

SOFTWARE QUALITY MODEL BASED ON SOFTWARE DEVELOPMENT APPROACHES

Kenyer Domínguez, María Pérez, Anna C. Grimán, Maryoly Ortega, Luis E. Mendoza
Laboratorio de Investigación en Sistemas de Información (LISI),
Departamento de Procesos y Sistemas, Universidad Simón Bolívar,
Apartado 89000, Baruta, Caracas 1080-A, Venezuela.
{kdoming, movalles, agriman, marortega, lmendoza}@usb.ve

ABSTRACT

Product quality is determined by the internal factors of the artifacts generated during the analysis, design and implementation stages. Assessing quality based on existing models is not a trivial process. Several development approaches exist (i.e., structured, object-oriented, component-based, and web-based) which involve different kinds of artifacts susceptible of being measured when estimating the product's internal quality. Such internal attributes affect the product's external quality expressed as external characteristics and sub-characteristics. This work aims to specify internal quality considering the approach used to develop the product. For the preparation of this proposal, we identified several artifacts associated to each approach to define internal quality metrics and their impact on the product's external quality. We analyzed the principles inherent to each approach and the artifacts included in their methods. The QM approach was applied to propose measurements and metrics determining artifacts quality and its relation with ISO/IEC 9126 characteristics and sub-characteristics. The result was a model that gathers the main quality aspects into the six ISO characteristics, and including 298 metrics adapted to these approaches. For reviewing purposes, our new model was applied to four case studies associated to each development approach.

KEY WORDS

Systemic quality model, ISO/IEC 9126, Software metrics, Software development approach

1. Introduction

As people's needs increase, software products must become more efficient, effective, reliable and secure; therefore, their development processes becomes more significant, since they determine the products' quality. Over the years, a variety of quality approaches have arisen to respond to emerging needs and market demands (e.g. efficiency, interoperability, usability, accessibility, reusability, etc.).

Software product quality is determined through the specification of requirements describing the problem, the design modeling the solution, the code leading to an executable program [1]. Hence, to determine the quality of a software product, we must consider a code evaluation

review with respect to the approach applied, thus evaluating devices derived from the analysis and design processes. All this closely relates to the external quality of a product [2, 3].

On the other hand, the concept of Software Product Quality is somewhat abstract and, as such, its evaluation must be operationalized through *quality models* [4]. A quality model is the set of characteristics, and the relationships between them, to provide the basis for specifying quality requirements and evaluating quality [5].

Existing quality models are limited to software quality characteristics for a specific approach, which is mostly structured or object-oriented [6, 7] and, more recently, web-oriented [7] or component-based [8].

This paper presents the improvements made to a quality model that did not identify the approach used for software development upon internal quality review. This new model includes a set of general and particular metrics for the four approaches considered, i.e. structured, object-oriented, component-based, and web-based; in order to define internal quality considering the approach used to develop the product. Both, the original [9] and new model, which introduces several approaches, are based on international standard ISO/IEC 9126 [5] as to the structure of the quality characteristics and sub-characteristics.

Section 2 of this article briefly describes the main development approaches; Section 3 presents the methodology used to specify the new model; Section 4 shows the new software quality model, including each approach's aspects; Section 5 shows the results from the application and review of the new model through four case studies; lastly, Section 6 presents the conclusions and future works.

2. Development approaches

Software Engineering has generated several development approaches aimed at facilitating the analysis, design and implementation of software products, thus satisfying quality characteristics, such as maintainability, efficiency, and reliability, among others. However, this variety of approaches has had an impact on the way software products are developed, since they introduce methods generating particular artifacts [11]. Next we present the

four main approaches, according to developers, and a description of their methods and their most relevant principles for our study.

2.1 Structured development

The structured design of a program is based on the application of the following concepts [4, 12, 13]:

- From general to specific, going deep in the program structure and its level of detail.
- The problem's initial definition is followed by an algorithm scheme described in pseudo codes.
- Language's initial independence.
- Design per levels, leaving details for subsequent levels. Verification of the right scheme at each level.
- Full algorithm's restructuring.

The objective of structured programming is to help designers defining easily readable, verifiable and updateable algorithms [1], based on the structure theorem, which establishes that any program containing only one entry and one exit is equivalent to a program containing only logical structures [13]. These logical structures are sequential, conditional and repetitive [1]. Artifacts generated by this approach include *entity-relation diagrams*, *data dictionaries*, *structured charts* and *data flow diagrams* [13, 14].

