Defining and Measuring Maintainability

R. Cheaito, M. Frappier, S. Matwin, A. Mili
Department of Computer Science
University of Ottawa
Ottawa, Ont. K1N 6N5
fax: (613) 562 5800 X 5188

D. Crabtree
Space Station Program
Canadian Space Agency
St Hubert, PQ, J3Y 8Y9
fax: (514) 926 4576

e-mail contact: amili@csi.uottawa.ca

Research Paper

Abstract

Maintainability is a measure of the ease with which a software system can be maintained. In this paper we propose a quantitative definition of maintainability, and discuss means to estimate the maintainability of a software system on the basis of a static analysis of its deliverables. Such an estimation effort is useful inasmuchas it allows us to predict the maintainability of a software system at delivery time.

1 Background: Motivation and Context

All the quantitative studies that have been carried out in the past two decades indicate that maintenance has a staggering financial impact on the software industry, and that it is a drain on human resources. Typical studies provide that on average a software organization spends sixty percent (60 %) of its man-month resources on software maintenance, and forty percent (40 %) on the development of new applications. Because of the increasing reliance on software in various application domains, new applications are being put into operation at a faster rate than older applications are phased out; as a result, the relative impact of maintenance is expected to increase in the future –although at a more moderate rate than was initially feared.

This state of affairs, as well as this trend, are clearly cause for concern. The key to dealing with this concern is to identify the factors that affect the costs of maintenance, quantify them, and endeavor to control them in the framework of software maintenance management. In keeping with this premise, the *Canadian Space Agency* is sponsoring a research project at the University of Ottawa [10, 11, 12, 13], with the objective of defining a measure of maintainability and investigating means to evaluate it.

The Canadian Space Agency participates in the *Space Station* program by providing the *Mobile Servicing System*; part of this system is the *Manipulator Development Simulation Facility*, which the agency is contracting out to private software firms. The intent of the agency is to impose maintainability requirements on future deliveries of software that is contracted out. To do so, the agency must provide a definition of maintainability (as part of the requirements specification), as well as means to estimate the maintainability of a software deliverable at delivery time (as part of acceptance testing).

In the next section we analyse the costs of software maintenance, identify the major controllable cost factors, and discuss how these can be controlled in the case of the MDSF project within CSA. In section 3 we introduce the measure of maintainability, which reflects the ease with which an

application can be maintained; then we show that this measure has an important impact on the cost of maintenance, and that it can be controlled. In section 4.2, we show how, in the context of the MDSF project, this measure can be certified at delivery time (for the benefit of CSA) and can be estimated at design time (for the benefit of CSA contractors). Finally in section 5 we present prospects of further work leading to a quantitative assessment of maintainability during the software lifecycle.

2 Software Maintenance Costs

The activity of software maintenance is traditionally divided into three categories [3, 18]:

- *corrective maintenance*, which deals with correcting faults in the code of the program to make the program satisfy its functional requirement. Modifications in this category are triggered by an observation of failure in the operation of the program
- *adaptive maintenance*, which deals with changing the program to reflect changes in the requirements. Modifications in this category are triggered by a change in the requirements specification, typically at the request of the user.
- *perfective maintenance*, which deals with improving the performance or design of the program while preserving its function. Modifications in this category are typically triggered by the maintenance staff, in the framework of periodic reviews.

The cost of maintaining a software product (measured in man-months) over some period of time can be analyzed as the product of two quantities: the frequency with which we must make modifications (measured in number of occurences); and the average cost of a modification (measured in man-months). Formally,

$$MM_{MAINT} = COST_{MODIF} \times FREQ.$$

Taking into account the three categories of software maintenance, we rewrite this formula as:

$$MM_{MAINT} = COST_{MODIF} \times (FREQ_{COR} + FREQ_{ADP} + FREQ_{PER}).$$

In the context of CSA's MDSF project, the term $FREQ_{COR}$ is determined by the reliability of the product, and can be controlled by imposing standards of reliability as part of the requirements; the terms $FREQ_{ADP}$ and $FREQ_{PER}$ can also conceivably be controlled, but there is ample incentive not to restrict them, so as to run an efficient application ($FREQ_{PER}$) and keep the specification flexible ($FREQ_{ADP}$). Hence $COST_{MODIF}$ is a prime candidate for monitoring and control in CSA's delivered software; all the more so that it is factored multiplicatively in the formula above. This factor is the focus of the next section.

