controls are graphical objects that allow the user to view data or perform an action. Controls include:
  - Text Boxes.
  - Label Control.
  - Combo Boxes and List Boxes.
  - Option Groups.
  - Control Alignment and Placement:
  - Changing Text Fonts and Sizes.
  - Command Buttons.

Any form could be classified as *Average* if it satisfies any of the changes described in the *Simple* class added to it any of the following changes:

- Tab Control.
- Link OLE documents to a F

- orm.
- Moving and Sizing Controls
- Text Alignment.
- Colors.
- Borders.
- Effects.
- Adding Shapes and Lines to Forms.

Any form could be classified as *Complex* if it satisfies any of the changes described in the *Average* class added to it any of the following changes:

- Changing a Control to Another.
- Unbound Controls.
- Calculated Controls.
- Adding Graphics to a Form.

4.2.5. Reports

It is the number of distinct reports that are defined in a database. Defining a report include defining the query or table that acts the source of retrieving information, changing the controls, and performing some programming on certain events. The factor that influences the classification of this category is the type of changes to the report.

Reports could be created using wizards based on the result of a query or on a table. The result of this report wizard is a report that could be modified as the requirements specify. The simplest way of modifying a report is only to change or size controls that are available in the report. If more changes were needed, then this would be more complex (and could be classified as Average); these changes are concerned with the properties of the reports and adding expressions that are concerned with the display of the fields. The most difficult changes (or as described Complex class) are those concerned with adding programmable events to the report.

Any report could be classified as *Simple* if it satisfies one of the following:

- Moving and sizing controls.
- Adding subreports.

Any report could be classified as *Average* if it satisfies any of the changes described in the *Simple* class added to it one of the following:

- Report customization with properties: numerous properties must be set when a report is created.
  - OrderBy Property.
  - Caption Property.
  - Picture Properties.
  - HasModule Properties.
- Expressions: The key role of a good report is displaying data in an understandable fashion.

- Concatenating Fields for a Text Box.
- Turning a Yes/No Field into Text.
- Using If...Then Statements with the IIF Function.
- Turning the Display Off a Field if the Value is Null.
- Retrieving Records through Dlookup Statements.
- Combining Text Fields for an Address Block.
- Referencing an Open Form on the Report.
- Adding Up Report Data with the Sum Function.

Any report could be classified as *Complex* if it satisfies any of the changes described in the *Average* class added to it one of the following:

- Embedding Hyperlinks into Access Reports: When a hyperlink is added to a report, it cannot be used directly as a clickable item, but the hyperlink will print on a sheet of paper as underlined text.
- Programming in reports: This includes the actions that any report could perform if programmed the following events:
  - OnOpen.
  - OnActivate.
  - OnDeactivate.
  - OnNoData.
  - OnPage.
  - OnError.
  - OnClose.

4.3. Aggregate Database Metrics

After this description of the five essential categories that describe any relational database project, Table 4.1 contains all the weights associated with all the classes of the categories. The weights defined later could be described as the time unit each class would require from the programmer. At the end of this section, we present a formula that sums up all these weights. This formula gives the effort in Working days, assuming six working hours per day. Moreover, there will be a number of adjusting factors that will influence the overall result of the project.

Table 4.1 Aggregate Database Metrics

	Complexity Weighting			
Category	**Simple**	**Average**	**Complex**	
Tables	☐ * 7 +	☐ * 10 +	☐ * 15 =	☐
Relationships	☐ * 2 +	☐ * 3 +	☐ * 5 =	☐
Transactions	☐ * 5 +	☐ * 7 +	☐ * 10 =	☐
Forms	☐ * 4 +	☐ * 5 +	☐ * 7 =	☐
Reports	☐ * 4 +	☐ * 5 +	☐ * 7 =	☐
DBP				☐

After classifying all the categories of the project and multiplying the number of categories by each associated weight, we get the total, which is the sum of all the categories. Later, use this total in the following formula to get the effort

$$\text{Effort} = \text{DBP} * (0.2 + [0.01 * \Sigma F_i]) \qquad \textbf{[4.1]}$$

Fi = the weights given to each of the adjusted factors.

Each of the following nine adjusting factors is given a weight which ranges from 0 to 5. The weights are given empirically and discussion of all these factors will be in the next section.

1. Does the system have an interface with Visual Basic?
2. What is the rate of the programmer's experience?
3. Did the programmer use Wizards and to what rate?
4. Does the system take networking into consideration?
5. Does the system provide Help for users?
6. Centralized or distributed processing?
7. Did the programmer use Visual Queries or Issue Queries?
8. Did the Access application use Windows API?

4.3.1. Adjusting Factors

In this section, there will be a description of the nine adjusting factors that have been mentioned in equation (4.1). These factors are considered to add complexity to the time measurement, but not major steps in developing a project in MS Access. These factors take into consideration the outside influences that apply directly to any project; they vary from the programmer and his experience to the system itself. Each of the adjusting factors could be empirically given a weight, which ranges from 0 to 5. The sum of the factors' weights will be used in equation (4.1); hence, this sum could vary from 0 to 45. The adjusting factors are the following:

1. Does the system have an interface with Visual Basic or any other interface?

   If a system has an interface with Visual Basic or any other interface, then this factor should be given more weight than when a system has no interface. Since developing an interface with Visual Basic or any other interface will require more time and effort.

