December 1997

Assessment Model for Software Maintenance Tools: A Conceptual Framework

M. Khan
Universiti Brunei Darussalam

M. Ramakrishnan
Universiti Brunei Darussalam

Bruce Lo
Southern Cross University

Assessment Model for Software Maintenance Tools: A Conceptual Framework

Md. Khaled Khan
M. K. Ramakrishnan
Department of Mathematics
Universiti Brunei Darussalam
BSB 2028; Brunei Darussalam
email: khaled@ubd.edu.bn
Fax:: 673-2-249502

Bruce Lo
Centre for Computing and Mathematics
Southern Cross University
Lismore, NSW, Australia 2480
email: blo@scu.edu.au
Fax: +61 66 221724

Executive Summary
With the rapid development in the field of software engineering and the increasing recognition of the complexity of the software maintenance process, the quality of maintenance tools and the procedure to measure those quality factors will need to be established. There are two aspects involved in selecting quality maintenance tools: to establish relevant tool-selection criteria and to provide methods for measuring these criteria. This work deals mainly with the second aspect. The main objective of this paper is to introduce an assessment model for the selection of software maintenance tools in terms of quality and productivity. In this regard, a modified multi-element component comparison and analysis (MECCA) model is proposed.

The paper suggests that three major steps should be followed before the actual tool assessment process. These steps are: *defining the maintenance goals, gathering information about the tools*, and *selecting candidate tools*. The maintenance goals should include not only the technical needs but the economic and strategic aspects as well. The criteria for tool selection will then be based on these maintenance goals.

The proposed assessment model has three major components: a hierarchy of attribute-classes, a percentage weighting scheme, and a numeric scoring for the final attribute subclasses. The attributes of the maintenance tools can be categorised into different attribute-classes and each of these attribute-classes can have subclasses according to the structuring scheme. The structure can be modified or reorganized according to the practitioner's specific preferences and project goals. The ultimate attribute subclasses are assigned numeric scores which measure their performances and functionality. The final score will be calculated for a specific maintenance tool by means of a weighted average. The scores then can be used to rank a given set of tools. Finally, a tool with the highest rating will be selected for the maintenance project.

To perform this assessment process, no tools expert is needed. This model allows information systems practitioners to define their own needs and preferences according to the specific application of their projects. The flexibility of this model will easily allow the practitioners to express their decisions easily. However, this approach is not just limited to maintenance tool assessment, rather the method may be used for evaluating other software tools as well.

1. Introduction

Software maintenance has increasingly been recognised as occupying a central role in the software process. In many IS departments, maintenance problems increase rapidly with the passage of time. Software maintenance has been considered as the major cause of software

development backlogs in the past (IEEE conferences 1990-1994). The natural question is why software maintenance problems are so difficult to deal with? The answer lies on various factors such as lack of management understanding and supports, incompatible design documents of the system, lack of well-defined maintenance methodology, and lack of assessment procedures in selecting appropriate maintenance tools.

It is generally accepted that appropriate maintenance tools can have significant impacts in assuring the quality of a modified information system and the productivity of the system maintenance process. Recent literature (IEEE conferences 1990-1994) indicate that software practitioners are paying increasing attention to software maintenance tools. It is recognised that developers can achieve greater system maintenance productivity and reliability by employing appropriate maintenance tools. It is also noted that in some cases software maintenance projects were not fully successful due to the use of inappropriate maintenance tools. Too often wrong tools are used to solve maintenance problems. Even right tools can be used to solve the wrong maintenance problem (Champoli 1990). Organisations may have a fair knowledge of the strengths and weaknesses of various classes of general purpose software tools, but for maintenance purpose they must fully understand the specific purpose tools. In the past, it was often assumed that maintenance had only to do with correcting errors. This restricted view of maintenance proved inadequate because software changes may arise due to changes in the technology, growing complexity of the systems, new requirements from the users and changes in the business environment (Fuggetta 1993). It is, therefore, important to set up a systematic approach that will allow the maintainer to assess a tool for specific application and maintenance situations. There are several CASE tools available with some maintenance functions. Some are good for some applications while some others perform well in other situations. It is obviously risky to select a maintenance tool in an arbitrary manner. In this paper, we attempt to introduce a systematic assessment approach that is both flexible and easy to apply.