This approach opened up new software development methods aimed at improving certain aspects of structured programming. Object-oriented development is a case in point.

2.2 Object-oriented development

Object-oriented development (OO) generates smaller systems than those implemented under structured development through the reuse of common mechanism, thus providing economy of expression and creating systems capable to evolve in time, since their design is based on stable intermediate forms [15].

OO programming is an implementation approach where programs are organized as cooperative object collections, i.e. each object represents an instance of a certain class, which in turn represents the members of a hierarchy of classes joined through inheritance relations [15].

The OO analysis method consists of three individual models: the Functional Model, represented by *use cases diagrams* and *scenarios*; the Analysis Model, represented by *class and object diagrams*; and the Dynamic Model, represented by *states machines and sequence diagrams* [16].

Generally, the principles of this development approach are: abstraction, encapsulation, modularity, inheritance, typification, concurrence and persistency [17, 18, 19], which encourage the reuse of classes. The latter aspect implies a certain level of code complexity.

In the late 90's, a new development approach emerged, which encourages reuse within the software development systems: component-based development [20].

2.3 Component-based development

Components are more abstract than object classes and may be considered as independent service suppliers. If a system requires a certain service, it calls for a component that provides this service, regardless of the component being already in use or the programming language used to develop it [20].

Even though this approach leads to significant reuse, it differs from the method adopted by other disciplines of Engineering, whereby reuse rules the design process. Instead of designing and looking for reusable components, this development approach poses the need to look first into the components before designing the system [14].

The fourth development approach borrows many of the basic concepts from the other approaches, adding some modifications to respond to the web systems' needs.

2.4 Web-based development

Web-based systems and applications have attributes that distinguish them from traditional software systems. Such attributes include network intensity, concurrence, unpredictable load, performance, data-governed availability, content sensitivity, continuous evolution, immediateness, security and aesthetics [20].

To respond to these new attributes, a new Software Engineering branch known as Web Engineering has emerged, encompassing six different types of designs that contribute to global quality [1]:

1. Interface design: it describes the structure and organization of the user's interface.
2. Aesthetic design: also known as graphic design; it describes the web application's "look and feel."
3. Content design: it defines the template, structure, and sketching of the content presented as part as of the web application.
4. Navigation design: it represents the navigation flow between the content objects and all web application functions.
5. Architectural design: it identifies the global hypermedia structure for the web application.
6. Component design: it develops the detailed processing logics required to implement functional components.

Once the main principles of the approaches considered in our study are described, artifacts associated to each approach [1, 12, 14, 15] are listed in the Table 1 below. These will determine the aspects to be reviewed in the quality model proposed, since they are part of the documentation accompanying the code, and will make up a full implementation unit.

All approaches are not independent, since there are different programming languages that allow to develop web systems under an OO, structured or component-based approach. The latter may be considered as a sub-set of the object-oriented approach. Figure 1 shows the relationship among these approaches. A generalization approximation has been used for concepts related to each

approach.

Table 1. Artifacts associated to each development paradigm

Approach	Static	Dynamic	Functional
S	<ul style="list-style-type: none"> Entity-relation diagram Data dictionary 	<ul style="list-style-type: none"> State machine 	<ul style="list-style-type: none"> Data flow diagram Use cases
O	<ul style="list-style-type: none"> Analysis model Design model 	<ul style="list-style-type: none"> State machine Interaction diagram / Event traces Activities diagram CRC model 	<ul style="list-style-type: none"> Use cases
C	<ul style="list-style-type: none"> Component diagram Deployment infrastructure 	<ul style="list-style-type: none"> Component interaction diagram 	<ul style="list-style-type: none"> Use cases
W	<ul style="list-style-type: none"> Class diagram (WAE) 	<ul style="list-style-type: none"> Interaction diagram 	<ul style="list-style-type: none"> Use cases

S: Structured. O: Object-oriented.
C: Component-based. W: Web-based

Although the principles herein described have been widely analyzed, there is no literature for a generic model that allows determining the software system quality, depending on the approach used to develop the product. This is not an easy task, and it should always be based on the principles above mentioned.