2. What is the rate of the programmer's experience in Access?

   The programmer's experience in developing any project is an important factor in increasing or decreasing time and effort. This would apply to any language. Therefore, if a programmer has good experience in Access, then this factor should be given less weight than when a programmer has no experience.

3. Did the programmer use Wizards and to what rate?

   Wizards take the programmer through several different steps, then it creates the initial skeleton (of a form, report... etc.) upon which to build. The use of wizards saves a lot of time. Therefore, if a programmer uses a lot of wizards, then this factor should be given less weight than when a user never uses wizards.

4. Does the system take networking into consideration?

A system that is developed to work on a network (or for multi-user environment) will require more time and effort in order to take into consideration all the issues that are related to networking. Some of these issues are the read/write access to the common database shared folders, user's workgroups, Access installation on all systems... etc. If a system takes networking into consideration then this factor should be given more weight than when a system does not take networking into consideration.

5. Does the system provide Help for users?

The availability of clear, concise and thoughtfully organized online Help will please the user and reduce the number of technical calls you receive. Providing Help for your application takes a little extra time. If this system provides online Help then this factor should be given more weight than when the system does not provide this feature.

6. Centralized or distributed processing?

Access can be used to develop standalone and LAN applications, but it can also serve as a powerful and flexible front end for client/server applications. This means that you can upsize applications for dozens or hundreds of users, or use Access as a data analysis tool for report writing, queries, and business graphics. To develop an efficient client/server application, the user must master not only Access but also the database server (Microsoft SQL Server, Sybase SQL Server, Oracle Server). If a system was designed to work on a

client/server environment, then this factor should be given more weight than when a system was designed to work on LAN or as a standalone.

7. Did the programmer use Visual Queries or Issue Queries?

Access has new and improved wizards and builders that make it much easier to create quite large and complex queries. These tools are the QBE (Query By Example) or Visual Queries. These tools have limits, however, and as excellent as they are, some things can not be done with them or simply won't work outside a native Access environment (for example, using SQL passthrough to query data on an SQL database server). The second type of queries is Issue queries where you write standard queries using SQL strings, it requires more time and effort to write and compile. Hence, if a programmer uses the second type more then this factor should be given more weight than when a programmer uses the first type.

8. Did the Access application use Windows API?

Using MS Access, you can develop sophisticated applications that can satisfy even the most demanding users. Sometimes, you might want more control of your application or want to do something that is not directly supported by Access. You can make a whole new set of functionality available to you by calling procedures contained in dynamic link libraries (DLLs). Microsoft Windows operating system contains several DLLs that make up what is called

the Windows API. In fact, all Windows applications use the Windows API to perform tasks such as creating windows, changing window size, reading and writing files, and so on.

If an application uses Windows API, then this factor should be given more weight than when an application doesn't use Windows API.

4.4. Empirical Justification of Table 4.1 and Formula 4.1

As we have mentioned in chapter 2, we made a survey of a number of structural and complexity metrics. Most of these metrics were not applicable to database languages due to several reasons. However, there was one metric which was independent of any language and it calculates time and effort, it is Function Points. This metric was the base of the Aggregate Database Metric that was mentioned in section 4.3.

The categories in Table 4.1, were not the same of Function Points, but were established as described in section 4.2 to be suitable for database languages. In other words, these categories take all the phases a programmer pass through when developing in Access, and these are the base for calculating the effort of a programmer. They resemble the four categories of Function Points: Input, Output, Interfaces and Files.

Each of the categories in Aggregate DB metric has three classes: Simple, Average and Complex. This was borrowed also from Function Points, since each category had the same three classes. In section 4.2, there was a detailed description of

the factors that influence the classification of each category and justification for dividing the classes.

The weights that were shown in Table 4.1 were borrowed from the Function Point metric with some modifications to be applicable to database languages. These weights could be considered as the time unit a programmer might spend when setting a category. These weights are modified as follows:

- Weights for the first category (Tables) were more than weights for other categories, since setting a table includes several steps: setting table properties, setting fields and field properties. While relationship category was given less weights, since defining a relationship requires less time, it only includes defining the type of relationship and its properties
- When a category was considered to be as complex as a category that was defined in Function Point metric, the weights for Simple, Average and Complex were borrowed exactly without any modifications. For example, the weights for number of files are 7,10,15 for simple, average and complex respectively. These weights were borrowed exactly the same for Tables since they were considered to be of the same complexity with respect to their metric.
- Relationships were given the least weights since it only includes (as described in section 4.2) setting the type of relationship and its properties. However, weights for transactions were fewer than those for Tables but higher than Forms and Reports since setting a transaction requires more time and effort.

- Reports and Forms were given exactly the same weights because they are of same complexity. Both include changes to the controls, modification to the overall report or form, and some programming effects could be applied at request.