2. Maintenance Tools

In software maintenance a wide range of tools is used such as code analyzer, reverse engineering tools, program debugger, program slicer, testing tools, CASE tools, interactive maintenance tools etc. These tools may be classified differently. The reader is referred to the articles by Fuggetta, Sharon and Chumra(1993,1994) for a more detailed discussion of the classification of software engineering tools. At present relatively few commercial maintenance tools are available on the market. However, there are considerable research efforts being directed in this area.

To evaluate maintenance tools, it is necessary to examine the main functions of these tools. Generally speaking, the functionality of software maintenance tools may include the following among others:

- Program understanding: This is an important function of a maintenance tool. This function can provide two types of program knowledge: syntactical knowledge, and semantic knowledge. Syntactical knowledge contains, for example call sequences, structured charts, control structure, variable declarations etc. Semantic knowledge consists of a description of the purpose of modules, purpose of calls, intentions of code fragments and so on. Program understanding tools can be further subdivided into two types: analysis-oriented tools, and code-oriented tools. Mayrhauser (1990) has provided fruther treatment on this subject.
- Ripple effect analysis: When a system or a part thereof is modified, secondary effects on the system are not avoidable. Ripple effect analysis aims to trace the extent of these secondary effects. This function should not only trace the side effects in the program due to code modification, it should also be able to analyse the "ripple" effects at a higher abstraction level, for example among the design specifications.
- Software retesting: Software retesting functions are not very different from the original software testing during the initial software development phase. Such tools should incorporate knowledge gained in previous testing as well as information provided by the ripple effect analysis process. Thus it should include regression testing, targeted path testing, stress testing and so on.

- Knowledge redocumenting: A clear and precise documentation of the modified system is crucial to software maintainability. A major problem that faces many legacy systems is the lack of adequate documentation of the system modification. This function helps to update existing documents to ensure that all changes to the software are properly reflected in its documentation.
- Information repository: Two types of information are important: history database which provides a recorded history of the system modifications, and semantic database which preserves the integrity and currency of the system characteristics.
- Display function: This provides the capabilities to display in textual and graphical format of the program information as a result of the maintenance modification. The user will also be able to navigate or browse the existing program structure using maintenance tools. The tool must have the capability to represent and examine data and program structure from different viewpoints.
- Tool interconnections: According to IEEE Standard 1175 (1991) tool interconnections are considered important during tool evaluations. Tool interconnections affect how a tool works in an organisation, i.e. how it communicates with other existing tools used by the organisation. If the tool interconnection aspects are neglected in the tool evaluation phase, the likelihood of successful tool adoption will be minimised.

Based on a classification scheme similar to the above, it is clear that the functions of any maintenance tool can be represented as a multi-level structure illustrated in Figure 1 in the next page.

