4th International Conference on Automation, Control Engineering and Computer Science (ACECS - 2017)
Proceedings of Engineering and Technology – PET
Vol.19, pp.52-55

The reconciliation of external and internal quality attributes of ISO using the OO quality concepts

Zineb BOUGROUN, Mohammed SABER and Toumi BOUCHENTOUF
Electronic, Computer and Image Systems Research Laboratory (LSE2I),
National School of Applied Sciences (ENSAO), Mohammed first university (UMP)
60050 Oujda, Morocco
Bougroun.zineb@gmail.com, mosaber@gmail.com, tbouchentouf@gmail.com.

Abstract— **Problem statement: The IT project assessment issue has been much studied to improve the quality in order to reduce the maintenance cost. In this area, to present software quality, many models have appeared, among which ISO 25000. In other side, to evaluate software quality, several metrics have appeared. The two axes do not have a clear link to assert a complete evaluation.**
Approach: In this paper, we present the main classifications; especially, our approach based on OO programming properties. What we have done to know all the characteristics evaluated by the metrics. Then we applied these properties on the ISO 25000 model in order to reconcile internal and external quality attributes. To validate our work we conducted a survey, answered by experienced developers and analysts in the OO.
Results: The result of the survey shows a strong correlation towards our approach.

***Keywords*— metrics, properties of OO programming, quality model, ISO 25000, internal quality attributes, external quality attributes.**

I. INTRODUCTION

Maintenance and enhancement of software products consume a major portion of the total life cycle cost. Rough estimates shows that each category of maintenance consumes a range as high as 75-80 percent of whole project programming resources. Maintenance and enhancement tend to be viewed by management as at least somewhat more important than new application software development, that why we must give more attention to this phase of life cycle.

To reduce the cost of software maintenance, it is essential to care about the quality of software. For measuring this quality, several metrics have emerged to assess it [1-5]. Moreover, several model are appeared to present the software quality [6, 7]. Our research is included in this aspect, focusing on measures and quality model applied to object-oriented systems.

The internal quality attributes are presented in the literature by the quality metrics. Which developers use to know the quality level of their source code. The internal quality attributes, are structured in various models, used to present the quality of the final product and to market it. These two axes, despite being connected, this relationship was not explicitly formulated to give a clear and precise evaluation way.

Our work in this paper consists in bringing together the two axes of research: attributes of external quality and attributes of internal quality.

In order to perform this work we classify the metrics according to quality properties of object-oriented programming. Then we applied this classification to the ISO 25000 model. And, we validated this work by a survey replied by the developers and the researchers experienced in the field.

This paper is composed as follows: the first section shown the state of metrics classification, the second present our approach to classify metrics, thereafter we explain how we reconciled the internal and external quality attributes of ISO 25000, then we present the survey achieved in this context.

II. STATE OF ART OF METRICS CLASSIFICATION.

A. Classification by kind

This classification divides metrics into two parts: traditional and oriented object.

Traditional [12]: This kind of metrics is applied in the methods of a class. Two types of traditional metrics are described in this section: cyclomatic complexity and size.

Object-oriented [11,13]: Since the appearance of the object-oriented paradigm, various metrics that have been developed deal the principles of OO programming. They apply mainly to the concepts of class: coupling, cohesion, size and inheritance. Chidamber & Kemerer introduced these metrics in 1994 when the six metrics of the Moose project were invented.

B. Classification by level

Several researchers have adopted this classification [14].
Sheetz [15] has defined four levels of classification: the variable level, the level of the method, the level of the object, and the level of application. It has adopted its own metrics for each level.
However, Henderson [16] considered Object Oriented perspectives :

- **Interclass level:** we find in this level the size and complexity measurements.
- **Level of the class:** concerns the interface of the classes, the metrics of this level can be considered as measures of the services offered by the class.
- **System level (ignoring relationships):** measurements from the previous two levels are accumulated at this level. Such as the size of the system.
- **Level of relations system (excluding inheritance):** coupling is the main measure in this level.
- *Inheritance level:* In this level we find the measures of the inheritance hierarchy of a system and the resulting complexity.

ISSN 2356-5608

C. Classification of Gurdev, Dilbag and Vikram Singh

In this classification, the Singh brothers [17] showed the two types of metrics: **process and product.**

Process metrics are known as management measures, used to measure the properties of the production process.

