

Quality Measurement Model for Analysis and Design Tools based on FLOSS

O. Alfonzo; K. Domínguez; L. Rivas, M. Pérez; L. Mendoza; M. Ortega
*Laboratorio de Investigación en Sistemas de Información (LISI),
Departamento de Procesos y Sistemas, Universidad Simón Bolívar,
Apartado 89000, Baruta, Caracas 1080-A, Venezuela*
*oras14@gmail.com, kdoming@usb.ve, lornelr@gmail.com, {movalles, lmendoza,
marortega}@usb.ve*

Abstract

The selection of Free/Libre Open Source Software (FLOSS) tools for Analysis and Design (A&D) is a hard task due to their complexity, their wide variety within the market, and their functionality level. This article aims at introducing a group of quality features to evaluate FLOSS tools for A&D. Such features are presented based on the perspective of the Software Quality Systemic Model (MOSCA), inspired by ISO/IEC 9126 [9], Dromey's quality model [5] and the Goal-Question-Metrics (GQM) Paradigm. This model includes attributes such as functionality, usability and maintainability, and establishes 52 new metrics for a total of 102 metrics that allow evaluating FLOSS tools for A&D within a specific context. Lastly, validation of this proposal is verified through its application on eight FLOSS-based A&D tools.

1. Introduction

The Free Software Foundation (FSF) [7] states that Free Software (FS) is a matter of liberty rather than price. FS refers to users' freedom to run, copy, distribute, study, change and improve software. The FSF establishes four kinds of freedoms for users to identify free software: (a) freedom to run the program, for any purpose; (b) freedom to study how the program works, and adapt it to their needs; (c) freedom to redistribute copies; and (d) freedom to improve the program, and release such improvements to the public. Another related movement focuses on providing licenses, which among others benefits, grant access to the source code: Open Source Initiative [16]. As a result thereof, the term FLOSS (Free/Libre Open Source Software) was coined to join both movements' efforts. But, are these 4 freedoms or access to the source code enough to support the high quality of a system? According to [21], software quality is the concordance of functional and performance

requirements explicitly established with properly documented development standards and implicit features expected for any professionally-developed software. However, this applies to traditional software engineering where well-defined roles and disciplines exist, but regarding FLOSS there is no consensus on how to achieve systemic quality [3]. Consequently, relevance has been given to the study of FLOSS' satisfaction of functional/non-functional requirements for systems developed under its principles. This article aims at introducing a group of quality features to evaluate FLOSS tools for A&D, but how can we identify an A&D tool?

On one hand, the analysis of a system comprises several steps, from the definition of the problem dominium until the formulation of a system proposal. The analysis may be understood as the process whereby we must identify the problem source, as well as all elements that will be subsequently incorporated into the system to find a solution (software) and their interrelations, and the general and specific objectives of the system under construction, from the abstract to the concrete. Here is where design begins. Components (elements' evolution), relations and identified objectives are modeled through diagrams to obtain a plot that illustrates how to go from the problem to its solution. A&D are directly related along the system formulation or modification process, even though they may be studied separately.

An A&D system tool should allow reproducing in detail the abstractions comprised in the design diagrams; it should handle diagrams in environments where analysts and designers may manage them, either individually or collectively, and group them according to the design views. Likewise, it should offer functionalities that allow linking diagram abstractions while facilitating the verification of traceability, from user's need to software coding. A model aimed at evaluating an A&D tool functionality must consider functionalities such as code generation, diagram

grouping into models such as 4 + 1 views (or any other kind of classification), transformation of one diagram into another that allows modeling other views, abstractions and different types of pre-determined diagrams, etc.

According to [10] there are four reasons to decide to work with A&D tools, namely increase in the analyst's productivity, improvement of analyst-user communication, integration of life cycle activities (insofar as continuity from one phase to another is provided) and evaluation of the impact of changes on maintenance. A&D tools should allow building abstractions from both, data and processes, by translating functional/non-functional requirements into specifications of architecture, data, processes and interfaces to be implemented. However, process models, approaches, and methodologies consider that the analysis and design by graphic models or representations is part of a development strategy aimed at guaranteeing products that remain consistent to their initial requirements, assertive communication with users, and processes and products that comply with the quality standards established.