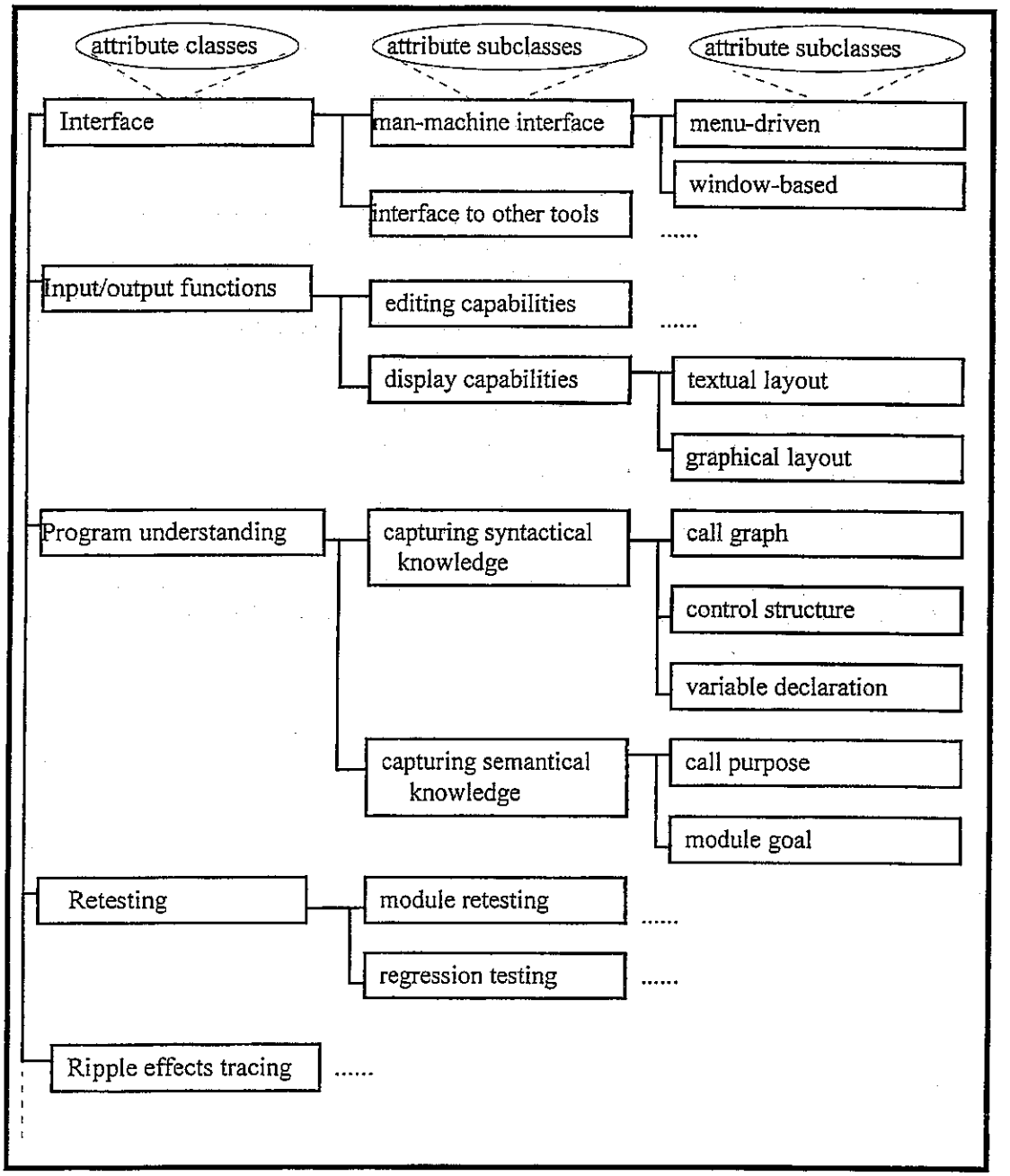

Figure 1: Structure of Attribute Classes and Subclasses of Maintenance Tools

However, not all of these functions are available in a single maintenance tool. Therefore, it is important to select the tool which serves best to the maintenance programmers for a specific types of application. We shall now proceed to propose an evaluation and selection procedure.

3. Three important considerations

The following three steps are important in the evaluation of maintenance tools.

Step 1: *Definition of maintenance goals.* The first and most important step is to identify the exact requirement of the project. Different projects may have different objectives depending on the domain of application, complexity of the system, and the development environment. To evaluate a maintenance tool, it is necessary to measure the tool's capabilities against the actual project requirements which are rephrased as the maintenance goals. This may then

lead to a clearer definition of the goals of software maintenance in the context of the current project.

It has been reported that in many cases, organisations purchase software tools without a clearly defined set of goals (Chikofsky 1988). Users as well as designers sometimes either do not know or fail to specify what services or functions they actually expect from the system.

A well-planned goal analysis will prevent various types of purchasing mistakes. There are two important aspects to be considered: what the organisation means by "maintenance" and what "activities" are to be included in this process. In establishing the maintenance goals, quality and productivity thresholds should also be defined. The quality-assurance department in the organisation may be consulted for this purpose, because they may have already compiled the needed productivity and quality statistics (Poston and Sexton 1992).

Once the maintenance goals are established, then the criteria for tool selection should be defined. The criteria based on the maintenance goals should include the essential characteristics of the tools required for the maintenance purpose. The criteria must clarify the purposes, constraints and rationale of the maintenance project. These must comprehensively focus on the quality, functionality and the performance of the candidate tool (Bockle et al. 1996).

In addition to these technical factors, economic and strategic aspects should also be considered -because they often determine the success of a product (Bockle et al. 1996). It includes cost of the tool, budget available, training costs and so on. The costs vary from vendor to vendor and tool to tool. For example, low-end personal-computer software can now be purchased for far less than $1 per function point, whereas mainframe based software tools are available for less than $100 per function point (Jones 1994).

Step 2: *Gathering of information about the tools.* In this step, the precise information about the available maintenance tools should be obtained in terms of their functionality. Such information can be collected from the sales brochures, advertisements, manuals, actual demonstrations and related literature. Detailed information about all the available functions of each tool should be recorded and categorised. It is also important to find out the tradeoffs of different classes of maintenance tools from different software companies. The demo disks from the vendors may be collected, and these should be used to get a better understanding of the tool.