Product metrics are quality measures, used to measure the properties of software. This type was presented in two categories:

Metric static: groups metrics statically applied to a class or its components. Among which we find the metrics of size, design (Singhs put here the metrics of Halstead), flow control (or else the metrics of complexity), information (contains the metric of Henry and Kafura), weighting (based on the Halstead Singhs defined this metric) and data structure metrics (CK metrics).

Dynamic metric: this type of metrics is applied to objects and not to classes. Singhs have not given a true definition to this type of metric that has not been treated sufficiently.

Despite the structuring of the latter classification, it is rather based on metrics and not concepts and needs to be clarified.

III. Our approach of metrics classification

Viewing the enormous use of object-oriented programming, and in order to organize the number of metrics developed by researchers as well as developers, we propose a classification by concepts of object-oriented programming. We also integrate into this catalog the principles of good practices of programming such as naming conventions, documentation and the number of parameters.

Our metric classification focuses on static product metrics. Nevertheless, we have also presented the other types of measurements in order to have a general classification scheme of measurements. (see figure 2)

Two sets of software measurements are required in a software measurement taxonomy:

The process measures, essentially developed to estimate the budgetary cost of a project, the execution time and the flow of information.

The second family concerns product measures, which in turn encompasses several categories:

Dynamic metrics: this type of metric is applied to objects and not to classes.

Static metrics are metrics applied to a component.

Metrics element includes all metrics that we apply in method, namely : size metrics (number of elements, number of lines of code in the element, number of comment lines, etc.), information metrics (The number of empty lines, as well as the bad smells), convention metrics (checking the method and attribute name, number of parameters, etc.) and complexity metrics.

Class metrics: inheritance metrics (NOC number of descending classes, DIT depth of inheritance), coupling metrics (coupling efferent, coupling afferent ...), cohesion (LCOM), encapsulation (number of public / private data ...), polymorphism (number of polymorphic methods, polymorphism factor ...) and messaging (RFC).

Component metrics contains all metrics that we apply to package, namely dependency metrics (RMI dependencies between packets, dependency cycle), instability metrics (the instability metric is interested to liability / Independence) and those of modularity (the specialization index).

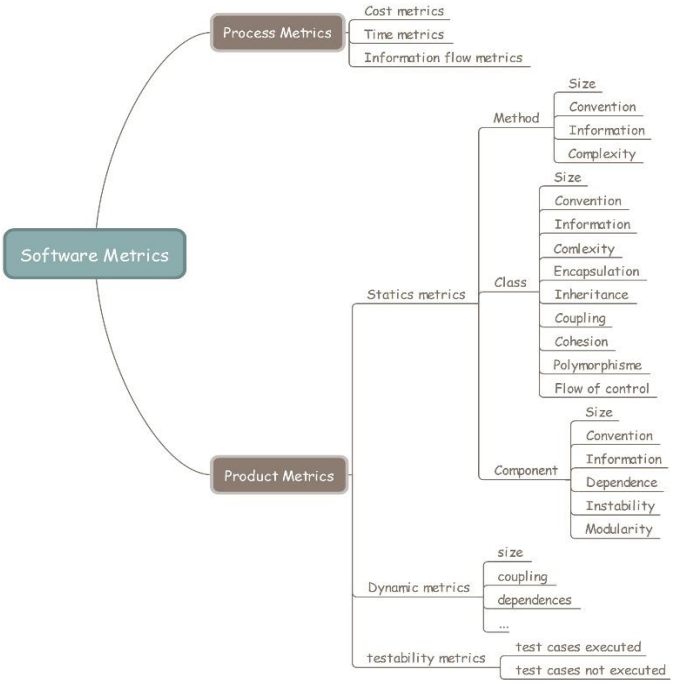

Fig. 2. Our approach to classify metrics using concepts of OO programming.

IV. Application of the classification to the characteristic "maintainability" of ISO 25000

The metrics, posed by the models in their base layer, are not clearly defined and there is not a precise method for choosing the sufficiency number used metric.

The reason why, we have applied the classification by concepts on the ISO 25000 model. Thus, make the model with four layers, an intermediate layer between the metrics and sub-characteristics.

So, the main factor that interests us is that of maintainability. However, the efficiency with respect to time behaviour is strongly related to, the number of objects created and the connection between them, the dynamic metrics.

Also, abstraction and encapsulation play an important role for privacy and data immunity.