Considering that the A&D of systems imply making key software development decisions, this article aims at introducing a software quality systemic model to evaluate FLOSS-based tools supporting A&D activities, which may serve as a reference for organizations when supporting the selection of those tools that best suit their needs.

Before proposing the model, it was necessary to generate a conceptual model that gathered all existing concepts comprised in the A&D literature, and their relation with the process models, development approaches and methodologies.

The research's specific objectives are as follows:

1. Generate an instantiation of the Quality Systemic Model (MOSCA) that allows assessing the quality of FS tools supporting A&D of systems.
2. Evaluate a group of software tools supporting A&D of systems and select those with the highest quality level.

In addition to identifying the highest quality tool in accordance with the model, the evaluation is performed in order to make subsequent functional contributions to the tool selected, submit the tool improvements to the specialized users' review, and submit the tool modifications to the development community's approval for its release.

This article consists of eight sections. Besides the introduction, conclusions and recommendations, the second section describes related works within this field; the third section describes the quality model used as a

basis for our proposal; the fourth section describes the methodology used; the fifth section introduces the quality model proposal; the sixth section introduces the proposal application and, lastly, the seventh section analyzes the results obtained.

2. Related works

FLOSS offers low-cost systems based on open standards, which favor collaborative work, increase technology capacity, reduce dependency on vendors, and promote the development of local corporations [27]. FLOSS also offers web solutions that have proved to be much more robust than their proprietary software analogues [14, 28].

In 2006, the Software Quality Observatory for Open Source Software [26] was created for the purpose of developing a group of software evaluation tools to analyze and compare source code quality and test their adequacy for corporate applications. Several projects such as FLOSSMetrics [6] have arisen ever since, for the main purpose of building, publishing, and analyzing large databases including information and metrics related to FLOSS systems, using current methodologies and tools already developed

This huge variety of tools supporting A&D is mainly due to the diversity of notations and software design methods. Such tools generally support requisites modeling and traceability, as well as software design creation and testing.

Some researches have been conducted to evaluate CASE tools supporting MDA and UML [8, 24] on a comparative basis. However, their related literature does not reflect any initiative to develop a quality model for FLOSS tools supporting A&D activities. Therefore, this research focuses on the Software Quality Systemic Model (MOSCA, or Modelo Sistemico de Calidad) [13] for the purpose of adapting it to the features of this type of tools. Following is the description of the quality model.

3. Software Quality Systemic Model (MOSCA)

The Software Quality Systemic Model proposed by [13] is intended to specify software system quality. MOSCA integrates three quality models: product [18, 19] and development process [23] while considering the human perspective [22]. This model relies on total systemic quality concepts [3]. 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, 5 relating 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. [12] introduced an algorithm to evaluate software quality using MOSCA, which is summarized as follows:

1) Measurement of product quality functionality. Product functionality must be measured first. If 75% of the required features established for Functionality are satisfied, then the next step is addressed. According to [18], this 75 % value was determined by the interested party in the evaluation so it can be modified depending on the requirements demanded of the product.

2) Instantiation of the product sub-model. In this activity, the customer must select two categories from the five categories of the product sub-model, including those considered as software musts (functionality) and those it wishes to evaluate. Then, each of the categories selected by the customer must be evaluated. It should be noted that the algorithm recommends working with a maximum of three product features (including functionality). If more than three product features are selected, these may cause conflicts with each other.

3) Quality measurement for each category. For the two categories previously selected, the following should be made:

- Apply metrics proposed in the product sub-model for selected categories.

- Verify that 75% of the metrics reach optimum values (3 or higher) for each feature.

- Evaluate the category. For a category to be fulfilled, at least 75% of its features must be highly satisfied, thus guaranteeing coherence and consistency regarding the model acceptability levels.

4) Measure the product quality from those categories evaluated. It should be remembered that when functionality is not satisfied, the algorithm ceases to work, and the software product quality is deemed *null*. If a software product meets its original purpose (functionality), it is deemed as having a *basic* quality. If it satisfies only one of the categories selected, besides functionality, it is deemed as reaching an *intermediate* quality level, but if it satisfies all selected categories, it is deemed as reaching an *advanced* quality level.