Another significant aspect often missing during the information gathering process is the future applicability of the tool. The organisations must find out how flexible the tool is for changes to the future applications (House 1995).

The functions or tool's attributes can be divided into two groups, namely, mandatory attributes and optional attributes.

Step 3: *Selection of candidate tools.* In this step, a subset of the available tools are selected for evaluation purpose. The procedure involves comparing the goals established previously in ***step 1*** with the information obtained in ***step 2***. At this stage only those tools which satisfy the project needs will be selected. However, very detailed and thorough evaluation is not envisaged in this step.

All these three steps are illustrated in figure 2.

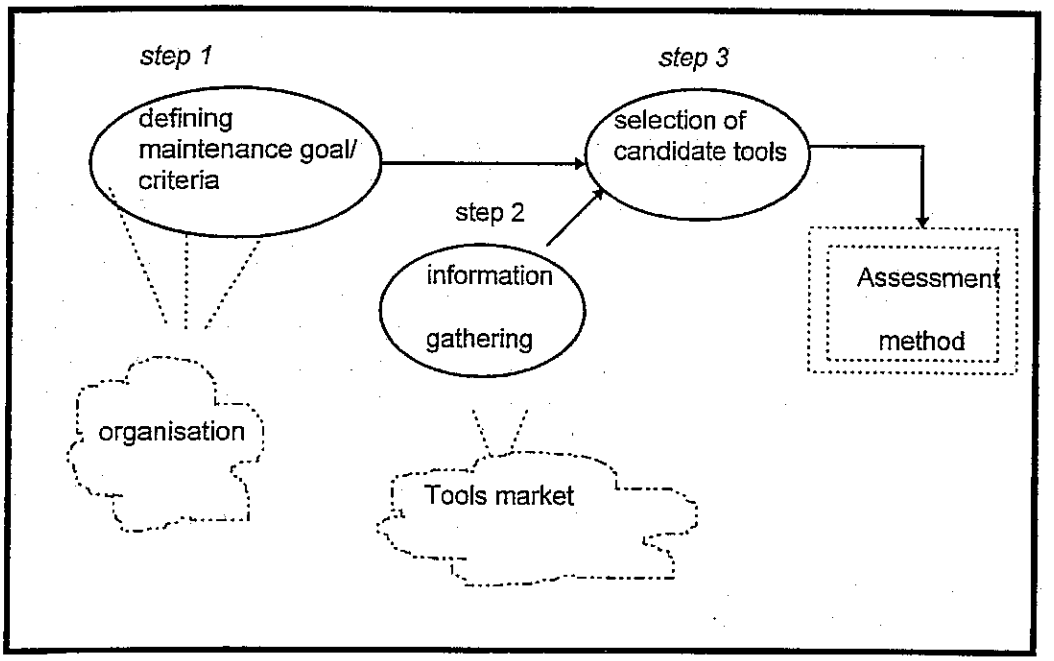

Figure 2: Three Steps in the Evaluation of Tools.

4. Preprocess

A preprocess is carried out to decide whether a tool is qualified for the final evaluation process. The available attributes of the maintenance tools, such as interface, functionalilty, input/output etc., can be categorised into different attribute- classes and each of these attribute-classes can have subclasses as well. Figure 1 shows an example of such a categorisation structure with most common maintenance tools attributes. Note that, the attribute-class "interface" is further divided into two attribute-subclasses, namely man-machine-interface and tool interface while the attribute-subclass "man-machine interface" is itself further divided into two subclasses; "menu-driven" and "window-based". The structure can be modified or reorganised according to the evaluator's preferences and project goals. However, it is important that the same structuring scheme can be used for assessing different maintenance tools in a specific evaluation process.

A typical structuring scheme is outlined in Figure 3. Each available attribute in a specific tool is assigned a weight depending on its importance; a higher weight reflects a higher importance. The mandatory attributes will fall in one category and the optional ones in another. Each selected tool will be assessed individually. Whenever mandatory attributes are not present in a specific tool, the tool should not pass the evaluation threshold. Only those tools which have passed the preprocess would be included for the final evaluation according to the model outlined below.