Measuring the ease of testing amounts to giving the number of possible test cases in a given code. It is therefore a question of testing all the possible paths in this software, which are established by complexity (McCabe metric in a procedural code, WMC in the OO).

The ease of analysis is explained by a non-complex code. The complexity concerns the number of paths in a program, the size of a program and the possible interactions between elements of program. This translates into terms of concepts by: complexity, size, coupling, dependency, inheritance, information and convention measures.

ISSN 2356-5608

The ease of modification: code is easy to modify if they are not a dependency between the code to be modified and the rest of the application. This translates into the object oriented by strong cohesion and weak coupling. Nevertheless, a strong cohesion complicates the change in a class, and then we use the metrics of complexity and size. As a result, we use the concepts of size, complexity, inheritance, coupling and dependence.

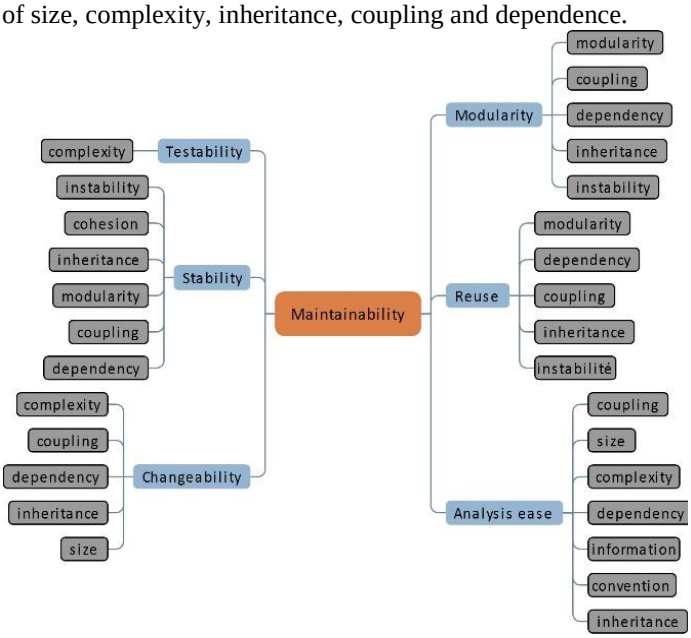

Fig. 3. Applying concepts into maintainability characteristics.

The stability of a software is the results of the stability of its components. Thus, a class must be cohesive and less coupled. A packet must have minimal dependency with other packets.

Modularity is measured by the interdependence of components application. In terms of concepts, it contains: coupling, dependence, instability, cohesion and index of specialization.

The stability of the modification is a characteristic that allows to measure the risks of a modification. So if the change is for a block that has no dependencies, the change will have no risk. So the concepts that concern these parameter are: coupling, dependence, instability, cohesion and specialization index.

V. Survey of Classification of Metrics and the Relationship between Internal Quality Attributes and Quality Measures

To be able to confirm the concepts already mentioned in the previous section, we aim to use a questionnaire. Which is composed of four parts:

- **Profile,**
- **Expertise,**
- **The maintenance and OO concepts,**
- **External attributes and quality concepts.**

The first part of the questionnaire is linked to the profile of the participants (figure 4). The figures show the results of profile responses. The questionnaire participants came from academia and industry with a more or less equitable percentage.

The second part of the questionnaire concerns the participant expertise in object-oriented programming and maintenance (figure 5). For the measurement of expertise, three levels were adopted: low level (0-6 months), average level (> 6 months to 2 years), and high level (> 2 years).

The results of this section show that half of the participants have good experience in maintenance and object-oriented programming.

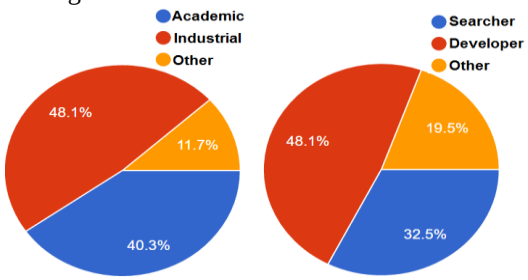

Fig. 4. Profile survey respondents.

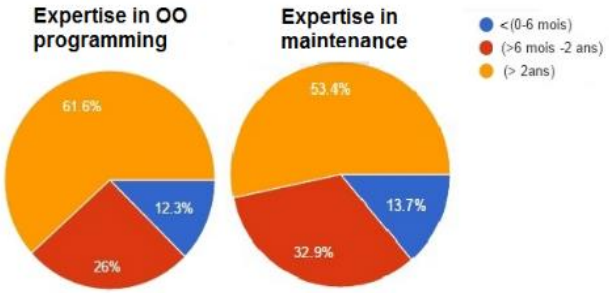

Fig. 5. Expertise of survey respondents.