3 Defining Maintainability

A software application may satisfy all the traditional lifecycle requirements (correctness, reliability, efficiency, user-friendliness, performance, etc..) and yet be of little use if the cost of maintaining it is prohibitive. Hence we consider maintainability as one of the requirements that must be imposed on software products. In this section we introduce a measure of maintainability then discuss the use of this measure in managing the maintenance of the MDSF software within CSA. We have derived our measure on the basis of the following premises:

1. We emphasize the *definition* of a measure of maintainability rather than the tools and techniques to measure it. We have found that whenever the question of *what is* maintainability and the question of *how to measure* maintainability are confused, the former becomes too strongly influenced by the latter, and the discussion soon evolves around the theme: How can software metrics help us predict maintainability?

2. The measure must be quantifiable; this is an essential requirement if we want to use it for rational decision making and for certifiable verification.

3. The measure must be intrinsic to the software product, and must be independent of the maintenance process (e.g. must be independent of whether maintenance programmers are competent, whether users submit frequent change requests, what software tools are used). In the *Intermediate COCOMO* cost model [3], a distinction is made between two families of cost factors:

   - Cost factors that are *intrinsic to the product*, such as $KDSI$ (the product size), $RELY$ (the required reliability), $DATA$ (the database size, whenever applicable), $CPLX$ (the product complexity).
   - Cost factors that reflect features of the software process, such as development machine attributes, personnel attributes, and project attributes.

   In keeping with this premise, we will normalize our effort estimates by adjusting them for the process-related cost factors. We refer to this adjusted effort as *intrinsic effort*.

4. The measure must be orthogonal to other (well defined) features of the software product, such as, e.g. reliability, correctness. We know for example that the more a software system is reliable, the less it will cost to maintain; yet we want to factor reliability out of our measure of maintainability, for the sake of orthogonality. Hence *reliability* and *maintainability* will represent two orthogonal features of the software (viz. how often the software fails, and how much each failure costs) rather than (somehow) twice the same feature.

5. The measure must form the basis for comparisons across projects (e.g. "A is more maintainable than B") even when the projects at hand deal with different applications. Hence we could not define maintainability as a mere function of the cost of a typical modification ($COST_{MODIF}$), because the same cost may seem reasonable or excessive depending on the service provided by the software. To obtain a meaningful measure of maintainability we must normalize the cost of a modification for the functionality of the software.

In light of these premises, we define the maintainability of a software product by the following formula:

$$MNT = \frac{FNC}{COST_{MODIF}},$$

where FNC is a measure of the functionality of the software and $COST_{MODIF}$ is the average intrinsic effort it takes to carry out a modification of the software system. We can think of three measures for FNC, which yield three distinct measures for MNT:

- FNC can be measured in function points; then maintainability is measured in function points per man month. The larger the maintainability, the fewer man months it takes to carry out a typical modification of a software product with a given functionality. For example, a system whose functionality is one thousand function points (the functionality of a healthy data base management system) and whose maintainability is 4000 fp/MM will require an average of five man days to perform a modification ($1/4MM = 5MD$).

- FNC can be measured in lines of code; then maintainability is measured in lines of code per man month. The number of man months required to carry out a modification on the software is then $SIZE/MNT$, where $SIZE$ is the size of the software. Hence, e.g. a product of size $100kloc$ whose maintainability is $200kloc/MM$ requires an average of 10 man days to perform a modification ($1/2MM = 10MD$).

- FNC can be measured by the intrinsic effort required to develop the software product; then maintainability is measured by an abstract number (ratio of man months over man months). If a software product has a maintainability of, say, 1000 (i.e. the intrinsic effort to make an average modification is one thousandth the development effort of the product), and a development cost of three hundred man months (typical for a $100kloc$ product) then the cost of carrying out a modification is on average seven man days ($1/3MM = 7MD$).

The MDSF system is the first major software product of the MSS program; CSA must maintain this one-million-line system over a period of fifteen years. Needless to say, good management of the maintenance process is crucial, given the stakes at hand. The introduction and use of a measure of maintainability in this proces is quite essential, for the following reasons.

- Using the definition of maintainability, CSA can now incorporate quantitative maintainability requirements in the software development contracts it awards external software developers.

- Using the methods of estimating maintainability at delivery time, CSA can now certify whether the maintainability requirements have been met by the contractor.

