Software Metrics for Small Database Applications

Sana Abiad, Ramzi A. Haraty, and Nashat Mansour
Lebanese American University
P.O. Box 13-5053
Beirut, Lebanon
Email:{rharaty, nmansour@beirut.lau.edu.lb}

ABSTRACT

Known software metrics for estimating complexity and effort are usually based on lines of code or the program's flowgraph. Such metrics are suitable for large-scale procedural or object-oriented software applications. In this work, we propose a new complexity metric, called DataBase Points (DBP), that is suitable for small-scale relational database business applications developed in the MS-ACCESS (ACCESS is a trademark of Microsoft Corporation) or similar environments. DBP is constructed from components that are derived from typical ACCESS design. Further, DBP is used to estimate the effort needed to develop such software. The results of applying this new metric to a number of applications show that it is promising and that it captures the complexity features of small database applications.

KEYWORDS

Software Metrics, Function Points, Effort Estimation, and Complexity Metric.

1. INTRODUCTION

Effective software project management requires measuring attributes of the software process and product. Such measurement would make project managers better informed about issues such as delivery time, effort, cost, and resources. Software measurement is essential to good software engineering since "you can not control what you can not measure".

SAC'00 March 19-21 Como, Italy

Software metrics are based on measurement and are used to help developers predict, understand, control, and improve both the development process and the software product [6]. Many metrics have been proposed for design, quality, testing complexity, maintenance cost, and code complexity [9, 15]. In particular, complexity and effort estimation metrics include: McCabe's cyclomatic number [13], Halstead's software science [16], function points [2,5,11], COCOMO's technique [4], SLIM's technique [14], and regression-based models [3].

All these metrics apply to procedural or object-oriented large-scale software applications. Only very recently, small-scale applications were considered [8,11] and none of the known metrics apply directly to relational database applications. In particular, all metrics that are based on lines of code or program's control flowgraph are not applicable to database programs.

In this paper, we are concerned with small-scale business applications that are developed in the relational database MS-ACCESS environment. We propose a new metric, called DataBase Points (DBP), which can be used in the design phase. This metric reflects the functionality and complexity of an ACCESS application and is used to estimate the effort required for implementing such applications. We empirically evaluate this metric by applying it to a number of small-scale business applications and comparing its results with those obtained from function points and with the real measured effort. The results obtained are promising.

This paper is organized as follows. Section 2 constructs the DBP metric and shows its components. Section 3 presents the aggregate metric and the effort estimation expression. Section 4 gives the empirical results. Section 5 concludes the paper.

2. COMPONENTS OF THE DATABASE METRIC

The design of MS-ACCESS relational database applications is typically based on five categories: tables, relationships, forms, and reports [7]. These categories can be classified as simple, average, or complex. We use these categories and their classification to construct a metric, called database points (DBP), for the complexity of MS-ACCESS applications. The five components and the factors that contribute to their complexity classification are briefly

described in the following subsections. For a more detailed description of these factors, see [1].

2.1. TABLES

Tables are the objects that store data in a 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 (Simple, Average, and Complex) are the following:

- Number of Fields per Table: It is the number of fields defined in each table. The number of fields is a measure of the effort and time a programmer may spend on setting a certain table. As the number of fields increases, so do the time and effort of a programmer.

- Properties per Field: It is the type of properties that are defined for each field. Some properties are set by default. At the stage of translating the Entity Relationship diagram, the properties of the field can be set. Field properties can be divided into two groups. The first group helps in defining the overall picture of the field, while the second enforces restrictions on the field as a whole. Some properties will increase the complexity of this category more than others will.

- Table Properties: It is the type of properties that are defined for each table. Some properties will increase the complexity than others.

2.2. RELATIONSHIPS

Relationships correspond to the distinct relations connecting any two tables defined in the Entity Relationship diagram. Defining a relation between two tables include defining the fields that make the relation, setting the type of the relationship and setting the properties that define these relations. The factors that influence the classification of this category are the following:

- Type of Relationship: Any relationship defined in ACCESS could be one of the following: A) One-to-One which is used when information has to be kept for each record, usually in numerous fields. B) One-to-Many which reflects a hierarchy of details, with a parent record containing information about a group of items and each item having details stored in a separate table. C) Many-to-One which reflects a hierarchy of details.

- Properties of Relationship: It is the type of properties that are defined for each relationship. Relationship properties enforce restrictions on the related table. Some properties will increase the complexity of this category more than others will. These properties include A) Enforce Referential Integrity which ensures that relationships between records in related tables are valid. This prevents the user from accidentally deleting or changing related data. B) Cascade Update Related Fields which affects the change of the "many" side of a relationship, so the records in the detailed table are not left orphaned. C) Cascade Delete Related Records, which when selected, the deletion of the record on the "one" side forces the deletion of the records on the "many" side.