4. Methodology

For the preparation of our work, we followed a Software Quality Generic Model Adaptation Guide [25], within a systemic quality approach. This guide focuses on the identification of evaluation needs, the knowledge area to be addressed, and users' specific requirements [25]. For metric generation purposes, the Goal-Question-Metrics (GQM) paradigm was used, where according to [1], the object is refined into several queries, and each query is subsequently refined into several metrics. Some metrics may be used to answer several queries for the same objective; several GQM models may even have queries and metrics in common, given that metrics may have different values depending on the perspective adopted.

The Research Lab LISI (Laboratorio de Investigación en Sistemas de Información) in USB (Universidad Simón Bolívar) stated the evaluation needs for A&D tools so to provide results on the assessment to small- and medium-sized FS developing companies in different sectors, mainly the public sector. In addition, the results of this evaluation are used in the laboratory to enhance a FS tool.

5. Quality Model Proposal for FLOSS-based A&D tools

The formulation of this model required the definition of functionality features of A&D tools, in accordance with the guide's second step [25], DESMET Feature Analysis method [11], and the support of previous researches and area experts. If we follow the MOSCA algorithm [12], we must evaluate the product Functionality in the first place.

5.1. Functionality

According to [9], 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 A&D tool must fulfill all functional requirements expected from a software product of this nature. Within this category, the following features were considered:

- *Suitability (FUN1)*. The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives [9]. At this point, three sub-features were added, related to *diagrams*, *documentation* and *linking between A&D and implementation*.

- *Accuracy (FUN 2)*. The capability of the software

product to provide the right or agreed results or effects with the needed degree of precision [9]. It was required that the product allowed creating a system design that translates into a correct implementation with respect to original requirements, and this will depend on the functionalities' accuracy and precision level.

- *Interoperability (FUN 3)*. The ability of a software product to interact with one or more specified systems [9]. Given that there are tools that support stages prior or subsequent to the A&D process, it will be relevant to know whether the evaluated product interacts with any tool including these or other features.

- *Correctness (FUN 5)*. This is divided into three categories, related to computing, completeness, and consistency capacity. The violation of any of such properties may show that the software does not have the functionality level expected [17]. Features such as completeness of models and source codes generated, and consistency in the different abstractions created by the tool, are essential for system design.

- *Encapsulated (FUN 7)*. Variables, constants and types must be used within the context they were originally defined. The manner in which they are used may have a significant impact on Modularity and, therefore, on the models and programs quality [17]. The functional ability to maintain variables linked to abstractions used for A&D within the corresponding context is deemed relevant inasmuch it allows orienting development by design principles such as modularity.

The new functionality metrics allows knowing if a tool supports A&D systems in accordance with the different development approaches. Therefore, we measure whether the application allows preparing diagrams of the structured OO approach, the aspect-oriented approach, or the component-oriented approach. In addition, in these new metrics added to the sub-feature Diagrams, we evaluate whether the tools allow creating the different design views proposed by the different authors. On the other hand, when formulating questions on documentation potentialities, we evaluate whether the tool generates the artifacts required when and agile or plan-oriented methodology is applied. Metrics added to the sub-feature Coupling between A&D and Implementation allow determining the capability of the tool to offer a traceable vision all along the development process, so to adapt to the methodologies that require it, and its capability to link different phases within the development process, so the analysis and design are adapted to the process models studied. Furthermore, it provides a potential classification of the application assessed, as to one type or another. Regarding metrics of the Correctness feature, these allow measuring

quality of the A&D of the system under construction, which will be provided by the tool. Lastly, metrics included in the Encapsulated feature are oriented to evaluate the extent to which the tool will be useful for the user in order to perform a granular, detailed and coherent analysis.

The algorithm's second step establishes the selection of two additional categories from Reliability, Usability, Efficiency, Portability and Maintainability. Following are the reasons for the selection of Usability and Maintainability.