- Using the methods of estimating maintainability at design time, the CSA contractors can now monitor the expected maintainability of their product, and adjust it as needed.

- Using the methods of estimating maintainability at delivery time and beyond, CSA can now make specific quantitative planning of maintenance activities, and can determine whether a maintenance problem is product related (software product has low maintainability, and has outlived its useful period) or process related (programmers are not up to maintenance task, software tools are inadequate, etc..).

- Using the methods of monitoring maintainability, CSA can now incorporate maintainability into its organization-wide standards, and thereby ensure graceful aging of its applications (i.e. a controlled evolution of its software maintenance costs).

Now that we have determined what is maintainability, we discuss in the next section how to measure it.

4 Measuring Maintainability

The maintainability of a software product is defined as the ratio of the functionality offered by the product over the average intrinsic effort to carry out a modification on the product. Because the average intrinsic effort to carry out a modification on the product cannot be computed as such before the product is put in operation and observed over a long period of time, the measure of maintainability cannot be evaluated until well after the product has been acceptance-tested and delivered. Yet, for the sake of both the customer (in our case, CSA) and the software developer (in our case, the CSA contractors), it is useful to determine the maintainability of the product (or an estimate thereof) as early as the phase of acceptance testing (i.e. upon delivery of the product).

Hence we resort to estimating equations: specifically, the measure of maintainability is estimated on the basis of quantitative observations made on the deliverable software assets. Section 4.1 discusses the various metrics that we propose to use in our estimating equations and section 4.2 discusses how we propose to derive estimating equations on the basis of statistical data.

4.1 Software Metrics

Given how we have defined the measure of maintainability, and what we mean it to represent, we expect that this measure reflects such features of the software product as: the simplicity of its design, the integrity of its structure, the modularity of its decomposition, the completeness and clarity of its documentation, etc...

Several authors have established a statistical correlation between maintainability and standard software metrics [21, 22, 24, 17]. A study by Shepperd and Ince [23] showed that modules with high information flow have a 600 % greater probability of a residual error as a consequence of a change than modules with low information flow. Kafura and Reddy [15] have determined that there is a relationship between the metrics of a given body of software, and its maintenance charcteristics. Their study is subjective: they have shown that the quantitative measures defined by the software metrics are consistent with the judgement of the experts who are initimately familiar with the software being studied. Procedures with unusually high complexity were found to be the source of many maintenance probles as well. Rombach [21] has investigated the effect of two different classes of metrics on maintainability. Internal complexity metrics are related to the length and structure of modules; external complexity charcterize inter-modular flow, including implicit flow. The basic finding of [21] is that a combined metric, taking into account both internal and external complexity, is the best predictor of the maintenance effort measured in person-hours. Henry and Selig [14] have gone one step further, proposing a procedure that estimates the quality of software (understood as its complexity measure) at the design stage. This procedure applies metrics on designs, written in an Ada-like Program Design Language. Results of this work, combined with the subjective results of [15] and the empirical results of [21] indicate that maintainability could be estimated at program design time, if the design is represented at the right linguistic level. More recently, [17] reported that the earlier results linking maintainability to software metrics, obtained mainly for Fortran and Pascal code, extend to code in OO languages. Clearly, appropriate OO metrics have to be used in the evaluation.

In a recent paper [7], Coleman *et al* proposed two measures of maintainability. Their first measure, HPMAS, is based on a decomposition of maintainability into three dimensions: control structure, information structure and readability (typographic, naming and commenting). For each dimension, they identify a number of metrics with their acceptable range of values. The maintainability of a dimension is the weighted sum of the proportional deviation from the acceptable range. A dimension maintainability varies between 0 and 1, the latter being the most maintainable. The overall maintainability is the product of the three dimension maintainability results. Weights were calibarated to match a subjective evaluation of maintainability conducted by software engineers using the abridged AFOTEC guide [1].

The second measure is an estimation of the AFOTEC numerical evaluation using four metrics: Halstead's Volume(V), extended cyclomatic complexity (EVG), lines of code (LOC) and number of comments (CMT). The formula is:

$$Maintainability = 171 - 5.2\ln(V) - 0.23EVG - 16.2\ln(LOC) + 50\sin(\sqrt{2.46CMT})$$

Coefficients were obtained using a linear regression.

