SOFTWARE QUALITY MANAGEMENT FLOSS TOOLS EVALUATION

María Pérez, Edumilis Méndez, Kenyer Domínguez, Luis E. Mendoza and Cynthia De Oliveira
Processes and Systems Department, LISI, Simón Bolívar University, PO Box 89000, Caracas 1080-A, Venezuela

Keywords: FLOSS Tools, Software Quality Management, Evaluation, Quality Model.

Abstract: Software Quality Management Tool (SQM) selection can be a quite challenge, as well as the precise definition of the functionalities that every tool of this kind should offer. On the other hand, establishing standards adequate to the FLOSS context, involves analyzing both commercial and proprietary software development paradigms in contrast to the FLOSS philosophy. This article presents the evaluation of 6 different tools (four proprietary and two FLOSS tools) through the application a Systemic Software Quality Model (MOSCA) instantiation according to the SQM tools context, aiming to illustrate an accurate method for selecting tools of this nature and identifying areas for improvement for each one of the evaluated tools.

1 INTRODUCTION

Software Quality Management Systems (SQM) were created to ensure that the software products and services of an organization satisfy client specifications. The processes underlying SQM are a) Quality Planning; b) Quality Assurance; and c) Quality Control (Pressman, 2007; Sommerville, 2007 and SWEBOK, 2004). A SQM tool should support the three processes in functional terms and meet other non-functional requirements, such as Efficiency, Usability and Reliability, among others. The research's specific objectives are as follows:

1) Generate an instantiation of the Systemic Quality Model (MOSCA) proposed by Mendoza et al (2005) that allows assessing the quality of FLOSS tools supporting SQM. 2) Evaluate a group of software tools supporting SQM of systems and select those with the highest quality level. For the preparation of our work, we followed a Software Quality Generic Model Adaptation Guide (Rincón et. al, 2004). For metric generation purposes, the Goal-Question-Metrics (GQM) paradigm was used, where according to (Basili et. al, 1994), the object is refined into several queries, and each query is subsequently refined into several metrics.

This article consists of six sections. Besides the introduction, conclusions and recommendations, the second section describes the quality model used as a basis for our proposal; the third section introduces the quality model proposal; the fourth section introduces the proposal application and, lastly, the fifth section analyzes the results obtained.

2 SYSTEMIC SOFTWARE QUALITY MODEL (MOSCA)

MOSCA was proposed by (Mendoza et al, 2005) to intendspecify software system quality.MOSCA integrates three quality models: product (Ortega et. al, 2003) and development process (Pérez et. al, 2001) while considering the human perspective (Pérez et. al, 2006). This model relies on total systemic quality concepts (Callaos and Callaos, 1996). MOSCA consists of four levels, as follows:

Level 0. Dimensions. Dimensions include the internal and contextual aspects of process, product and human perspective.

Level 1. Categories. 14 categories have been included herein, as follows: 6 relating to product, 5relating to the development process, and 3 to human perspective.

Level 2. Features. This in-depth level correspond to a group of features defining the key areas to be satisfied to achieve secure and control quality, both for product and process.

Level 3. Metrics. For each feature, a series of metrics to measure systemic quality was proposed.

Mendoza et al. (2005) introduced an algorithm to evaluate software quality using MOSCA, which will be instantiated on the context of SQM for the purpose of this research.

3 QUALITY MODEL PROPOSAL

The formulation of this model required the definition of functionality features of QSM tools, in accordance with the guide's second step (Rincón et. al, 2004). DESMET Feature Analysis method (Kitchenham, 1996) and the support of previous researches and experts in this area. If we follow the MOSCA algorithm (Mendoza et. al, 2005) we must evaluate the product Functionality in the first place. The other categories selected were Usability and Reliability. Selection of these categories and features described below was based on how a SQM tool provides a set of adequate functions in accordance with user specific objectives.

Once the categories and respective features are selected, the metrics for measuring their level of software presence are formulated, thus achieving MOSCA adaptation for SQM tools. The most relevant features were selected for each category and subsequently the metrics that best apply to this product evaluation process. In certain cases, new sub-features and metrics were created. Due to space limitations, we cannot detail all categories and subcategories of the original model (Ortega et. al, 2000).

The model proposed to evaluate FLOSS-based SQM tools contains 43 metric, of which 17 are new and distributed as follows: 21 correspond to Functionality, 16 correspond to Usability and 6 correspond to Reliability. Table 1 shows an example of new metrics included in the model, which related to Functionality. The next section describes the application of the model proposed.

Table 1: Example of new metric.

Features	Sub-feature	Metric´s Description	Metric´s Formulation
Suitability	Software Quality Planning	Is there any tool-related functionality that allows identifying quality objectives?	5= Absolutely. 4= Mostly. 3= Moderately. 2= Scarcely. 1= No.