2.3. TRANSACTIONS

Transactions correspond to 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 Transactions: In ACCESS, there are a big number of transactions. Even though several different variations of transactions may be created, there are only two types of transactions that could be 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.

2.4. FORMS

Forms correspond to the number of distinct forms that are defined in any database. They 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. Users can perform a large number of modifications to the resulting form, and these changes are the main metrics at this step.

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, and adding graphics. The factor that determines the classification of this category is the type of modifications done to any form.

Forms can be created using wizards or without wizards. The majority of forms that we encountered and the easiest way of setting a form are 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.).

2.5. REPORTS

Reports correspond to the number of distinct reports that are defined in a database. Defining a report includes 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.

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.

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 deal with the properties of the reports and adding expressions that are concerned with the display of the fields. The most difficult changes are those concerned with adding programmable events to the report.

3. AGGREGATE DATABASE METRIC AND EFFORT ESTIMATION

3.1. DATABASE POINTS AND EFFORT

The five categories of database applications, which are described in the previous section, are grouped together to construct an aggregate complexity metric, which we call DataBase Points (DBP). Table 1 shows how the DBP metric is computed, where the software engineer has to fill in the required values based on the design information. Clearly, our methodology for determining the DBP is analogous to that for determining the well-known Function Points [5]. In a similar way, we use DBP to estimate the effort needed for implementing the database software. The effort involves eight adjusting factors that are derived from typical ACCESS programming. This effort expression is

$$\text{Effort} = \text{DBP} * (0.2 + 0.01 * \Sigma F_i) \qquad (1)$$

where F_i is the weight given to each of the adjusting factors (i = 1, 2, ..., 8). These factors are explained in the next subsection. The implementation effort includes coding and testing and is given in Person-Days (8 hours day).

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 (Sum)							

Table 1 Computing the DBP Metric.

The weights shown in Table 1 are borrowed from the Function Point metric with some adaptation to database software. These weights can be considered as the time unit a programmer spends when setting a category. These weights are adapted as follows:

- Weights for Tables were more than weights for other categories, since setting a table includes several steps. While Relationships category was given less weights, since defining a relationship requires less time.

- 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 they only include 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, all these weights are summed up in Equation (1). This equation resembles the Function Point formula [5] but with modification to the constants. To empirically determine these constants, we collected six ACCESS applications for

which we knew the effort required from the programmer. Then, we applied suitable weights to Table 1 to compute DBP and used best-fit techniques on the effort-application curve to set the coefficients in Equation (1) to 0.2 and 0.01.

3.2. ADJUSTING FACTORS

Based on the nature of ACCESS database programs, we use eight adjusting factors. Each of them can be given a weight that ranges from 0 to 5, where the software engineer determines the particular value. The factors and the associated weights are based on the following questions:

- F_1. Does the system have an interface with Visual Basic or any other interface?
- F_2. What is the rate of the programmer's experience in ACCESS?
- F_3. Did the programmer use wizards and to what rate?
- F_4. Does the system take networking into consideration?
- F_5. Does the system provide help for users?
- F_6. Is the system designed for centralized or distributed processing?
- F_7. Did the programmer use Visual Queries or Issue Queries?
- F_8. Did the Access application use Windows APIs?

These factors add to the complexity and the effort required to implement an ACCESS application. Clearly, they involve aspects that apply directly to Access 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 (1); hence, this sum could vary from 0 to 45.

4. EMPIRICAL RESULTS

To test the DBP metric, we use six small applications developed by senior university students for different businesses. In this section, we consider one application (hospital application) in detail and then tabulate the results for six applications. We compare the DBP results with those of function points and the real effort.

Table 2 summarizes our results for six ACCESS applications. It gives a comparison between the estimated effort based on DBP, the approximate real effort, and the effort based on Function Points. All the efforts are in Person-Days.

Table 2 shows that the error in the estimated DBP-effort ranges from 24.8% to 127.7% for the considered set of applications. This is an acceptable range for two reasons. Firstly, it is well known that effort estimation errors are large and different estimation models can yield values that are enormously different [10]. Secondly, the set of applications used for best fitting the coefficients' values in Equation (1) is small. Thus, large errors in the results are expected.

However, the DBP-effort estimates are consistently better than the Function Point-effort estimates. This validates our hypothesis that known metrics are not suitable for small database projects, whereas the DBP metric is designed specifically for such projects.