The most significant distinction between this model and ours is the definition of maintainability itself. Ours is an objective measure based on maintenance costs. For us, the essence of maintainability resides in its impact on maintenance costs. Coleman's definition is based on a subjective evaluation of maintainability using the AFOTEC guide. This guide consists in a series of questions regarding different software attributes which were deemed to influence maintainability. To evaluate maintainability, an expert is asked to rate each question on a scale of 1 to 6. Although experience and judgement are valuable for selecting attributes, we have no idea how effective it is at predicting the overall impact on maintainability when all attributes are combined. For instance, one may deem that a module length over 100 LOC, or a cyclomatic complexity greater than 10 EVG, are detrimental to maintainability. However, it is very difficult to evaluate the impact of a module with 500 LOC and 2 EVG. Does the low EVG compensates for the high LOC? What is the connection between the two measures? If this knowledge was available to software engineers, then the task of building cost models would be a great deal easier.

Another distinction is that our model allows ratio comparisons whereas Coleman's allows only ordinal comparisons. For instance, given two systems of the same size, if we compute a cost of modification of 10 person-days for the first, and a cost of 20 person-days for the second, it is meaningful to say that the first system is twice as maintainable as the second. Using similar values in Coleman's models, it seems difficult to conclude that the ease of making a change is twice as much. All we can infer is that the first system is easier to maintain, but not by how much. Ratio comparisons are critical when maintainability must be evaluated with other conflicting quality attributes, such as efficiency or usability, for selecting a software system. For instance, one may be willing to gain twice as much in efficiency if maintainability is only reduced by a factor of 2 or less.

In [11], we have identified a number of metrics as potentially correlated to our measure of maintainability; we present these below, along with a short rationale for each. We focused our attention on four deliverables of the development process: the Contract End Item Specification (CEIS), which is the first level of specification for the MDSF system, the Requirements Specification (RS), the Software Preliminary Design Documents (SPDD), the Software Detailed Design Documents (SDDD) and the Source Code (SC). We identified attributes of these deliverables which we consider to have an impact on maintainability. For each attribute, we selected a set of software metrics.

Unreferenced Requirements The number of requirements not referenced by a lower document in the documentation hierarchy. It measures the *traceability* of the documentation. Traceability is an important aspect of maintainability. It helps in determining the requirements applying to a module or the modules associated to a requirement. The metric applies to CEIS, RS, SPDD and SDDD.

Non Referencing Items The number of items not referencing a requirement in an upper document of the documentation hierarchy. It is the dual of the previous measure.

COCOMO Product Cost Factors The combination of the COCOMO product cost factors $RELY$ (the required reliability), $DATA$ (the database size, to the extent that it is meaningful) and $CPLX$ (the product complexity).

Module Coupling This measures the flow of information *betwen* modules. It is based on the tables provided by Myers [20]. Only the nature of the coupling between module is addressed, not the quantity of information. Maintainability should decrease as coupling increases. This measure is computed on SDDD or source code.

Module Cohesion This measures the flow of information *within* a module. It is also based on the tables provided by Myers [20]. Maintainability is expected to decrease as cohesion increases. This measure is computed on SDDD or source code.

Design Complexity A measure of intermodule and intramodule complexity of a system based on fan-out and global variables [6]. Many studies have shown that information flow measures of this type are correlated with maintainability [21, 23]. This measure is computed on SDDD or source code.

Cyclomatic Complexity Number This well-known measure represents the number of independent basic paths (covering all segments of code) in a module [19]. The more basic paths a module has, the more difficult it is to comprehend and test.

Knots A measure of the number of crossing lines (unstructured goto statements) in a control flow graph. A study by Blaine and Kemmerer [5] shows a significant correlation between this measure and the effort required to change a module.

Comments Volume of Declarations The total number of characters found in the comments of the declaration section of a module. The declaration section comprises comments before the module heading up to the first executable statement of the module body. Several studies have shown that comments affect comprehensibility and readability of a module [16, 25]

Comments Volume of Structures The total number of characters in the comments found anywhere in the module except in the declaration section.

Average Length of Variable Names This is the mean number of characters of all variables used in a module. Unused declared variables are not included. The experiment of Jorgensen [16] shows the importance of variable name length in module understanding.

Lines of Code The number of lines in the source code of a module excluding blank lines or comment lines. This well-known metric has been the subject of many criticisms when used as a measure of *productivity*. However, it is a good indicator of the effort required to change a module [8, 21].