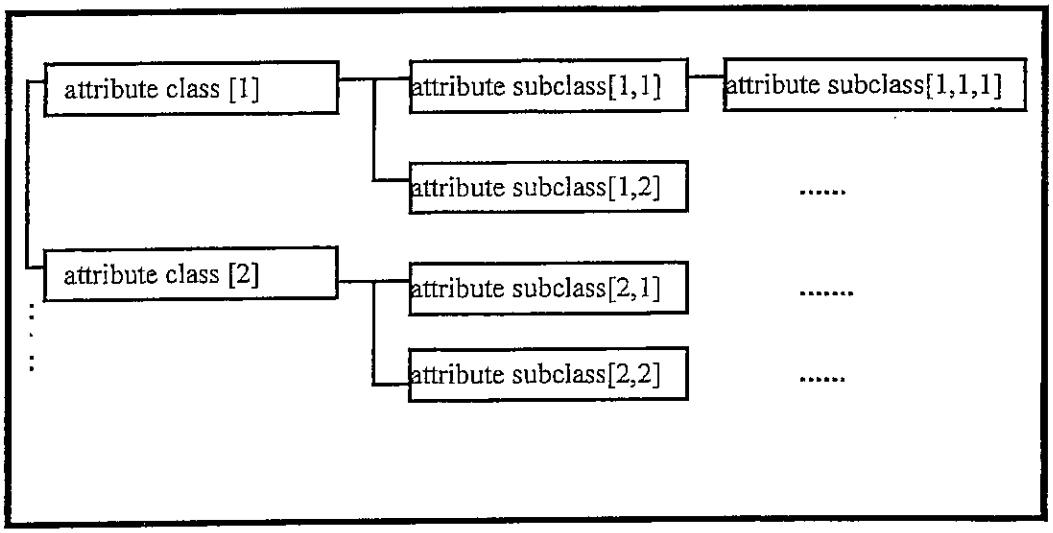

Figure 3: Attribute Structuring Scheme

5. The MECCA -model

The Multi-Element Component Comparison and Analysis (MECCA) method has been used as a system evaluation technique (Glib 1977). The underlying idea was first proposed by Zangemeister (1970) and it is now extensively used in decision science. The method was also adopted in the assessment of reverse engineering tools (Skramstad and Khan 1992) but the mathematical notations for the method were not developed at that time.

The method uses percentage weighting and a hierarchy of attribute-classes as shown in Figure 5. A percentage weight is assigned to each component attribute-class throughout the structure. The ultimate attribute-classes are also assigned numeric scores which measure their performance. The MECCA model has three essential ingredients: a *hierarchy of attribute-classes, a percentage weighting scheme,* and *a numeric scoring* for the final attribute-subclasses. It is clear that for each stage, the sum of the weights should add up to 100. The assessment process is guided by a short informal list of attributes to consider.

We can now compute the final score for a specific maintenance tool by means of a weighted average as described below. Assigning weights on attributes and the calculation procedures are considered an easy but important activity because it brings tool users to a consensus view on which requirements are most important for them. The final scores can be used to rank a given set of tools.

Computational method

We will represent various stages of the hierarchy by the letters i, j, k,..., m, n. For example, suppose that there are T attribute classes in the first stage. We will denote them by [i], i = 1, 2, ..., T. If the i-th attribute class is divided into N_i sub-attribute classes in the second stage, we will denote them by [i,j], j = 1, 2, ..., N_i. Likewise k will denote the third stage which refers to the N_{ij} sub-attribute classes of the j-th second stage of the i-th first stage and they will be denoted by [i,j,k]. Continuing like this, the last stage of the hierarchy is represented by n and the $N_{ijk....lm}$ ultimate attribute-subclasses of the penultimate subclass [i,j,k,...l,m] are denoted by [i,j,k,...,l,m,n].

The weights for each stage will be denoted by W with the corresponding suffixes. Thus, W_i : i = 1,2,...,T will denote the T first stage weights, W_{ij} : j = 1,2,...,N_i will denote the N_i second stage weights etc., and $W_{ijk..lmn}$ will be the last stage weights. The final individual scores for the attribute-subclasses of the last stage are denoted by $Y_{ijk..lmn}$.