Our results in Table 2 involve a small set of projects. Clearly, we need a larger set to tune the values of the coefficients in Equation (1) (currently 0.2 and 0.01). This will allow better validation of the DBP-effort estimation.

Application	Real Effort	DBP Effort	DBP Error %	FP Effort	FP Error %
Hotel	20	38.88	94.4	68.9	244.5
LAU Lab.	20	25.2	26	57.85	189.3
Pharmacy	20	24.96	24.8	73.45	367.3
Restaurant	20	25.2	26	68.25	241.3
Video Shop	20	45.54	127.7	88.4	342
Hospital	90	149.8	66.4	245.7	173

Table 2 Summary of the Results.

5. CONCLUSION

We have proposed a new metric, called Database Points (DBP), for the complexity and effort estimation of small database business software projects. This metric is constructed from components that make up typical relational database programs.

We have applied the DBP-based effort equation to a few business applications and compared its estimate with the real effort as well as function points-based estimate. The results show that DBP is more suitable than function points for estimating the effort required for database applications and that the DBP-based effort values lie within the usual margin of error. Therefore, the DBP results are promising, although further work is required to improve the values of the coefficients used in the effort equation. Employing a large set of projects in best fitting the coefficient values can do this.

REFERENCES

[1] Abiad S. 1999, Software Metrics for Small Database Applications, Master's Thesis. Lebanese American University, Beirut.

[2] Albrecht A.E. and Gaffney J.E. 1983, "Software function, lines of code, and

development effort prediction: a software science validation", IEEE Transactions on Software Engineering, 9(6), pp. 639-647.

[3] Bailey J.W. and Baisili V.R. 1981, "A meta-model for software development resource expenditure", Proceedings of the 5th Int. Conference on Software Engineering, IEEE Computer Society Press, pp. 107-116.

[4] Boehm B.W., Clark B., Horowitz E. et al. 1995, "Cost models for future life cycle processes: COCOMO 2.0", Annals of Software Engineering 1, pp. 1-24.

[5] Dreger B. 1989, Function Point Analysis, Prentice-Hall.

[6] Fenton N. E. and Pfleeger S. 1996, Software Metrics A Rigorous Approach and Practical Approach, International Thomson Computer Press, London.

[7] Gifford D. 1998, Access 97 Unleashed, Sams.

[8] Grable R., Jernigan J., Pogue C., and Divis D. 1999, "Metrics for small projects: Experiences at the SED", IEEE Software, March/April, pp. 21-29.

[9] Henderson-Sellers B. 1996, Object-Oriented Metrics Measures of Complexity, Prentice Hall, New Jersey.

[10] Kemerer C. 1987, "An empirical validation of software cost estimation models", Communication of the ACM, 30(5), pp. 416-429.

[11] Kautz K. 1999, "Making sense of measurement for small organizations", IEEE Software, March / April 1999, pp.14-20.

[12] Matson J., Barrett B., and Mellichamp J. 1994, "Software Development Cost Estimation Using Function Points", IEEE Trans. Software Engineering, Vol. 20, No.4, April 1994, pp. 275-287.

[13] McCabe T.J. 1976, "A Complexity Measure", IEEE Transaction on Software Engineering, 2(4), pp. 308-320.

[14] Putnam L.H. 1978, "A general empirical solution to the macrosoftware sizing and estimating problem", IEEE Transactions on Software Engineering, pp. 345-361.

[15] Shepperd M. 1993, Software Engineering Metrics, McGraw-Hill International.

[16] Shen R., Conte S. and Dunsmore H. 1983, "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support", IEEE Transactions on Software Engineering, No. 2, pp. 155-165.

BIOGRAPHY

Sana Abiad received her B.S. degree in Computer Science from the American University of Beirut, and her M.S. degree in Computer Science from the Lebanese American University. Her research interests include database systems and software engineering.

Ramzi A. Haraty is an Assistant Professor of Computer Science at the Lebanese American University in Beirut, Lebanon. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database systems, artificial intelligence, and multilevel secure systems engineering. He has well over 35 journal and conference paper publications.

Nashat Mansour is an Associate Professor of Computer Science at the Lebanese American University in Beirut, Lebanon. He received his B.E. and M.S. degrees in Electrical Engineering from the University of New South Wales, Australia, and M.S. in Computer Engineering and Ph.D. in Computer Science from Syracuse University, New York. His research interests include software testing, metrics, and maintenance, and evolutionary algorithms. He is a member of IASTED and is on the executive committee of the Lebanese Association for the Advancement of Science.