Documentation Accuracy Ratio A verification of the accuracy of the CEIS, RS, SPDD and SDDD with respect to the source code. A sample of modules is randomly selected. For each module, an expert verifies that the requirements applicable to the module are those implemented in the source code. Weights are associated to various components of a specification like inputs, outputs, validations and algorithms. The average sum of the weights is computed for the set of modules selected. Needless to say, documentation accuracy is a key aspect of maintainability, especially when maintenance must be done over a long period of time with a high turnover of maintenance personnel.

Consistency This measures the extent to which the documentation and source code contains uniform notation, terminology and symbology within itself [2]. It is computed by counting, for a sample of programs and documents, the number of inconsistencies like, e.g., using the same name for different entities, using different names for the same entity or using different structure of comments for program headings. Consistency should improve the understandability of documents and programs.

4.2 Model Building

In our analysis of software maintainability we had come to the conclusion that our measure of maintainability is quite dependent on the structure of the software product (modularity, design integrity, size, complexity, etc..). Consequently we strongly expect that our measure be statistically correlated to the software metrics that reflect structural properties; these include in particular the traditional source-code based metrics such as volume, effort, development time, cyclomatic complexity, etc... We are interested in building an estimation model that allows us to predict the maintainability of a software product from readings of its source code metrics.

To this effect we have conducted a survey whose object is to consider a set of programs currently under operations and maintenance, enquire about their average cost per modification (hence their maintainability), and collect information on their structural metrics. The survey was conducted in the winter 1995 in the Ottawa area, involved nineteen (19) software projects, and had the following characteristics:

- The size of our projects varies between 400 lines of code and 6000 lines of code, all in Pascal.
- The date of development ranges from 1987 to 1991.
- The application domains include systems programming, systems simulation, and systems administration.
- These applications were developed and maintained by local telecommunications companies or software development companies.

The metrics data was collected using *PC-Metric*, ©*Set Laboratories, Mulino, Oregon.*

Using the data collected, we propose to derive an estimation model for the measure of maintainability; our intent is to be able to predict the maintainability of the software product at delivery, on the basis of readings we get from PC-Metric. Our initial proposal is to consider that the cost of maintaining a software product depends on the same factors, and in the same manner, as the cost of developing this product. We referred to the COCOMO model for guidance and proposed that the cost of a single modification takes the form

$$\alpha \times Size^{\beta},$$

where $Size$ is some measure of the size of the software (lines of code, software science length, volume, etc..). Also, following intermediate COCOMO [3], we wish to take into account the product factors that affect development (and maintenance cost), namely: RELY, CPLX, DATA [3]. Because RELY cannot be determined by static analysis of the program source and DATA is not relevant to our sample of projects, this leaves only CPLX; so that we propose to model the cost of a single modification by the equation:

$$COST_{MODIF} = CPLX \times \alpha \times (Size)^{\beta}.$$

The COCOMO model [3] provides criteria for assigning ratings to the complexity of a software component, and maps these ratings to numeric values. We have decided to assign these ratings on the basis of the average cyclomatic complexity of procedures and functions (which is given by PC-Metric under the name AVG; we call it AVG) in the software product. The table below shows the mapping between COCOMO's complexity ratings and the AVG metric, as well as the effort multiplier that corresponds to the rating in question.

AVG	COCOMO rating	Multiplier
1	very low	0.70
2	low	0.85
3	nominal	1.00
4	high	1.15
5	very high	1.30
6	extra high	1.65

Because the cyclomatic complexity scale is an open scale (can go on to to 7, 8, 9, 10, etc..) whereas COCOMO's complexity ratings are limited, we chose to define the effort multipliers as a direct function of AVG, so that they can be estimated for any value of AVG. We propose the following formula, and leave it to the reader to check that it provides a reasonable approximation of the table above:

$$CPLX = 1.15^{AVG-3}.$$

Using this formula for complexity and taking the software science length (N) for variable $Size$ (ref: the equation of modification cost above), we perform a least square regression in order to determine α and β. The results of the regression are rather discouraging, since we find an R^2 very near zero (=0.001439). This suggests that $COST_{MODIF}$ cannot be estimated by the equation we propose.