Our ultimate weighted score is computed in an iterative way starting from the last stage. A dot '.' in the suffix indicates summation with respect to that suffix and a bar on the top refers to weighted mean score for that stage. The computation scheme is shown below in Figure 4:

$$\bar{Y}_{ijk\ldots lm\,.} = \sum_{n} W_{ijk\ldots lmn}\, Y_{ijk\ldots lmn}$$

$$\bar{Y}_{ijk\ldots l..} = \sum_{m} W_{ijk\ldots lm}\, \bar{Y}_{ijk\ldots lm.}$$

$$\bar{Y}_{ijk\ldots ...} = \sum_{l} W_{ijk\ldots l}\, \bar{Y}_{ijk\ldots l..}$$

.....

$$\bar{Y}_{i.. ...} = \sum_{j} W_{ij}\, \bar{Y}_{ij.....}$$

$$\bar{\bar{Y}}_{... ...} = \sum_{i} W_{i}\, \bar{Y}_{i.....}$$

Figure 4: Computation Scheme

The last one gives our final *weighted mean score.* If we combine them, we can write a single formula for the final weighted mean score which of course will look rather clumsy. Thus

$$\bar{\bar{Y}}_{... ...} = \sum_{i} W_i \sum_{j} W_{ij} \ldots \sum_{m} W_{ijk\ldots lm} \sum_{n} W_{ijk\ldots lmn}\, Y_{ijk\ldots lmn}$$

$$= \sum_{i}\sum_{j}\ldots\sum_{m}\sum_{n} W'_{ijk\ldots lmn}\, Y_{ijk\ldots lmn}$$

where $W'_{ijk\ldots lmn} = W_i W_{ij} W_{ijk} \ldots\ldots W_{ijk\ldots lmn}$.

Example

Consider the MECCA model given in Figure 5 below where the percentage weights are also shown. It is clear that there are 11 ultimate subclasses in the hierarchy for which numerical scores have to be assigned based on the information gathered in