5.2. Usability

According to [9], Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. Products subject to evaluation must allow designing different system models. They require making several utilities available to users in order to integrate the necessary elements, so design and analysis are accurate and consistent; therefore, it is important that the product complies with minimum Usability standards.

In addition, the main disadvantages faced within the FLOSS world relate to Usability, since developers who are not traditional users tend to develop systems [15] that require specific IT knowledge for installing free software, thus making maintenance harder for customers.

The following features were considered for this category:

- *Understandability (USA 1)*. The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use [9]. These metrics should be able to evaluate the behavior of those users without prior knowledge of software operation and measure difficulty levels when understanding software functions, operations and concepts [17]. It is essential that the A&D supporting tool can be used for development purposes by adopting different approaches and implementing different methodologies, whether agile or plan-oriented. In this respect, the product must provide all necessary facilities, so analysts and designers can easily understand how to use it in accordance with the development context.

- *Graphic interface (USA 3)*. The ability of a software product to be appealing to users [9]. While a wide range of functionalities of this kind of tools are design-oriented, the graphic interface must be appealing and friendly to users in order to perform A&D related activities within a comfortable and

suitable environment and speed up man-system interactions.

- *Operability (USA 4)*. The capability of the software product to enable the user to operate and control it [9]. The relevance of considering operability as a feature to be evaluated in an A&D tool lies in the users' need to count on all possible assistance for executing software, so they can control all necessary variables to achieve the desired A&D.

- *Effectiveness (USA 8)*. A structural form (the statement types and the statement components of the implementation language [5]) is deemed effective when it counts on all the necessary elements to be defined and implemented [17]. Creation of a mechanism to make an effective use of the product ensures a better man-system interaction in terms of time and results.

- *Self-description (USA 11)*. A structural form is deemed self descriptive when its purpose is evidenced in the name of the modules and when labels have meanings that refer to the application context [17]. If the tool is self-descriptive, an adequate system-user interaction will be easily achieved.

The new metrics added to the Usability category allow evaluating the dependency of the tool to the FLOSS development philosophy. The greater the users' need for FLOSS knowledge to install or use the system, the higher the difficulty to understand and use it under different conditions.

Lastly, the third category selected was Maintainability.

5.3. Maintainability

According to [9], Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. A FLOSS tool must be built so its modification and maintenance are the easiest possible and it must count on sufficient information to give the development community easy access to necessary information for software maintenance and enhancement. Within this category, the following features were considered:

- *Analyzability (MAB 1)*. The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for identification of the parts to be modified. [9]. It is required that tools are of easy diagnosis, since evaluation and selection of parts to be improved will be dependant thereon.

- *Changeability (MAB 2)*. Capability of the software product to enable a specified modification to be

implemented [9]. This is the main feature of a FLOSS tool. The greater the changeability of a tool, the greater its potential to be improved with contributions from the software development community. This category was added a new feature related to the type of license to release the A&D tool.

- *Stability (MAB 3)*. The capability of the software product to avoid unexpected effects from modifications of the software [9]. Quality compliance of specific functional and non-functional requirements of an unstable tool may be lost upon one single modification, which although aimed at improving software, may also generate failures in sectors known to have operated in a proper manner in the past.

- *Coupling (MAB 5)*. It helps measuring the interconnection of software structure modules. Coupling depends on the complexity of the interface between modules, the module entry or reference point, and data entering through the interface. In terms of software design, the lowest possible coupling level is pursued [17]. Coupling is a desirable feature in any tool to be modified, since the simpler the connection between the software-integrating modules, the easier functioning and identification of sectors to be modified to achieve desired changes.

- *Cohesive (MAB 6)*. A structural form is deemed cohesive when its elements are closely linked to each other and contribute to meet a simple objective or function. The main objective is to achieve a high cohesion level and recognize when there is little cohesion, so the software design can be modified to achieve higher functional reliability [17]. During a software modification, updating or maintenance process, cohesive modules shall allow concentrating the development community efforts on key functionalities.

- *Software maturity attributes (MAB 8)*. These are the physical and measurement features associated to age and use of the target software system [17]. This project focuses on the selection of a tool with sufficient maturity as to ensure future modifications with high probabilities of success. This feature was added new metrics related to the size of the community supporting the tool to be evaluated.