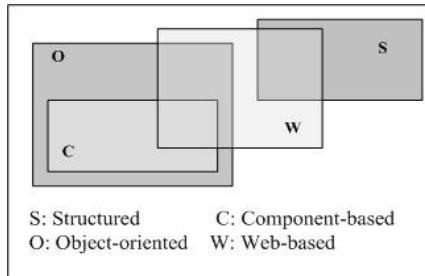


Figure 1. Relationship among the four development approaches selected

3. Methodology used

In order to meet the research's objective, i.e. achieving a software quality model enhancement, and considering all four development approaches herein mentioned, we used the Systemic Methodological Framework [21]. This framework is based on the action-research approach, and it includes the DESMET methodology to evaluate solutions generated. This way we used DESMET to evaluate the new model proposed in this work. It identifies nine (9) different evaluation methods and suggests a set of technical criteria that affect the selection of the evaluation method [22]: the evaluation context, nature of the impact, nature of the object evaluated, the impact's reach, the maturity of the item, the time spent on learning and the maturity of the evaluating organization. It

also suggests three restrictions that can influence the final selection of the evaluation method, such are [23]: the time required for different evaluation options, the trust that the user has in the results of an evaluation and the cost of an evaluation. As a result of applying this method, the *Feature Analysis - Case Study* was used.

In addition, for proposing a new model, we analyzed each approach's principles, and all artifacts included within their methods. Then, the Goal Question Metric (GQM) approach was applied [24] to propose the metrics and measurements associated to each ISO/IEC 9126 characteristics and sub-characteristics [5]. For applying GQM approach in this work, we performed the following steps proposed in [24]: identifying clear goals for the measurement activities; asking questions to determine if the goals have been achieved; and finally, generating the attributes which must be measured to answer these questions.

On the other hand, literature provides several terms for software measurement [25]; therefore, the following definitions have been assumed [5]:

- Metric: the defined measurement method and the measurement scale.

- Measurement: the use of a metric to assign a value (which may be a number or category) from a scale to an attribute of an entity.

After the evaluation and measurement methods were established, we defined the quality characteristics based on ISO 9126. Also, we performed a review of prior works in the field of software metrics. Finally, we selected a set of metrics oriented to the following approaches:

- Structured [3, 4]

- Object-oriented [6, 18, 19]

- Component-based [8, 26, 20, 14]

- Web-based [10, 7, 1]

Upon metrics proposal's preparation, an analysis was performed to determine whether each selected or proposed metric related to one or more approaches, and the artifact to which it would be applied. A sample of this analysis is shown in Table 2.

Table 2. Metrics analysis per approach for Feature, Maintainability, Sub-feature, Analysis capability

Metric	Measurement	Approach	Artifact
Easy to diagnose	Level of difficulty when diagnosing software	All	The approach has no influence
Identification of all relationships	Have all entities of the entity-relation diagram been defined in the data dictionary?	S	Data dictionary
Presentation per level	Has the diagram been divided into separate levels? Is it applicable to DFD?	S	DFD
Numeric identification of databases	Have databases been properly identified with a unique reference number to identify their level? Is it applicable to DFD?	S	DFD
Maximum level number	Does the diagram represent a maximum of nine levels? Is it applicable to DFD?	S	DFD

Concept definition	Is there a definition (glossary of terms) for all concepts involved?	O, W	Class diagrams
User's understanding of all concepts and relations	Does the user understand the concept and relation describing its business?	O, W	Class diagrams
Element location within the diagram	Have elements been properly located within the diagram to facilitate their interpretation?	O, W	Interaction diagram
High-level processes	Has the full system name been assigned to high-level processes? Is it applicable to DFD?	S	DFD
S: Structured. O: Object-oriented. C: Component-based. W: Web-based			

It should be noted that 298 metrics associated to development approaches were identified; therefore, an underlying hierarchy was proposed for the quality model proposed (see Figure 2). Finally, a quality evaluation method was proposed.