After setting the weights for each category that was described in section 4.2, we wanted to sum all these weights in a formula that is capable of giving the exact time and effort. This formula should resemble the function point formula but with modification to the constants.

$$FP = DBP *(x + [y * \Sigma F_I]) \quad \textbf{[4.2]}$$

In order to find the x and y in formula (4.2), we collected a number of Access applications (small or medium size). We applied weights as were described in section 4.3, and in advance we knew how long each application required from the programmer. After making a statistics of all these applications, we tried to find constants that represent x and y (in formula 4.2) to satisfy approximately the effort and time of each application. That what lead us to set x = 0.2 and y = 0.01 (as described in formula 4.1).

The last step was to find adjusting factors that resemble those discussed in function points but that are relevant to database languages. Those factors were discussed deeply in section 4.2.1. The verification of the weights in Table 4.1 will be in the next chapter where we are going to apply the Aggregate DB metric to a number of applications of different size and complexity.

4.5. Summary

A new metric on relational database languages was formulated to calculate the database points (DBP) which are the main attributes in calculating the effort and duration of a project. Moreover, some adjusting factors were also discussed which also influence the overall complexity of a system. As a result, you can estimate the business value of a system, cost, etc.

Chapter 5

Empirical Validation

5.1. Introduction

Setting aggregate database metrics is a major step. However, validating this metric is even more important, in order to be sure that this metric will accurately calculate the effort and duration of Access projects. To validate these metrics, we surveyed a number of applications designed for various institutions by different programmers of various complexities and sizes.

5.2. Hotel Application

This is a small application that is concerned with reserving rooms for customers, supplying food and beverages for these rooms.

Properties

Number of Tables: 10

Properties of Fields:

Table 1: Simple

Number of fields: 7 fields

Field 1,5: Indexed

Field 2,3,4,6,7: no properties.

Table 2: Simple

Number of fields: 3 fields

Field 1: Indexed

Field 2: no properties

Field 3: Format

Table 3: Simple

Number of fields: 2

Field 1: Indexed

Field 2: no properties

Table 4: Simple

Number of fields: 3

Field 1: Default value, Indexed

Field 2: Indexed

Field 3: Default value

Table 5: Simple

Number of fields: 2

Field 1: Default value, Indexed

Field 2: No properties

Table 6: Simple

Number of fields: 7 fields

Field 1, 5, 6: Default value, Indexed

Field 2,3: no properties

Field 4: Indexed

Field 7: Format, Default value

Table 7: Simple

Number of fields: 6

Field 1, 6: Indexed

Field 2: Default value, Indexed

Field 3, 4: Format

Field 5: no properties

Table 8: Simple

Number of fields: 4

Field 1,4: Default value, Indexed

Field 2,3: no properties

Table 9: Simple

Number of fields: 5

Field 1,2,4: Default value

Field 3,6: no properties

Table 10: Simple

Number of fields: 2

Field 1: Indexed

Field 2: no properties

Number of Relationships: 8 Complex

Type of relationship: 8 relations are One-to-Many

Properties of relationship: Enforce Referential Integrity

Cascade Update Related Fields

Cascade Delete Related Fields

Transactions: 2 Simple

Type of transactions: 2 Select queries

Forms: 7 Simple

1 Average

Reports: 1 Simple

	Simple		Average		Complex		Total
Tables	10	* 7 +		* 10 +		* 15 =	70
Relationships		* 2 +		* 3 +	8	* 5 =	45
Transactions	2	* 5 +		* 7 +		* 10 =	10
Forms	7	* 4 +	1	* 5 +		* 7 =	33
Reports	1	* 4 +		* 5 +		* 7 =	4
DBP							162

1. Does the system have an interface with Visual Basic? 0
2. What is the rate of the programmer's experience? 2
3. Did the programmer use Wizards and to what rate? 2
4. Does the system take networking into consideration? 0
5. Does the system provide Help for users? 0
6. Centralized or distributed processing? 0
7. Did the programmer use Visual Queries or Issue Queries? 0
8. Did the Access application use Windows API? 0

Effort = 162 * [0.2 + (0.01 * 4)]

= 162 * 0.24 = 38.88 Working Days

5.3. LAU Listening Center

This system was designed for LAU listening center, where you can set schedules for professors, assistants and students. You can list all the information mentioned above.