3.1 Functionality

According to (ISO/IEC 9126, 2001), Functionality is the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. Essentially, an SQM tool must fulfill all functional requirements expected from a software product of this nature. The characteristics of Functionality are suitability, accuracy, interoperability, software product security, correctness, structured, encapsulated and specified. Within this category, the following features were considered: Suitability, Accuracy and Interoperability.

3.2 Usability

According to (ISO/IEC 9126, 2001), Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. The characteristics of Usability are understandability, learnability, graphic interface, operability, compliance, completeness, consistent, effective, specified, documented and self-descriptive. However, only three characteristics were selected, namely Understandability, Learnability and Graphic Interface. Selection of these characteristics is based on the relevance for quality management, for user understandability, usability and their ability to provide guidance on learning and the attributes that make it more appealing to users.

3.3 Reliability

Selection of Reliability was mainly due to the importance of maintaining a level of performance, with the accuracy required, when the software is used under specified conditions (ISO/IEC 9126, 2001).The characteristics of Reliability are maturity, fault tolerance, recoverability, correctness, structured and encapsulated. However, only two characteristics were selected, namely Fault Tolerance and Recoverability.

4 MODEL APPLICATION

The proposed model was applied to 6 tools following MOSCA algorithm (Mendoza et. al, 2005), as explained in the section 3. There is also a web tool that supports the algorithm (http://www.lisi.usb.ve). The tools selected for our study were Rational Quality Manager, Rational AppScan, Rational ClearQuest Test Management, Rational Performance Tester, Sonar and Valgrind, a free alternative to Rational Purify Plus.

Table 2: Model Application Data Summary.

Category	Feature	Sub-Feature	RQM	RAppS	RPT	RCQTM	Sonar	Valgrind	Median
Functionality	Suitability	Quality Planning	85,45%	14,55%	29,63%	41,82%	27,27%	7,27%	28,45%
		Quality Assurance	83,95%	33,33%	28,05%	43,21%	22,22%	9,26%	30,69%
	Accuracy		100,00%	100,00%	100,00%	100,00%	66,67%	33,33%	100,00%
	Interoperability		100,00%	0,00%	0,00%	100,00%	0,00%	0,00%	0,00%
Functionality Porcentage			**100,00%**	**25,00%**	**25,00%**	**50,00%**	**0,00%**	**0,00%**	**25,00%**
Usabilility	Understandability	Quality Assurance	100,00%	0,00%	0,00%	33,33%	100,00%	0,00%	16,67%
	Learnability	Quality Assurance	100,00%	100,00%	100,00%	100,00%	100,00%	0,00%	100,00%
	Graphic Interface	Quality Assurance	100,00%	85,71%	71,43%	100,00%	85,71%	57,14%	85,71%
Usabilility Porcentage			**100,00%**	**66,66%**	**33,33%**	**66,66%**	**100,00%**	**0,00%**	**66,66%**
Reliability	Fault Tolerance	Quality Assurance	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
	Recoverability	Quality Assurance	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
Reliability Porcentage			**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**

5 RESULT ANALYSIS

Given the breadth of the SQM concept, it is hard to find a group of tools that meet the quality metrics of the proposed model, as they must deal, within the Functionality category and specifically in the Suitability feature, with the three main aspects of SQM: Planning, Assurance and Control. The results of the evaluation are presented in Table 2. Rational Quality Manager is the tool that fully meets SQM requirements and shows a leading-edge quality by reaching 100% in the three categories evaluated.

Considering software quality management is a far-reaching discipline, these tools would not cover all sub-disciplines, but may engage in specifically managing some of them. Many of these tools do not widely meet SQM requirements, but give support to some specific SQM areas or processes.

A case in point is Rational ClearQuest Test Management, a tool centered on managing the software application test process, a sub-discipline included in Quality Management Techniques, applicable to other sub-areas of Quality Assurance, Verification and Validation *(*SWEBOK, 2004). In addition, it manages information provided by quality metrics to obtain an overview of the project in terms of the results achieved from the execution of test plans prepared and managed through its support. These strengths help to positioning Rational ClearQuest Test Management above other tools, such as Rational Performance Tester, mainly oriented towards planning and execution of performance tests, including load and stress tests; and Sonar, a tool solely oriented towards management of software quality metrics, a medullar activity for SQM *per se*.

Rational AppScan and Valgrind deal with the specific areas of safety tests and memory performance checks for software application purposes, inserting SQM activities in the group of support tools, but encompassing a more reduced spectrum within the *Quality Assurance, Verification and Validation* sub-areas. One of the most appealing features of Rational AppScan is its ability to adapt to internationally certified standards and generate reports that meet such requirements.