The new metrics added to the Maintainability category allow defining the tool's contextual aspects that may or may not facilitate its subsequent modification as part of the common effort of FLOSS developments. Information derived from the application of such metrics on a tool will help determining the amount and quality of human resources (development community members) and legal and documentation resources that will be made available at

the time modifications or functionalities are incorporated in the tool. Specifically, measuring the number of software downloads per year give an idea of the software popularity and acceptance among users.

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 sub-categories of the original model [17]; however, Figure 1 shows the sub-tree of MOSCA's product dimension including the modifications made according to the specific evaluation context.

The model proposed to evaluate FLOSS-based A&D tools contains 102 metric, of which 52 are new and distributed as follows: 41 correspond to Functionality, 9 correspond to Maintainability and 2 correspond to Usability. Appendix A and B shows an example of new metrics included in the model, which related to Functionality and Maintainability. The next section describes the application of the model proposed.

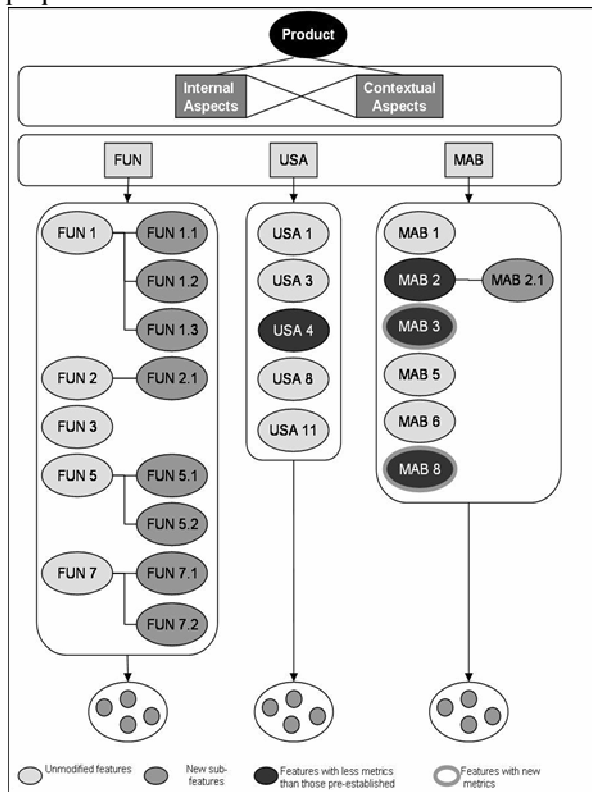


Figure 1: Modifications made to the product dimension for instantiation of MOSCA in analysis and design tools.

6. Model Application

To be considered as an A&D tool, a tool should not only allow plotting, but it should provide functionalities that help integrating the analysis process to the diagrams; therefore, tools such as *Microsoft Paint*, *Power Point* and even UML plotters, which generate graphics from plain plots as *Graphviz* and *UMLGraph*, do not offer the necessary functionalities to link such diagrams and manage their relations, thus restricting analysis activities.

The tools subjected to study are StarUML [31], ArgoUML [29], BOUML [2], Fujaba [33], UMLet [32], Papyrus [21], DIA [30] and DBDesigner [4]. All tools allow drawing UML diagrams with the exception of DBDesigner. This tool is oriented to A&D for Data Base. There is a wide variety of UML modeling tools within the market, and those selected constitute a representative sample thereof inasmuch they have resulted from particular initiatives, academic projects or as analog alternatives to commercial tools. In the case of DIA, this tool was selected for its versatility and popularity within the academic sector. DBDesigner was selected to encompass tools that were not exclusively UML modeling tools and that allow database A&D, and views of the software modeling, which is optional in UMLs.

The proposed model was applied to 8 tools following MOSCA algorithm [12], as explained in the section 3. There is also a web tool that supports the algorithm (<http://www.lisi.usb.ve>).

In order to respond to the Functionality and Usability metrics, we downloaded the tool's most recent versions available at the official websites of the respective development community. Applications were run through step-by-step verification of compliance/non-compliance with functionality and usability.