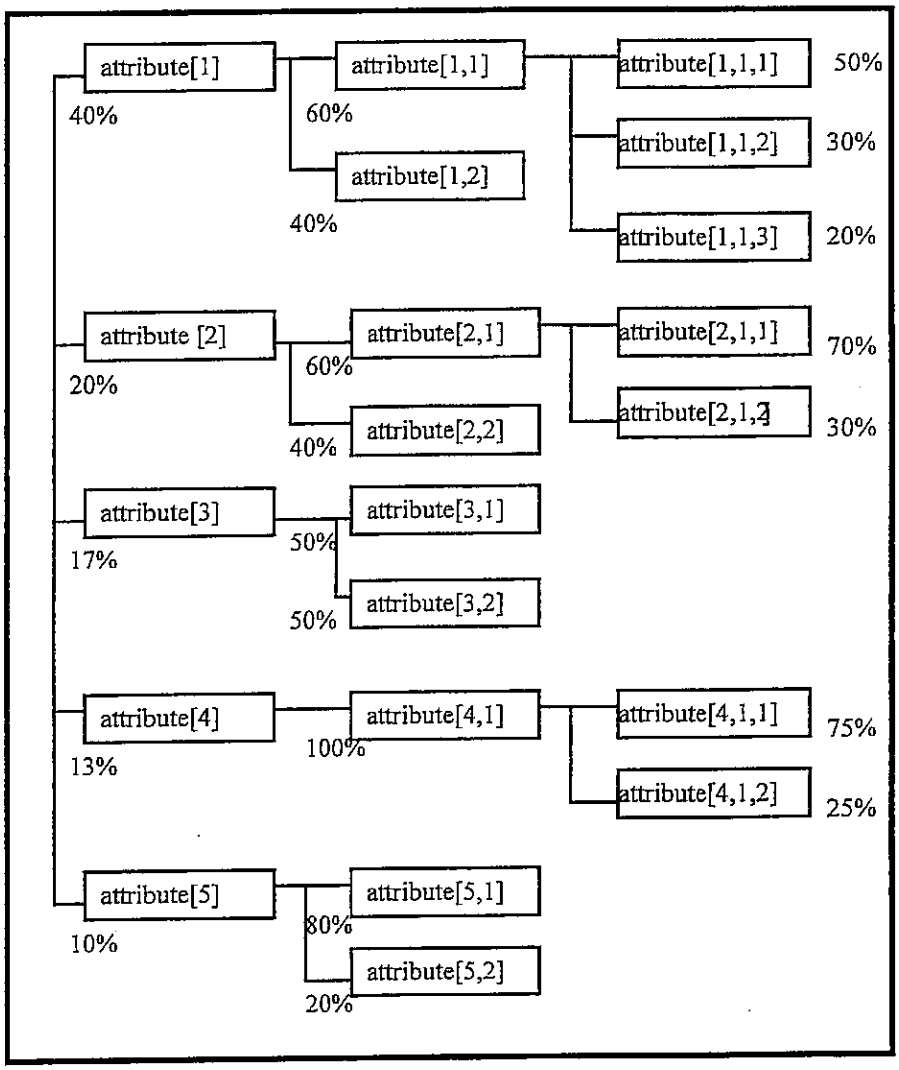

Figure 5: An example of MECCA Model

step 2 discussed earlier in *section 2* about the candidate tool. The numeric score may range from 0 (poor) to 10 (excellent) with 5 representing the average. Once the lowest level attributes have been assigned scores, the primary evaluation process is complete. Then the weights can be calculated according to the percentage assigned to the each attribute component. For example in Figure 5, if attribute[3,1] is given a score of 6 and attribute[3,2] is assigned 5, then attribute[3] at the highest level of the hierarchy will have a score (6 * 0.6) + (5 * 0.4) = 5.6. Thus, the contribution of the attribute[3] in the final evaluation will be 5.6 * 0.17 = 0.952.

In this fashion, each chosen tool will be assessed and given a rating. The tool with the highest rating will be finally selected for the maintenance project.

The process of assigning numeric values to the attributes will depend on the information gathered about the specific tool. So considerable amount of attention should be given in collecting information on the candidate tools at the earlier stages. So ***step 2*** discussed in the earlier section is the most significant phases in this evaluation process.

6. Conclusion

We have introduced an approach aiming at a more systematic evaluation of maintenance tools. To perform this assessment process, no tools expert is needed. This model allows practitioners to define their own needs and preferences according to the specific application of

their project. The flexibility of this approach will allow the practitioners to express their evaluation decisions more easily. The proposed model is not absolute or limited. The attribute-classes and their subclasses can be reorganised according to the development situation and the complexity of the candidate software. Even new component attributes can be identified and included in the attribute structure hierarchy. However, this approach is not just limited to maintenance tool assessment, rather the method may be used for evaluating other IS tools as well.

References

Bockle, G.; Hellwagner, H.; Lepold, R.; Sandweg, G.; Schallenberger, B.;Thudt, R.; Wallstab, S.; "Structured Evaluation of Computer Systems" *IEEE Computer*, June 1996, pp. 45-51.

Champoli,R.; "Does Maintenance Really Exist?" *Newsletter* , European SIG in Software Maintenance 1990.

Chikofsky, Elliot J.; "How To Loose Productivity With Productivity Tools" in Chikofsky, E. J. (ed.), *CASE*; IEEE Computer Society press, 1988.

Fuggetta, A.; "A Classification of CASE Technology", *IEEE Computer*, Dec. 1993, pp. 25-38.

Glib,T.: *Software Metrics*, Cambridge MA, Winthrop, 1977.

House, R; "Choosing the Right Software For Data Acquisition", *IEEE Spectrum* May 1995, pp. 24-39.

IEEE Standard 1175/1991. *A Trial-Use Standard Reference Model for Computing System Tool Interconnections*, IEEE Standards Office, NJ., 1991.

Jones, C; "Build, Buy, or Outsource?" *IEEE Computer*, Dec. 1994, pp. 77-78.

Zangemeister, C.: *Nutzwertanealyse in der Systemtechnik.* Eine Methodik zru multidimensionalen Bewertung und Auswahl von Projektaltermativen; Munchen, 1970.

Mayrhauser, A.von;"We need Code Processing or Should CASE Care About Maintenance?" *Proceedings of CASE '90 Workshop* 1990. pp. 20-21.

Poston, R.M; Sexton, M. P.; "Evaluating and Selecting Testing Tools" *Proceedings Symposium on Assessment of Quality Software Development Tools*, IEEE Computer Society press, 1992, pp. 55- 64.

IEEE, *Proceedings of Conferences on Software Maintenance*, IEEE Computer Society Press, 1990-94.

Sharon,D.;"Software-Engineering Tool Classification", *IEEE Software*, Sept, 1993, pp. 106-109.

Sharon D.; Chumra, A.; "Tool-Classification Scheme Revisited", *IEEE Software*, July 1994, pp.122-125.

Skramstad,T.; Khan, M. K.; "Assessment of Reverse Engineering Tools: A MECCA Approach", *Proceedings of IEEE. Symposium on Assessment of Quality Development Tools*, New Orleans, 1992, pp. 120-126.