Our second proposal is based on the premise that, in fact, the cost of a modification is dependent not on the size of the whole software, but the size of the software that had to be read, analyzed and modified to perform the modification. If the program is divided into subprograms, then typically modifications occur on the scale of a subprogram. So that rather than to involve the size of the whole software, we involve the average size of a subprogram, which can be determined by dividing the size of the product by the number of subprograms (we call it NS); this latter quantity is given by PC-Metric. If we take the software science length (N) as the measure of size, our formula for the cost of a modification becomes:

$$COST_{MODIF} = (1.15)^{AVG-3} \times \alpha \times (\frac{N}{NS})^{\beta}.$$

We perform a least square regression to derive α and β, and find $\alpha = 6.32$, $\beta = -0.79$ and $R^2 = 0.308$. While it is not quite satisfactory, the value of R^2 is substantially better than that we had earlier, suggesting that the premise of our second proposal is well founded; note also that when we use COCOMO's table for CPLX rather than the approximation $CPLX = (1.15)^{AVG-3}$, we find $R^2 = 0.404$. This yields the following formula for the cost of modification:

$$COST_{MODIF} = (1.15)^{AVG-3} \times 6.32 \times (\frac{N}{NS})^{-0.79}.$$

In our discussions about defining maintainability, we had suggested that there are many ways to define maintainability, depending on how we measure functionality (function points, lines of code, effort to develop). For the sake of this discussion, we propose to measure functionality by the count of lines of code. This yields the following estimation equation for the measure of maintainability:

$$Maint = 0.16 \times LOC \times (1.15)^{3-AVG} \times (\frac{N}{NS})^{0.79}.$$

All the parameters involved in this formula, namely LOC, AVG, N and NS can readily be evaluated at delivery time; in fact they can be evaluated automatically. Hence this measure of maintainability can be evaluated at delivery time. Table 1 gives the estimates of maintainability for the projects in our sample and compares them to the actuals. Note that residuals are quite reasonable: 56 % of residuals are less than 0.20; the average absolute value of residuals is 0.20.

Actual Maintainability	Estimated Maintainability	Residual (abs. value)
880.20	1079.37	0.23
3330.00	4278.41	0.28
1837.15	1832.70	0.00
2061.70	3056.67	0.48
8859.82	8043.00	0.09
6431.56	5897.97	0.08
8842.86	8295.50	0.06
3640.00	1835.17	0.49
9222.00	7301.65	0.21
4701.27	3183.87	0.32
837.07	987.14	0.18
4044.73	4929.16	0.22
8943.87	9486.80	0.06
17008.56	20277.48	0.19
1599.07	1502.07	0.06
6638.18	7904.15	0.19

Figure 1: Maintainability: Estimates vs Actuals

5 Conclusion and Prospects

Une Science a l'age de ses instruments de mesure.
Louis Pasteur

Software maintenance accounts for a large and increasing part in software costs across the software industry; software maintenance costs must necessarily be monitored and controlled if the software industry is to keep offering new products (as it is expected). For the *Canadian Space Agency* and similar organizations, the stakes involved in monitoring and controlling software maintenance costs are even higher than the industry at large because the agency maintains very large one-of-a-kind systems over long periods of time.

In this paper we propose an original measure of maintainability, as the ease to maintain a software product, and investigate means to predict/estimate the maintainability of a product from features of its intermediate (and final) deliverables. Our measure can be characterized by the following premises:

- It is quantifiable; as such it can be used to define enforceable and certifiable maintainability requirements, as well as to compare candidate software solutions on the basis of maintainability.
- It is intrinsic to the software product, and does not depend on the software maintenance process. As such, it can be used to determine whether maintenance costs are due to the product (unmaintainable product) or to the process (inadequate maintenance procedures).
- It is orthogonal to other features of the software product, such as e.g. *reliability* (with which it is sometimes equated). Hence, it can be used, e.g. to determine whether software costs are due

to poor reliability (too many required modifications) or to poor maintainability (individual modifications are too costly).

In addition to the definition of maintainability, we have also given a quantitative formula for estimating the maintainability of a software product on the basis of traditional metrics that can be derived from the source code at delivery time. Our estimation model is based on a small sample of projects; we envisage to extend our investigation to a larger sample, involving larger software products. Also, we envisage to refine our cost estimation model further, to obtain more accurate predictions.

References

[1] *Software Maintainability — Evaluation Guide*, AFOTEC Pamphlet 800-2 (updated), HQ Air Force Operational Test and Evaluation Center, Kirkland Air Force Base, N.M., Vol. 3, 1989.