Properties

Number of Tables: 4

Properties of Fields:

Table 1: Average

Number of fields: 13 fields

Field 1: Indexed, Format, Validation Text, Required

Field 2: Validation Text, Required, Indexed

Field 3,4,6,7: Required

Field 5,8,9,10,11,12: no properties

Field 13: Format

Table 2: Simple

Number of fields: 17 fields

Field 1: Indexed, Required

Field 2: Required

Field 3,4,5,6.17: no properties

Table 3: Average

Number of fields: 27 fields

Field 1: Indexed, Validation Text, Required

Field 2: Input Mask, Validation Text

Field 3: Validation Text, Required

Field 4,10: Required

Field 5,6: Format, Validation Rule

Field 7,8,9,27: no properties

Field 11,12,13.26: Format

Table 4: Average

Number of fields: 24

Field 1: Format, Validation text, Required, Indexed

Field 2,4: Validation text

Field 3,7: Required

Field 5,6: Format, Validation rule

Field 8,9,10..23: Format

Field 24: no properties

Number of relationships: 1 Average

Type of relationship: One-to-One

Properties of relationship: Enforce Referential Integrity

Transactions : 1 Simple

Type of transactions: 1 Select query

Forms : 6 Simple

Reports : 9 Simple

	Simple		Average		Complex		Total
Tables	1	* 7 +	3	* 10 +		* 15 =	37
Relationships		* 2 +	1	* 3 +	8	* 5 =	3
Transactions	1	* 5 +		* 7 +		* 10 =	5
Forms	6	* 4 +	1	* 5 +		* 7 =	24
Reports	9	* 4 +		* 5 +		* 7 =	36
DBP							105

1. Does the system have an interface with Visual Basic? 0
2. What is the rate of the programmer's experience? 3
3. Did the programmer use Wizards and to what rate? 1
4. Does the system take networking into consideration? 0
5. Does the system provide Help for users? 0
6. Centralized or distributed processing? 0
7. Did the programmer use Visual Queries or Issue Queries? 0
8. Did the Access application use Windows API? 0

Effort = 105 * [0.2 + (0.01 * 4)]

= 105 * 0.24 = 25.2 Working Days

5.4. Pharmacy Application

This is a small application that was designed for a pharmacy. You can have an invoice for each customer. This application provides an inventory system for medicines as well as a checking list for expiry date.

Properties

Number of Tables: 6

Properties of Fields:

Table 1: Simple

Number of fields: 1

Field 1: no properties

Table 2: Simple

Number of fields: 4 fields

Field 1: Indexed

Field 2,3: no properties

Field 4: Input mask

Table 3: Simple

Number of fields: 5 fields

Field 1: Indexed

Field 2,4: no properties

Field 3: Format

Field 5: Input mask

Table 4: Simple

Number of fields: 6

Field 1,6: Indexed

Field 2: Input mask, Indexed

Field 3: Input mask

Field 4,5: no properties

Table 5: Simple

Number of fields: 3 fields

Field 1,2,3: no properties

Table 6: Simple

Number of fields: 5 fields

Field 1: Indexed

Field 2,3,5: no properties

Field 4: Input mask

Number of relationships: none

Transactions: 2 Simple

Type of transactions: 2 Select query

Forms: 13 Simple

Reports: none

	Simple		Average		Complex		Total
Tables	6	* 7 +		* 10 +		* 15 =	42
Relationships		* 2 +		* 3 +		* 5 =	
Transactions	2	* 5 +		* 7 +		* 10 =	10
Forms	13	* 4 +		* 5 +		* 7 =	52
Reports		* 4 +		* 5 +		* 7 =	
DBP							104

1. Does the system have an interface with Visual Basic? 0
2. What is the rate of the programmer's experience? 3
3. Did the programmer use Wizards and to what rate? 1
4. Does the system take networking into consideration? 0
5. Does the system provide Help for users? 0
6. Centralized or distributed processing? 0
7. Did the programmer use Visual Queries or Issue Queries? 0
8. Did the Access application use Windows API? 0

Effort = 104 * [0.2 + (0.01 * 4)]

= 104 * 0.24 = 24.96 Working Days

5.5. Restaurant Application

This is a small application that was designed for Pizza Hut delivery system, Labban branch. This system accepts an order, issue an invoice, have a schedule for all employees at this branch.

Properties

Number of Tables: 7

Properties of Fields:

Table 1: Simple

Number of fields: 10 fields

Field 1: Indexed, Caption, New value

Field 2,4,5,6,7: Caption

Field 9: Input mask, Caption

Field 10: Format, Caption

Table 2: Simple

Number of fields: 3 fields

Field 1: Caption, Indexed

Field 2,3: Caption

Table 3: Simple

Number of fields: 3 fields

Field 1: Indexed, New value, Caption

Field 2: Caption

Field 3: Caption, Indexed

Table 4: Simple

Number of fields: 6 fields

Field 1: Caption, Indexed

Field 3,4,5: Caption

Field 6: Input mask, Caption

Table 5: Complex

Number of fields: 60 fields

Table 6: Simple

Number of fields: 5 fields

Field 1: New Value, Caption, Indexed

Field 2,3: Caption, Indexed

Field 4: Input mask, Caption

Field 5: Caption

Table 7: Simple

Number of fields: 5 fields

Field 1,2,3,4,5: no properties

Number of relationships: 3 Simple

Type of relationship: One-to-Many

Properties of relationship: no properties

Transactions : 1 Simple

Type of transactions: 1 Select query

Forms: 9 Simple

Reports: none

	Simple		Average		Complex		Total
Tables	6	* 7 +		* 10 +	1	* 15 =	57
Relationships	3	* 2 +		* 3 +		* 5 =	6
Transactions	1	* 5 +		* 7 +		* 10 =	6
Forms	9	* 4 +		* 5 +		* 7 =	36
Reports		* 4 +		* 5 +		* 7 =	
DBP							105