4. Software quality model based on development approaches

As indicated in Section 3, we incorporated each analyzed metric into an ISO/IEC 9126 characteristic or sub-characteristic. After this analysis, we presented the final result in the form of hierarchical tree, as can be seen in Figure 2. Notice that Level 1 in Figure 2 represents the ISO 9126 characteristics [5]; Level 2 presents the quality sub-characteristics of each characteristic according to the ISO 9126 model [5] (i.e. suitability-FUN1, accuracy-FUN2, interoperability-FUN3, security-FUN4, maturity-FIA1, fault tolerance-FIA2, recoverability-FIA3, understandability-USA1, learnability-USA2, operability-USA3, attractiveness-USA4, time behaviour-EFI1, resource utilization-EFI2, analyzability-MAB1, changeability-MAB2, stability-MAB3, testability-MAB4, adaptability-POR1, installability-POR2, co-existence-POR3, replaceability-POR4), and also includes some other sub-characteristics related to internal quality aspects [9] such as correctness (FUN5, FIA4, MAB12), structured (FUN6, FIA5, MAB13), encapsulation (FUN7, FIA6, MAB7, POR7), specified (FUN8, USA9, POR9), completeness (USA6), consistency (USA7, POR5), effectiveness (USA8, EFI3), documented (USA10, POR10), self-descriptive (USA11, MAB11, POR11), redundancy (EFI4, POR12), direct (EFI5), used (EFI6), coupling (MAB5), cohesion (MAB6, POR8), maturity (MAB8), control and information structure (MAB9, MAB10), modularity (MAB14), parameterized (POR6), auditory (POR12), and quality management (POR14). Level 3 organizes the metrics presented in Level 4 according to the development approach.

Table 3 shows an example of metrics per each development approach analyzed and associated to an ISO

9126 quality characteristic.

Once all metrics are proposed and incorporated into the model, the Methodological Framework [21] sets out the need to review the proposal's application; therefore, we applied a Feature Analysis - Case Study method.

Table 3. Examples of metrics incorporated into the model

Appr.	Characterist	Sub-charac	Metric	Measurement
S	Functionality	FUN.5. Correctness	Identification	Have entities, data flows, processes and databases been properly identified? Is it applicable to DFD?
O	Efficiency	EFL.6. Used	Polymorphism	Can entities be applied to different objects with the same function?
C	Efficiency	EFL.4. Redundancy	Components' reutilization	Does the system allow reusing the previously prepared code to perform different tasks?
W	Maintainability	MAB.2. Changeability	Continuous evolution	Is the system content regularly updated, and keeping software in constant evolution?
S: Structured. O: Object-oriented. C: Component-based. W: Web-based				

5. Case studies and results

The Feature Analysis - Case Study method establishes a set of features to be evaluated in a model or method; it also requires the selection of a project to be used for the assessment of the model's technical features.

A. Case study:

For the selection of our project, the following criteria were used:

- System should have certain documentation (analysis and design models); it was supposed to be developed in accordance with one of the development approaches analyzed.
- Developers should be available all along the model application and subsequent reviews of the different development methods.

In accordance with the parameters established, 4 projects were selected in order to evaluate the proposed model.

B. Results of the new model application to the selected case studies:

Table 4 shows the results of each project's quality characteristic evaluation. The most relevant characteristics of each project were selected for their review. However, for Project 1 (Structured) only the Functionality review was performed because, as proposed in [27], a lower-than-required functionality (75%) does not require evaluating other characteristics, since it does not comply with the most important one, i.e. Functionality.