The third part of the questionnaire concerns the concepts that make problems in maintenance.

Respondents believe that size measures, complexity, conventions and information are very important for the maintenance of a method. The maintenance of a class also includes coupling, inheritance, encapsulation, cohesion, flow control and polymorphism. Component maintenance difficulties depend on size, complexity, conventions and information measures as well as dependence.

The last part of the questionnaire concerns the relationship between external attributes and quality concepts presented by figure 6, 7, 8 and 9. In this section, we have listed the external quality attributes of the ISO 25000 model, and we have left the choice to the respondents to designate the quality concepts that help to highlight the attribute in question. We also gave the respondent a hand to add other concepts to the external attributes.

The results of this part of the survey are categorically close to our approach.

VI. Conclusion

In this paper, we have described our classification of metrics according to the concepts of OO programming and some concepts of conceptions and good practices. Therefore, an application of this taxonomy allows us to know if the number

ISSN 2356-5608

of metrics used is sufficient to conclude that the product is of good quality. And, we applied our metric classification approach to the ISO 9126 and ISO 25000 model in order to link the metrics to the quality factors / criteria.

In addition, to compare our approach with that of developers and researchers, we conducted a survey that included questions to learn about quality concepts that pose maintenance difficulty and questions to link external quality attributes And quality concepts. The result of this survey is categorically close to our study.

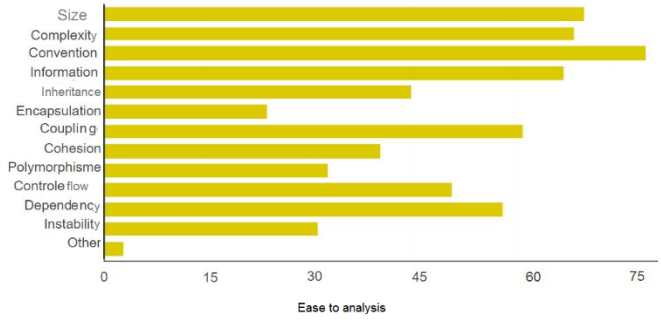

Fig. 6. Result of the concepts posing a difficulty in analysis.

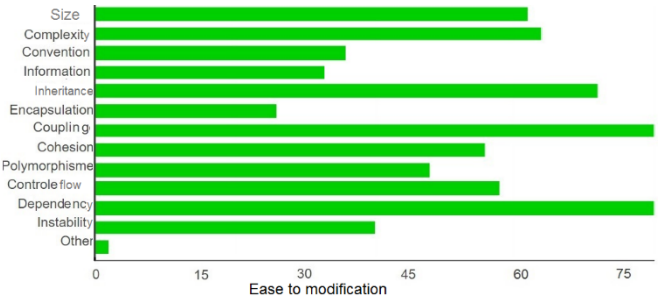

Fig. 7. Result of the concepts posing a difficulty in modification.

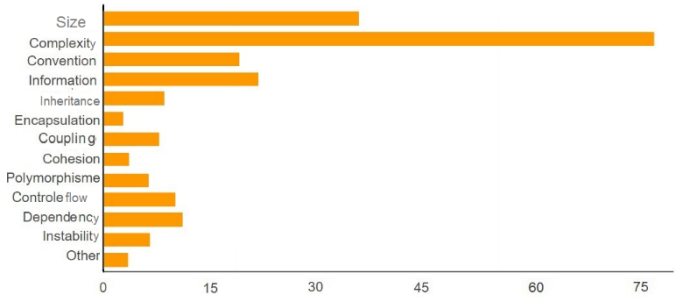

Fig. 8. Result of the concepts posing a difficulty in testability.

REFERENCES

[1] N.I. Churcher and M.J. Shepperd. Comments on : 'a metrics suite for object oriented design'. IEEE Transactions on Software Engineering, March1995.

[2] S.R. Chidamber and C.F. Kemerer. Authors' reply to : 'comments on : A metrics suite for object oriented design'. IEEE, March 1995.

[3] W. Li and S. Henry. Object-oriented metrics that predict maintainability. Journal of systems and software, 1993.