1. Does the system have an interface with Visual Basic? 0
2. What is the rate of the programmer's experience? 3
3. Did the programmer use Wizards and to what rate? 1

4. Does the system take networking into consideration? 0
5. Does the system provide Help for users? 0
6. Centralized or distributed processing? 0
7. Did the programmer use Visual Queries or Issue Queries? 0
8. Did the Access application use Windows API? 0

Effort = 105 * [0.2 + (0.01 * 4)]

= 105 * 0.24 = 25.2 Working Days

5.6. Video Shop Application

This is a small application that was designed for a video store. This system keeps records of all customers and video tapes.

Properties

Number of Tables: 7

Properties of Fields:

Table 1: Average

Number of fields: 6 fields

Field 1: Indexed, Format, New value

Field 2: Caption, Indexed

Field 3,4: Input mask, Caption, Validation rule, Validation text

Field 5: Format, Caption, Default value, Validation rule, Validation text

Field 6: Format, Caption, Validation rule

Table 2: Average

Number of fields: 9 fields

Field 1: Indexed, New value

Field 2: Caption, Validation rule, Validation text, Required, Indexed

Field 3,4: Caption, Validation rule, Validation text

Field 5: Input mask, Caption, Validation rule, Validation text

Field 6,8: Caption

Field 7: Caption, Default value

Field 9: no properties

Table 3: Average

Number of fields: 4 fields

Field 1: Indexed, Caption, New value

Field 2: Caption, Default value, Validation rule, Validation text, Indexed

Field 3,4: Caption, Validation rule, Validation Text

Table 4: Simple

Number of fields: 2 fields

Field 1,2: Default value, Indexed

Table 5: Simple

Number of fields: 5 fields

Field 1,2,3,4,5: no properties

Table 6: Simple

Number of fields: 8 fields

Field 1,2: Indexed

Field 3,4,5,6,7,8: no properties

Table 7: Average

Number of fields: 7

Field 1: Indexed

Field 2,3,4,5: Caption, Validation rule, Validation text

Field 6,7: Format

Number of relationships: 4 Complex

Type of relationship: One-to-Many

Properties of relationship: All properties

2 Simple

Type of relationship: One-to-Many

Properties of relationship: no properties

Transactions: 10 Simple

Type of transactions: 10 Select queries

Forms: 18 Simple

Reports: none

	Simple		Average		Complex		Total
Tables	3	* 7 +	4	* 10 +		* 15 =	61
Relationships	2	* 2 +		* 3 +	4	* 5 =	24
Transactions	10	* 5 +		* 7 +		* 10 =	50
Forms	18	* 4 +		* 5 +		* 7 =	72
Reports		* 4 +		* 5 +		* 7 =	
DBP							207

1. Does the system have an interface with Visual Basic? 0
2. What is the rate of the programmer's experience? 1
3. Did the programmer use Wizards and to what rate? 1
4. Does the system take networking into consideration? 0
5. Does the system provide Help for users? 0
6. Centralized or distributed processing? 0
7. Did the programmer use Visual Queries or Issue Queries? 0
8. Did the Access application use Windows API? 0

Effort = 207 * [0.2 + (0.01 * 2)]

= 207 * 0.22 = 45.54 Working Days

5.7. Hospital Application

This is a medium size application that was designed for a hospital. It takes into consideration most of the departments in a hospital: accounting, administration, admission, circulation, inventory, laboratory. It keeps records of all patients, staff and doctors. It has an inventory system for all medicine, as well as a billing system

Properties

Number of Tables: 28

Properties of Fields:

Table 1: Average

Number of fields: 5 fields

Field 1,5: Indexed

Field 2,3: no properties

Field 4: Validation rule, Validation text

Table 2: Simple

Number of fields: 5 fields

Field 1,2: Indexed

Field 3,4,5: no properties

Table 3: Simple

Number of fields: 3 fields

Field 1: Indexed

Field 2: no properties

Field 3: Format

Table 4: Simple

Number of fields: 2 fields

Field 1: Indexed

Field 2: no properties

Table 5: Simple

Number of fields: 4 fields

Field 1,2,3,4: no properties

Table 6: Average

Number of fields: 7 fields

Field 1,5: Indexed

Field 2,4,7: no properties

Field 3,6: Validation rule, Validation text

Table 7: Simple

Number of fields: 2 fields

Field 1: Indexed

Field 2: no properties

Table 8: Simple

Number of fields: 6 fields

Field 1: Indexed

Field 2,3,4,5,6: no properties

Table 9: Simple

Number of fields: 7 fields

Field 1,2,4,5,6: no properties

Field 3: Indexed

Field 7: Input mask

Table 10: Simple

Number of fields: 5 fields

Field 1,2,3.4,5: no properties

Table 11: Simple

Number of fields: 8 fields

Field 1: Indexed

Field 2,3,4,7,8: no properties

Field 5,6: Format

Table 12: Simple

Number of fields: 3 fields

Field 1,2,3: no properties

Table 13: Simple

Number of fields: 4 fields

Field 1,2,3,4: no properties

Table 14: Simple

Number of fields: 2 fields

Field 1: Indexed

Field 2: no properties

Table 15: Simple

Number of fields: 8 fields

Field 1: Indexed

Field 2,3,4,7,8: no properties

Field 5,6: Format

Table 16: Average

Number of fields: 5 fields

Field 1: Indexed

Field 2,3: no properties

Field 4,5: Validation rule, Validation text

Table 17: Simple

Number of fields: 4 fields

Field 1,2,3,4: no properties

Table 18: Simple

Number of fields: 3 fields

Field 1: Indexed

Field 2,3: no properties

Table 19: Simple

Number of fields: 4 fields

Field 1,2,3,4: no properties

Table 20: Simple

Number of fields: 3 fields

Field 1,3: Indexed

Field 2: no properties

Table 21: Average

Number of fields: 7 fields

Field 1: Indexed

Field 2,3,4,5,6: no properties

Field 7: Validation rule, Validation text

Table 22: Simple

Number of fields: 7 fields

Field 1,2,3,4,5,6: no properties

Field 7: Input mask

Table 23: Simple

Number of fields: 5 fields

Field 1,2,3,4,5: no properties

Table 24: Simple

Number of fields: 2 fields

Field 1: Indexed

Field 2: no properties

Table 25: Simple

Number of fields: 9 fields

Field 1,2,3,4,5,6,7,8: no properties

Field 9: Input mask

Table 26: Simple

Number of fields: 2 fields

Field 1: Indexed

Field 2: no properties

Table 27: Average

Number of fields: 13 fields

Field 1,3,4,5,6,7,8,9,10,11,12,13: no properties

Field 2: Validation rule, Validation text

Table 28: Average

Number of fields: 12 fields

Field 1,3,4,5,7,8,9,10,11,12: no properties

Field 2: Validation rule, Validation text

Field 6: Indexed

Number of relationships: 10 Complex

Type of relationship: One-to-Many

Properties of relationship: all

Transactions: 51 Simple

Type of transactions: 41 Select queries

7 Delete queries

3 Update queries

Forms : 6 Simple

Reports : 14 Simple

	Simple		Average		Complex		Total
Tables	22	* 7 +	6	* 10 +		* 15 =	214
Relationships		* 2 +		* 3 +	10	* 5 =	50
Transactions	51	* 5 +		* 7 +		* 10 =	255
Forms	6	* 4 +	1	* 5 +		* 7 =	24
Reports	14	* 4 +		* 5 +		* 7 =	56
DBP							599

1. Does the system have an interface with Visual Basic? 0
2. What is the rate of the programmer's experience? 1
3. Did the programmer use Wizards and to what rate? 1
4. Does the system take networking into consideration? 0
5. Does the system provide Help for users? 0
6. Centralized or distributed processing? 0
7. Did the programmer use Visual Queries or Issue Queries? 0
8. Did the Access application use Windows API? 0

Effort = 599 * [0.2 + (0.01 * 2)]

= 599 * 0.22 = 131.78 Working Days

Table 5.1 Summary Table of the Results

Application	Component Functions	Estimated Effort (Days)	Real Effort (Approx.) (Days)	Error Margin
Hotel Management	1. Reserve a room 2. Ordering Food and Beverage	38.88	20	94.4%
LAU listening center	1. Student Clinic 2. Student Listening Schedule 3. Assistant Schedule 4. Instructor Schedule 5. Clinic Report 6. Listening Schedule Report 7. Memos 8. Assistant Schedule Report 9. Instructor Schedule Report	25.2	20	26 %
Pharmacy Application	1. Customer Order 2. Medicine Order 3. Drug Information 4. Customer Information 5. Employee Information	24.96	20	24.8 %
Restaurant Application	1. Company Information 2. Customer Application 3. Employee Application 4. Employee Shifts	25.2	20	26 %
Video Shop Application	1. Customer Files 2. Video Files 3. Make Order 4. Return Videos	45.54	20	127.7 %
Hospital Application	1. Accounting 2. Administration 3. Admission 4. Circulation 5. Inventory 6. Laboratory	131.78	90	46.42 %

As you have seen that the error margin between the estimated effort and the real effort was between 25 % and 127 %, which could be considered as an acceptable range. Some of the classic effort software metrics , as Cocomo and Function points, the error margin reached up to 200 %. Therefore, the error margin that we have

reached in this project was within the acceptable range and hence we can conclude that our results can be considered to be acceptable. Therefore, this aggregate database metrics could be applied to any project implemented using Access in order to estimate the effort and duration.

5.8. Summary Table of the Results

After applying the aggregate database metrics on a number of Access applications, a comparison between the estimated results obtained in the previous sections and the approximate value of the real effort is presented in Table 5.1. I

It is an approximate value of the real effort due to the following reasons:

1. The days that were counted did not have uniform number of hours. In aggregate database metrics, we assumed a working day to be of six working hours, but in reality it may be more or less than six.
2. When counting the number of days, we have to assume a 20 % error factor.