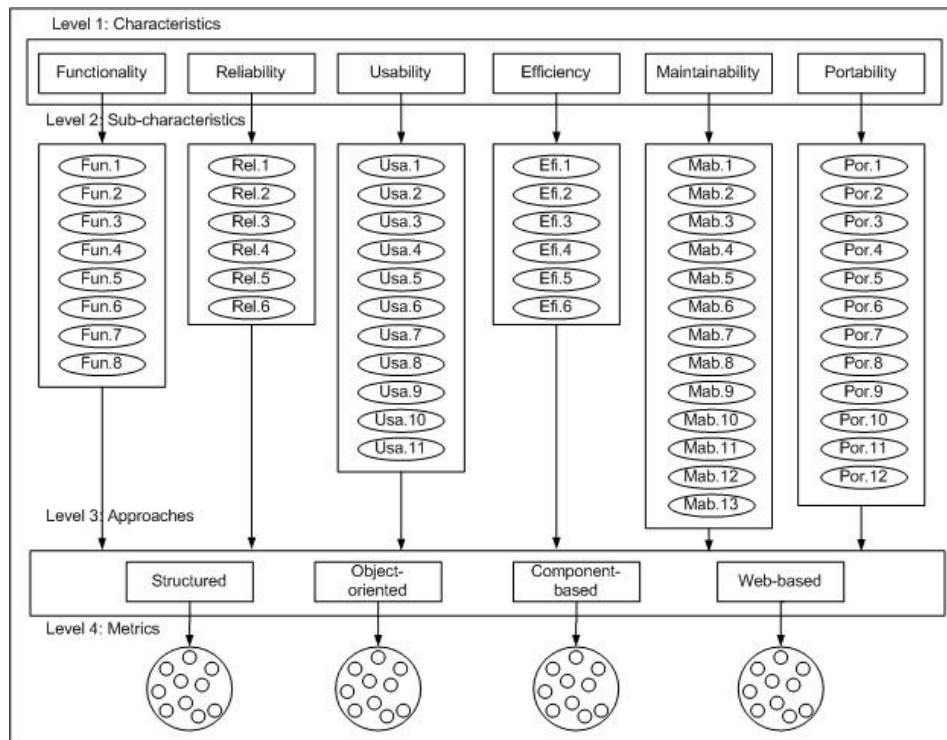


Figure 2. Software product quality model as per development approach

Table 4. Results from the application of the model to 4 Case Studies developed with different approaches

Feature	S	O	C	W
Functionality	12.5	75	100	75
Reliability	NE	83.3	NE	NE
Usability	NE	NE	NE	90.9
Efficiency	NE	NE	83.3	NE
Maintainability	NE	76.9	NE	NE
Portability	NE	NE	66.6	100

S: Structured. O: Object-oriented.
C: Component-based. W: Web-based. NE: Not evaluated

Based on the scores shown in Table 4 and the quality scale proposed in [27], the final results of the evaluation were as follows:

- Project 1 (Structured). Quality level: Null.
- Project 2 (OO). Quality level: Advanced.
- Project 3 (Component-based). Quality level: Intermediate.
- Project 4 (Web-based). Quality level: Advanced.

C. Features evaluation applied to the new model:

In addition to the products' quality assessment, those metrics selected for each project were evaluated according to four criteria: pertinence, feasibility, depth and scalability. Figure 3 shows the average results of this assessment.

D. Results of the Feature Analysis-Case Study applied to the new model:

We established an acceptance level (75%) for each feature evaluated on each metric in the new model. Since the average for each feature surpass the acceptance criteria established, we concluded that the metrics proposal

involved in the model was **adequate**.

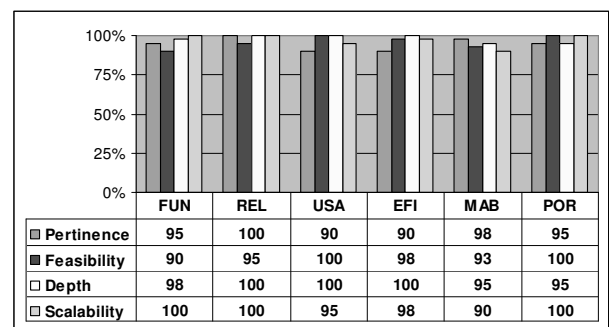


Figure 3. Model features' evaluation

This way we tested the internal validity of the new model in terms of pertinence, feasibility and depth of each metric. The external validity was also demonstrated by determining the scalability of the metrics and the applicability of the model to four case studies. For these reasons we can affirm that the new model was effective in the particular conditions we tested it, as it generated reliable results and illuminated the quality assurance process.

6. Conclusions and future work

The approach used to develop a software system affects the determination of its internal quality, since each approach has particular issues and principles. Evaluating each approach within software product quality determination is a useful tool for software engineers and

architects, who wish to perform product quality reviews. In this work, this was possible by systematically incorporating quality metrics, thus allowing to identify the artifacts' internal quality which subsequently affecting the product's external quality. We also conclude that certain metrics are shared by different methods; otherwise, they are independent from the development method. This reveals that different approaches are not exclusive.

Our model allows quality determination, based not only on the system behavior (external quality), but on its internal quality, represented by its artifacts (e.g. requirement or architecture specifications). The use of this model is aimed at improving current and previously-developed models, which consider the software product external quality, but do not provide details on its internal quality. This way, a systematic evaluation of the product's quality is achieved, identifying potential causes of the product's weaknesses and strengths based on their artifacts' assessment.

For future works we will consider the association of other emerging development approaches (e.g. services-oriented architecture - SOA, the aspect-oriented approach, etc.), which implies changes not only to the product itself, but to the way it is developed.

Acknowledgements

The authors wish to thank K. Barrios and A. Mendoza for its valuable contribution to this work. This research was financed by FONACIT, Venezuela, through Project S1-2005000165 and Universidad Simón Bolívar, through Project DID S1-IN-CAI-012-06.