Sonar, which is also a FLOSS tool, is particularly interesting in terms of its evaluation results: the only characteristic where Sonar is outperformed by the Rational tools group is Functionality, although its results should be closely analyzed for the *Quality Planning* sub-feature, where it approaches Rational Performance Tester, and far exceeds Rational AppScan's performance, due to specific management of quality metrics and indicators, statistics, and its ability to track the evolution of such metrics and handle a project portfolio. The metrics management feature is vital for SQM quality and process planning, even though tests, in their wide variety, are very important for this process too. Unsurprisingly, in the evaluation of the *Quality Assurance* sub-feature, Sonar is outperformed by proprietary tools, as the latter provides process management mechanisms of at least one software test type. Interpreting the medians for each feature measurement may give us an idea of which metrics would not be applicable within the context of this sample. In the case of *Interoperability,* measurements, the median is 0.00, which is somewhat of an overall notation given the nature of free-software developed applications. Interoperability usually poses challenges although it should be noted that strengthening of this requisite compliance features may be key for competing in the proprietary software field.

The free software community should bet on the development of highly-interoperable small applications or tool suites, instead of trying to develop increasingly large and complex tool structures in terms of performance that need less interoperability features, such as privative tools where interoperability metrics are rarely applicable.

Metrics of the *Understandability* feature within the Usability category are also one of the least applicable; for privative tools specifically (except for Rational Quality Manager, which shows outstanding results due to its highly intuitive design), this is mainly due to the fact that the tools extensions make them much more complex and understandability thereof may require training and several months of application to achieve full command. In the case of free tools (except for Sonar with excellent results), insufficient documentation and less usable designs are the main reasons affecting their understandability.

From the medians of the metrics applicable to the *Quality Planning and Quality Assurance* sub-features, within the Functionality category, and for *Suitability* - which is an essential feature for evaluation purposes - it might be inferred that most tools belonging to this sample do not meet the quality standards established by the evaluation instruments used for SQM tools.

6 CONCLUSIONS

The model proposed herein provides for appropriate evaluation as it specifies the quality of SQM tools while considering the processes embedded at functional level, such as quality planning, quality assurance and quality control. This contributes to effective software project management taking into consideration the three main processes of SQM.

For future research projects, we recommend defining a process that support organizations in the application of the proposed model, fulfill their requirements and contributes to tool identification, classification, and sourcing.

ACKNOWLEDGEMENTS

This research has been financed by FONACIT Venezuela, Project G-2005000165. Special thanks to Eng. A. Castillo.

REFERENCES

Basili, V. R., Caldiera, G., Rombach, H. D. 1994. Goal Question Metric Paradigm. In J. J. Marciniak (ed.), Encyclopedia of Software Engineering, John Wiley & Sons.

Callaos, N. and Callaos, B. 1996. Designing with Systemic Total Quality, International Conference on Information Systems, Orlando, Florida, July, 548-560.

ISO/IEC 9126-1. 2001. Software engineering - Product quality - Part 1: Quality model. First edition,

Kitchenham, B. 1996. Evaluating Software Engineering Methods and Tools. Part 1: The Evaluation Context and Evaluation Methods. ACM SIGSOFT - Software Engineering Notes, 21, 1, 11- 14.

Mendoza, L; Pérez, M. and Grimán, A. 2005. Prototipo de Modelo Sistémico de Calidad (MOSCA) del Software: Computación y Sistemas, 8, 3, 196-217.

Ortega, M., Pérez, M. and Rojas, T. 2000. A Model for Software Product Quality with a Systemic Focus, 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000 and The 6th International Conference on Information Systems, Analysis and Synthesis ISAS 2000, Orlando, Florida, July, 395-401.

Ortega, M., Pérez, M. and Rojas, T. 2003. Construction of a Systemic Quality Model for evaluating a Software Product, Software Quality Journal, Kluwer Academic Publishers, Julio, 11:3, 219-242.

Pérez, M., Rojas T., Mendoza, L. and Grimán, A. 2001. Systemic Quality Model for System Development Process: Case Study, Seventh Americas Conference on Information Systems - AMCIS, Boston, Massachusetts, August, 1297-1304.

Pérez, M., Domínguez, K., Mendoza, L. and Grimán, A. 2006. Human Perspective in System Development Quality. 12th Americas Conference on Information Systems (AMCIS). Acapulco, México. Agosto.

Pressman, R. 2007. Software Engineering: A Practitioner's Approach. 7th Edition. Mc Graw Hill.

Rincón, G., Mendoza, L. & Pérez, M. 2004. Guía para la Adaptación de un Modelo Genérico de Calidad de Software. IV Jornadas Iberoamericanas en Ingeniería de Software e Ingeniería del Conocimiento - JIISIC, Madrid, España.

Sommerville I. 2006. Software Engineering. Addison Wesley; 8th edition.

SWEBOK. 2004. *SWEBOK: Guide to the Software Engineering Body of Knowledge - 2004 Version*. IEEE Computer Society.