To respond to the Maintainability metrics, we visited the official websites of the development communities and searched for the mechanism that provide access to the code, its licenses, and bugs databases, as well as new requirements, faults, and registered users, and downloads statistics, among others,

The stage corresponding to the search, download and installation of recent versions took approximately 1 hour per tool, totaling 8 hours. The stage corresponding to "familiarization" with the applications (i.e. to know the modalities of use, extension of files generated, location of functionalities, profile management, among others) took approximately 3 hours per tool, totaling 24 hours. The application of the Functionality metrics took approximately 2 ½ hour per tool, totaling 20 hours. The application of Usability metrics took approximately 1

hour per tool, totaling 8 hours. The application of Maintainability metrics took approximately 3 hours per tool, totaling 24 hours.

The evaluation process and the process of comparative analysis to identify those tools to be improved, and which improvements should be made. Results (Table 1) consisted in summarizing the number of metrics which values were greater than three, and calculating their percentages per category. The MOSCA [12] algorithm was applied to define each tool's quality level in accordance with the level of satisfaction achieved (percentage of metrics greater than three, Table 1), for comparison of the eight results obtained and the selection of the tool with the highest percentages for each category. It is quite difficult to determine the duration of the process of result's comparison, tool's analysis and selection, and identification of features to be improved, since the evaluation underwent several revisions. Nevertheless, we may estimate approximately 15 hours for this task, excluding the results' discussion and revision sessions.

Given these estimated times, we may state that 32 hours were invested in the assessment's preliminary phase (search, download, installation and familiarization). Approximately, 52 hours were invested in MOSCA application; and, in sum, stages corresponding to preparation, evaluation and results' analysis took 99 hours altogether.

The evaluation was performed by a pre-graduate Computer Engineering student without prior experience on software quality models' application, but with expertise in the use of A&D tools within application development processes, for academic purposes. The evaluator also performed the MOSCA instantiation process for its application on tools supporting analysis and design activities.

Table 1 shows the results obtained, thus detailing for each tool the values of each sub-feature. These results correspond to the application of the MOSCA algorithm [12]. Cells containing values have been filled-in with two kinds of colors, as follows:

- Cells with a light fill-in color and black font reflect the lowest value obtained by this category or tool, e.g. Fujaba is the tool with the lowest value for the Diagrams sub-feature, since it provides the lowest number of diagrams.

- Cells with a dark fill-in color and white font reflect the highest value obtained for this category or tool, e.g. ArgoUML is the only tool with a 100% of "Software Maturity attributes" and the tool with the highest percentage as to Maintainability.

It can be observed that the last column shows the median value for each sub-feature in order to analyze the behavior of the new metrics.

7. Result analysis

From these results we can obtain conclusions related with the behavior of the new metrics and, conclusions related with the quality attributes of the tools.

In Figure 2, we can observe that not all Functionality metrics have high percentages, indeed "Taxonomy" and "Abstractions Context" tend to zero. This behavior warns us to review those metrics, due to the fact that most tools show low values. Stability and Interoperability show low values too, but these metrics have been validated in previous evaluations with acceptable results [18]; therefore, we can conclude that the tools show a low level of stability and interoperability.

Figure 2 shows the evaluation results of the A&D tools selected considering their Functionality. StarUML is the only tool exceeding the 75% required by the MOSCA algorithm. Only three of the tools (DBDesigner, BOUML and Papyrus) reached a percentage higher than 50%. Tools with the lowest functionality level (ArgoUML and DIA, with 33.33% each) are the tools with the lowest number of released versions (version 0.24 and version 0.96.1, respectively).

Regarding Usability, Figure 3 shows results where most FLOSS-based A&D tools satisfy this feature with a percentage over 75%. Both, StarUML and ArgoUML, share 100%, while the rest shows an 80% satisfaction level. The only tool that did not satisfy the Usability level was Fujaba, probably due to its low-quality graphic interface and little understandability.