[4] W. Li, S. Henry, D. Kafura, and R. Schulman. Measuring object-oriented design. Journal of Object Oriented Programming, 1995.

[5] F. Brito e Abreu and W. Melo. Evaluating the impact of object-oriented design on software quality. March 1996. IEEE

[6] ISO, International Organization for Standardization (2001), "ISO 9126-1:2001, Software engineering - Product quality, Part 1: Quality model".

[7] ISO, International Organization for Standardization (2005), "ISO 25000, Software engineering - software product quality requirement and evaluation ".

[8] Maurice H. Maurice Howard Halstead. Elements of software science. Operating and programming systems series. Elsevier, New York, 1977. Elsevier computer science library.

[9] Thomas J. McCabe and Charles W. Butler. Design complexity measurement and testing. Commun. ACM, 32(12):1415_1425, December 1989.

[10] T. McCabe, L. Dreyer, A. Dunn, and A.Watson. Testing an object-oriented application. Quality Assurance Inst., 8(4) :21_27, October 1994

[11] LE Hyatt LH Rosenberg. Software quality metrics for object-oriented environments. Crosstalk journal, 1997.

[12] D. E. Monarchi D. P. Tegarden, S. D. Sheetz. Effectiveness of traditional software metrics for object-oriented systems. In Proceedings : 25th Hawaii International Conference on System Sciences, volume 4, pages 359 _ 368, 1992.

[13] Bin-Shiang Liang Yen-Sung Lee and Feng-Jian Wang. Some complexity metrics for object-oriented programs based on information flow : A study of c++programs. Journal of Inforamtion Science and Engineering, 10(1), 1994.

[14] J. Abounader and D. Lamb. A data model for object-oriented design metrics. Technical report, Queen's University, Kingston, ON., 1997.

[15] Steven D. Sheetz, David P. Tegarden, and David E. Monarchi. Measuring object-oriented system complexity. In In Proc. 1st Workshop in IT and Systems, 1991.

[16] Brian Henderson-Sellers. Identifying Internal and External Characteristics of Classes likely to be useful as Structural Complexity Metrics, pages 227_230. Springer London, London, 1995.

[17] Vikram Singh Gurdev, Dilbag. A study of software metrics. IJCEM International Journal of Computational Engineering & Management, 2011.

[18] J McCall. Factors in Software Quality : Preliminary Handbook on Software Quality for an Acquisiton Manager, volume 1-3. General Electric, November 1977.

[19] Barry W. Boehm, John R. Brown, and Hans Kaspar. Characteristics of Software Quality. Vol. 1. TRW series of software technology. North-Holland Publishing, Amsterdam, New York, 1978.

[20] ISO/IEC. ISO/IEC 9126. 9126-1 software engineering -product quality part1 Quality model. ISO/IEC, 2001.

[21] ISO/IEC. ISO/IEC 9126. 9126-2 software engineering -product quality Part 2 _ External metrics. ISO/IEC, 2001.

[22] ISO/IEC. ISO/IEC 9126. 9126-3 software engineering -product quality part 3 _ Internal metrics. ISO/IEC, 2003.

[23] Alain Abran , Rafa E. Al-Qutaish , Jean-Marc Desharnais and Naji Habra, ISO-BASED MODELS TO MEASURE SOFTWARE PRODUCT QUALITY, Jan 2014 · Journal of Web Engineering (JWE)

[24] ISO/IEC 25000, International Organization for Standardization (2008), "ISO 25000, Software engineering - software product quality requirement and evaluation ". web site : miageprojet2.unice.fr/@api/deki/files/2222/=ISO_25010.pdf

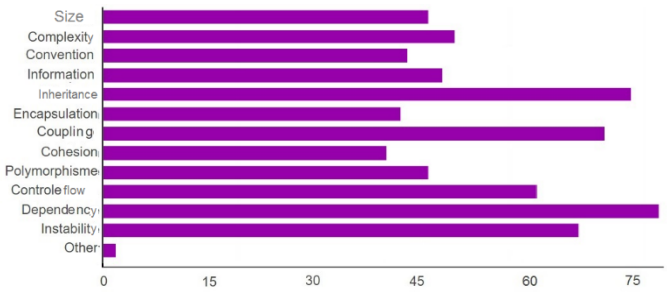

Fig. 9. Result of the concepts posing a difficulty in reuse.

ISSN 2356-5608