[2] Boehm, B.W., J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod and M.J. Merrit. *Characteristics of Software Quality*, TRW series of software technology, North-Holland, (1978).

[3] Boehm, B.W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall, 1981.

[4] Benzecri, J.P. *L'Analyse de Données. Tomes 1 et 2*. Third edition. Paris, France: Dunod. 1979.

[5] Blaine, J.D. and R.A. Kemmerer. Complexity Measures for Assembly Language Programs, *Journal of Systems and Software*, 5, pp. 229-245, (1985).

[6] Card, D.N. and W.W.Agresti. Measuring Software Design Complexity. *Journal of Systems and Software*, 8(3), pp. 185-197, (1988).

[7] Coleman, D., D. Ash, B. Lowther, P. Oman. Using Metrics to Evaluate Software System Maintainability, *IEEE Software*, August 1994, pp 44–49.

[8] Davis, J.S. and R.J. LeBlanc. A Study of the Applicability of Complexity Measures. *IEEE Trans. on Soft. Eng.*, SE-14(9), pp.1366-1372, (1988).

[9] Diday, E. *Eléments de l'Analyse de Données*. Paris, France: Dunod, 1983.

[10] Frappier, M., S. Matwin and A. Mili. *Maintainability: Factors and Criteria*, Software Metrics Study, Tech. Memo. 1, Canadian Space Agency, St-Hubert, Canada, (1994).

[11] Frappier, M., S. Matwin and A. Mili. *Software Metrics for Predicting Maintainability*, Software Metrics Study, Tech. Memo. 2, Canadian Space Agency, St-Hubert, Canada, (1994).

[12] Frappier, M., S. Matwin and A. Mili. *Maintainability and Model Building Techniques*, Software Metrics Study, Tech. Memo. 3, Canadian Space Agency, St-Hubert, Canada, (1994).

[13] Frappier, M., S. Matwin and A. Mili. *A Proposal for Evaluation and Estimation of Maintainability*, Software Metrics Study, Tech. Memo. 4, Canadian Space Agency, St- Hubert, Canada, (1994).

[14] Henry, S. Selig, C., Predicting Source-Code Complexity at Design Stage, *IEEE Software*, pp. 36-45, March 1990.

[15] Kafura, D. Reddy, G.R., The Use of Software Complexity Metrics in Software Maintenance, *IEEE Trans. on Software Engineering*, vol. SE-13, pp. 335-343 (1987).

[16] Jorgensen, A.H. A Methodology for Measuring the Readability and Modifiability of Computer Programs, *BIT*, 20, pp. 394-405, (1980).

[17] Li, W. and S. Henry. Object Oriented Metrics that Predict Maintainability. *Journal of Systems and Software*. Vol 23, pp 111-122 (1993).

[18] Lientz, W and B. Swanson. *Software Maintenance Management*. Reading, Ma: Addison Wesley, 1978.

[19] McCabe, T.J. A Complexity Measure, *IEEE Trans. on Soft. Eng.*, SE-2(4), pp. 308-320, (1976).

[20] Myers, G.J. *Reliable Software Through Composite Design*. New York, Van Nostrand Reinhold, (1975).

[21] Rombach, H.D. A Controlled Experiment on the Impact of Software Structure on Maintainability. *IEEE Transactions on Software Engineering*. Vol SE-13, pp 344-354, (1987).

[22] Rombach, H.D. Design Measurement: Some Lessons Learned. *IEEE Software*, pp 17-25 (1990).

[23] Shepperd, M. and D. Ince. Design Metrics and Software Maintainability: An Experimental Investigation, *Journal of Sofware Maintenance: Research and Practice*, 3(4), pp. 215-232, (1991).

[24] Wake, S. and S. Henry. A Model Based on Software Quality Factors which Predicts Maintainability. *Proceedings, Conference on Software Maintenance*, 1988, pp 382-387.

[25] Woodfield, S.N., H.E. Dunsmore and V.Y. Shen. The Effect of Modularization and Comments on Program Comprehension, *Proceedings, 5th International Conference on Software Engineering*, pp. 215-223, (1981).

[26] Zaiem, H. *Les Méthodes Exploratoires de l'Analyse des Données*. Université de Tunis I, Institut supérieur de l'Education et de la Formation Continue. 1988.