Chapter 6

Conclusion

In this project, we have proposed new aggregate database metrics that compute the effort and duration of small Access business application. We have presented a number of metrics that are considered to be the major attributes of this metric. We have also described in detail the categories, classification of each category and the weights corresponding to each class.

The idea of this project started by a need to search for complexity metrics on relational database languages. We borrowed the idea of the aggregate database metrics from the Function Points metrics.

This metric was established due to the following reasons:

1. No complexity or structural metrics could be applied to relational database languages.
2. A need to estimate the effort and duration of Access applications.

Finally, we applied these metrics on a number of applications, and the error factor between the estimated and the real effort ranged between 25% and 127 %, which could be considered as an acceptable range since the error margin of some classical complexity metrics (Cocomo or Function Points) reached up to 200%.

References

Amman M. and Cameron R., 1994, *Measuring Program Structure with Inter-Module Metrics,* Proceedings of the Eighteenth International Conference on Softawre Engineering (COMPSAC 94), Taipei, Taiwan.

Boehm, B. 1981, *Software Engineering Economics*, Prentice- Hall, USA.

Dreger B. 1989, *Function Point Analysis*, Prentice-Hall, USA.

Fenton N. E. 1993, *Software Metrics A Rigorous Approach*, CHAPMAN & HALL, London, UK.

Gifford D. 1998, *Access 97 Unleashed,* Sams, USA.

Henderson-Sellers B. 1996, *Object-Oriented Metrics Measures of Complexity*, Prentice Hall, New Jersy, USA.

Lorenz M. and Kidd J. 1994, *Object-Oriented Software Metrics*, Prentice-Hall , New Jersy, USA.

McCabe T.J. 1976, *" A Complexity Measure"*, IEEE Transaction on Software Engineering.

Mi Kim E., Chang, O.B. , Kusumoto S. and Kikuno T. 1996, *"Heuristics for Computing Attribute Values of C++ Program Complexity Metrics,"* ,Proceedings of the Twentieth International Conference on Softawre Engineering (COMPSAC 96), Seoul, Korea.

Prather R.E. 1996, *"Convexity and independence in software metric theory"*, Software Engineering Journal **10**.

Schach S. 1996, *Classical and Object-Oriented Software Engineering*, Irwin, USA.

Shepperd M. 1993, *Software Engineering Metrics*, McGraw-Hill International, UK.

Shen R., Conte S. and Dunsmore H. 1983, *" Software Science Revisted: A Critical Analysis of the Theory and Its Empirical Support"*, IEEE Transactions on Software Engineering, No.2.

Bibliography

Code

Lyon G. and Kacker R. 1994, *" A Simple Scalability Test for MIMD Code",* Proceedings of the Second International Software Metrics Symposium, London, England.

Design

Kang B. K and Bieman J. 1996, *"Design-level Cohesion Measures: Derivation, Comparison, and Applications"* ,Proceedings of the Twentieth International Conference on Softawre Engineering (COMPSAC 96), Seoul, Korea, 1996, pp. 92-97.

Model

Bastani F. B., DiMarco G. and Pasquini A. 1993, *"Experimental Evaluation of a Fuzzy-Set Based Measures of Software Correctness Using Program Mutation, ",* Proceedings of the Fifteenth International Conference on Software Engineering (COMPSAC 93), Baltimore, Maryland, 1993, pp.45-54.

Cogan B. I. and HunterR. B. 1996, *"Definition and collection of metrics for comprehensive software measurement",* Software Quality Journal **5.**

Module

Bharghavan V., Ramomorthy C. V. and Paul R. 1994, *"Simulation Based Metrics Prediction in Software Engineering,"* , Proceedings of the Eighteenth International Conference on Softawre Engineering (COMPSAC 94), Taipei, Taiwan.

Object-Oriented

Ghosh P. 1993, *"Software Metrics and Object-Oriented Systems"*, ACSOM 93.

Henderson-Sellers B. 1996, *Object-Oriented Metrics Measures of Complexity*, Prentice Hall, New Jersy, USA.

Lorenz M. and Kidd J. 1994, *Object-Oriented Software Metrics*, Prentice-Hall , New Jersy, USA.

Mi Kim E., Chang, O.B. , Kusumoto S. and Kikuno T. 199, *"Analysis of Metrics for Object-Oriented Program Complexity"* ,Proceedings of the Eighteenth International Conference on Softawre Engineering (COMPSAC 94), Taipei, Taiwan.

Nesi P. and Campani M. 1996, *"Metric Framework for Object-Oriented Real-Time Systems Specification Languages,"* ,The Journal of Systems and Software.

Paul R., Shinagawa Y., Day Y.F., Khan M.F.and Ghafoor A. 1996, *"Object-Oriented Framework for Metrics Guided Risk Management,"* , Proceedings of the Twentieth International Conference on Softawre Engineering (COMPSAC 96), Seoul, Korea.