References

- [1] Pressman, R. S. *Ingeniería de Software. Un enfoque práctico*. 6th Ed. Madrid, España. McGraw-Hill. (2005)
- [2] Bhuvan Unhelkar, *Process Quality Assurance for UML-based Projects*. Addison Wesley. pp 24. (2003)
- [3]. Dromey, G. *Cornering the Chimera*. IEEE Software. 33-43. January (1996).
- [4] Yourdon, E. and Constantine, L. *Structured Design*. New York: YOURDON Press. (1975)
- [5] ISO/IEC 9126-1:2001. *Software engineering product quality part 1: Quality model*. ISO/IEC 9126-1, International Standard Organization, June (2001).
- [6] Bansiya, J. y Davis, C. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*, Vol. 28 (1), pp. 4 – 17, January (2002).
- [7] Malak G., Badri L., Badri M., Sahraoui H. Towards a Multidimensional Model for Web-Based Applications Quality Assessment. *Proc. of the fifth I. C. E-Commerce and Web Technologies (EC-Web'04)*, Spain, LNCS 3182. Springer-Verlag, 316-327 (2004)
- [8] Bertoa, M. F. y Vallecillo. A. Atributos de Calidad para Componentes COTS. *Proceedings de IDEAS 2002*, 352-363. La Habana, Cuba, April (2002)
- [9] Ortega, M., Pérez, M., Rojas, T. Construction of a Systemic Quality Model for evaluating a Software Product. *Software Quality Journal*, 11 (3), 219-242. (2003)
- [10] Calero, C. Ruiz J and Piattini, M. A Web Metrics Survey Using WQM. *International Conference on Web Engineering. Lecture Notes In Computer Science* Vol. 3140. 147-160. (2004)
- [11] Kruchten, P. *The Rational Unified Process. An Introduction Third Edition*. Addison Wesley. 2004
- [12] Gane, C. and Sarson, T. *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall. (1978)
- [13] Joyanes, L. A. *Metodología de la Programación*. 1th Ed. Madrid, España. McGrawHill. (1990)
- [14] Szyperski, C. *Component Software Beyond Object-Oriented Programming*. 2nd Ed. London, Inglaterra. Pearson Education Limited. (2002)
- [15] Booch, G. *Análisis y Diseño Orientado a Objetos*. 2nd Ed. Estados Unidos. Addison-Wesley Iberoamericana S.A. (1996)
- [16] Bruegge, B. & Dutoit, A. H. *Ingeniería de software orientado a objetos*. 1th. Ed. México: Pearson Education. (2002)
- [17] Booch, G. *Object Oriented Development*. *Transactions on Software Engineering*, Vol. 12 (2), 211-221, February (1986)
- [18] Jacobson, I., Christenson, M., Jonsson, P. and Övergaard, G., *Object Oriented Software Engineering*, Addison Wesley, (1992)
- [19] Rumbaugh, H., Blaha, M., Premlarani, W., Eddy, F., and Lorenzen, W. *Object-Oriented Modelling and Design*, Prentice-Hall, 1995.
- [20] Sommerville, I. *Ingeniería de Software*. 7ma Ed. Inglaterra. Pearson Education Limited. (2006)
- [21] Perez, M. Grimán, A. Mendoza, L. E. Rojas, T. A Systemic Methodological Framework for IS Research. *American Conference on Information Systems AMCIS 2004*. New York, USA. (2004)
- [22] B. Kitchenham, *Evaluating software engineering methods and tools, part 1: the evaluation context and evaluation methods*, *ACM SIGSOFT – Software Engineering Notes* 21 (1) (1996) 11–14.
- [23] B. Kitchenham, *Evaluating software engineering methods and tools, part 2: selecting an appropriate evaluation method – technical criteria*, *ACM SIGSOFT – Software Engineering Notes* 21 (2) (1996) 11–15.
- [24] Basili, V., Caldiera, G., Rombach, H. *The Goal Question Metric Approach*. second ed., Wiley Interscience, 528–532, (2001)
- [25] García, F., Bertoa, M., Calero, C., Vallecillo, A., Ruiz, F., Piattini, M. and Genero, M. Towards a consistent terminology for software measurement
- [26] Sedigh-Ali, S. and Paul, R. *Software Engineering Metrics for COTS-Based Systems*. *Computer*. May, 2001, 44-50.
- [27] Mendoza, L., Pérez, M., Grimán A. *Prototipo del Modelo Sistemico de Calidad (MOSCA) del Software*. *Computación y Sistemas*, Vol. 8. México. (2005).