Rubin H.A, 1993, *"Software process maturity: measuring its impact on productivity and quality"*, Proceedings of the Second International Software Metrics Symposium, London, England.

Wong B. and Verner J. 1993, *"Object-Oriented And Metrics"*, ACSOM 93.

Process

Cazabon Y. and Bauer M. 1995, *"A Policy Independent Metric for Process Selection in Distributed Systems,"* ,Proceedings of the Nineteenth International Conference on Software Engineering (COMPSAC 95), Dallas, Texas.

Program Complexity

McCabe T.J. 1976, *" A Complexity Measure"*, IEEE Transaction on Software Engineering.

Mi Kim E., Chang, O.B. , Kusumoto S. and Kikuno T. 1996, *"Heuristics for Computing Attribute Values of C++ Program Complexity Metrics,"* ,Proceedings of the Twentieth International Conference on Softawre Engineering (COMPSAC 96), Seoul, Korea.

Shen R., Conte S. and DunsmoreH. 1983, *" Software Science Revisted: A Critical Analysis of the Theory and Its Empirical Support"*, IEEE Transactions on Software Engineering, No.2.

Project Management

Paul R., Chee C.L., Kunii T. and Shinagawa Y. 1996, *"Data Models For Metrics-Based Project Management Systems,"* , Proceedings of the Twentieth International Conference on Softawre Engineering (COMPSAC 96), Seoul, Korea.

Risk Management

Hudepohl J.P., Aud S.T., Khoshgoftaar T.G. and Allen E.B. 1996, *"Emerald : Software Metrics and Models on the Desktop,"*,IEEE Software.

Structure

Amman M. and Cameron R., 1994, *Measuring Program Structure with Inter-Module Metrics,* Proceedings of the Eighteenth International Conference on Softawre Engineering (COMPSAC 94), Taipei, Taiwan.

Prather R.E. 1996, *"Convexity and independence in software metric theory"*, Software Engineering Journal 10.

Van Den Broek P.M. and Van Den Berg K.G. 1995, *" Generalised approach to software structure metrics"*, Software Engineering Journal **5.**

Testing

Liggesmeyer P. 1995, *"A set of commplexity metrics for guiding the software test process,"*, Software Quality Journal **4.**

Obara E., Kawasaki T., Ookawa Y. and Maeda 1997, *"Metrica And Analysis In The Test Phase of Large-Scale Software"*, The Journal of Systems and Software, Vol 38, No. 1.

Van Den Broek P.M. and Van Den BergK.G. 1994, *"Axiomatic Testing of Structure Metrics"*, Proceedings of the Second International Software Metrics Symposium, London, England.

Others

Blaine J. and Kemmer R. 1985, *"Complexity Measures for Assembly Language Programs"*, The Journal of Systems and Software, Vol. 5.

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

Dreger J.B. 1989, *Function Point Analysis*, Prentice-Hall.

Fairley R. 1992, *"Recent Advances in Software Estimation Techniques"*, International Conference on Software Engineering 1992.

Fenton N.E. 1993, *Software Metrics A Rigorous Approach,* CHAPMAN & HALL, London, UK.

Granja-Alvarez J.C. and Barranco-Gracia M.J. 1997, *"A Method for Estimating Maintenance Cost in a Software Project: A Case Study"*, Software Maintenance: Research and Practice, Vol. 9, No. 3.

Harrison W. and Cook C. 1987, *"A Micro/Macro Measure of Software Complexity"*, The Journal of Systems and Software, Vol. 7.

Kafura D. and Henri S. 1981, *"Software Quality Metrics Based on Interconnectivity"*, The Journal of Systems and Software.

Li H. 1987, *"An Empirical Study of Software Metrics"*, IEEE Transactions on Software Engineering, Vol. SE- 13, No. 6.

Lokan C. 1993, "Software Size Estimation: A Review", ACSOM 93.

Pollock G. and Sheppard S. 1987, *"A Design Methodology For The Utilization of Metrics Within Various Phases of Software Lifecycle Models"*, COMSAC 87.

Prather R.E. 1994, *"Axiomatic Foundation for Structural Software Metrics"*, Proceedings of the Second International Software Metrics Symposium, London, England.

Shepperd M. 1993, *Software Engineering Metrics*, McGraw-Hill International, UK.

Walrad G. and Moss E. 1993, *"Measurement: The Key to Application Development Quality"*, IBM Systems Journal, Vol. 32, No. 3.

Yan,Zhang Y. and Ma Q.1997, *"Software Support for Multiprocessor Latency Measurement and Evalution"*, IEEE on Software Enginerring , Vol 23, No. 1.

Yu W., Smith P. and Huang S. 1990, "Software Productivity Measurements", AT&T Technical Journal, Vol. 69, No. 3.

Zhenyu W. and Li C.1994, *"Ada Concurrent Complexity Metrics Based on Rendez Vous Relations"*, Proceedings of the Eighteenth International Conference on Softawre Engineering (COMPSAC 94), Taipei, Taiwan.