Lastly, Figure 4 shows the results for the selected tools regarding the Maintainability category. Even though all A&D tools evaluated are based on FLOSS principles, they show a low Maintainability level, which is common for this area, since in most cases access is granted to the source code, without complete documentation of the product. Upon adoption of the MOSCA algorithm, and having obtained a satisfaction percentage over 75%, StarUML is the only tool that reached an Intermediate quality level for 2 of the three selected categories, including Functionality.

Table 1: Model Application Data Summary

Category	Feature	Sub-feature	StarUML	ArgoUML	BOUML	Fujaba	Dia	Papyrus	UMLet	DBDesigner	MEDIAN
Functionality	Suitability	Diagrams	81,25%	62,50%	68,75%	31,25%	42,86%	50,00%	68,75%	100,00%	65,63%
		Documentation	100,00%	100,00%	100,00%	50,00%	50,00%	50,00%	0,00%	100,00%	75,00%
		Link A&D and Implementation	100,00%	50,00%	100,00%	100,00%	0,00%	50,00%	0,00%	100,00%	75,00%
	Accuracy	Abstraction details	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
	Interoperability	Interoperability	25,00%	25,00%	25,00%	25,00%	25,00%	83,33%	83,33%	25,00%	25,00%
	Correctness	Complete	100,00%	0,00%	100,00%	0,00%	100,00%	100,00%	100,00%	0,00%	100,00%
		Consistent	75,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
	Encapsulation	Taxonomy	100,00%	0,00%	100,00%	0,00%	100,00%	0,00%	100,00%	0,00%	0,00%
Abstraction Context		100,00%	0,00%	0,00%	100,00%	0,00%	100,00%	0,00%	0,00%	0,00%	
Functionality Percentage			88,89%	33,33%	66,67%	44,44%	33,33%	66,67%	44,44%	71,43%	55,56%
Usability	Understandability	Understandability	80,00%	100,00%	80,00%	60,00%	100,00%	80,00%	80,00%	100,00%	80,00%
	Graphic Interface	Graphic Interface	100,00%	100,00%	75,00%	62,50%	75,00%	100,00%	75,00%	87,50%	81,25%
	Operability	Operability	84,62%	76,92%	53,85%	46,15%	53,85%	69,23%	46,15%	61,54%	57,69%
	Effectiveness	Effectiveness	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
	Self-description	Self-description	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
Usability Percentage			100,00%	100,00%	80,00%	40,00%	80,00%	80,00%	80,00%	80,00%	80,00%
Maintainability	Analizability	Analizability	50,00%	100,00%	50,00%	50,00%	50,00%	50,00%	50,00%	50,00%	50,00%
	Changeability	Changeability	60,00%	40,00%	60,00%	40,00%	80,00%	60,00%	60,00%	60,00%	60,00%
		License	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
	Stability	Stability	0,00%	50,00%	0,00%	50,00%	50,00%	0,00%	50,00%	25,00%	
	Coupling	Coupling	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	0,00%	100,00%	100,00%
	Cohesive	Cohesive	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	0,00%	0,00%	100,00%
	Software Maturity attributes	Software Maturity attributes	88,89%	100,00%	44,44%	66,67%	44,44%	33,33%	55,56%	77,78%	61,11%
Maintainability Percentage			57,14%	71,43%	42,86%	42,86%	57,14%	42,86%	14,29%	42,86%	42,86%

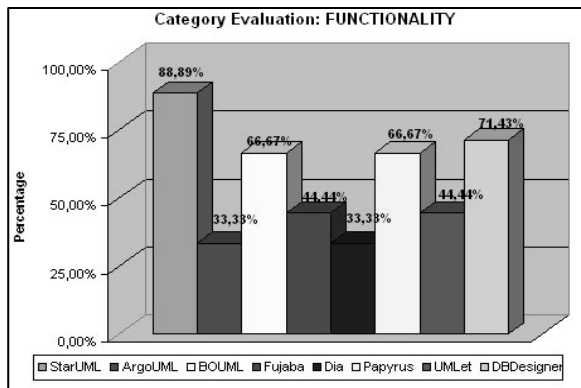


Figure 2: Comparison of FS-based A&D tools according to their Functionality

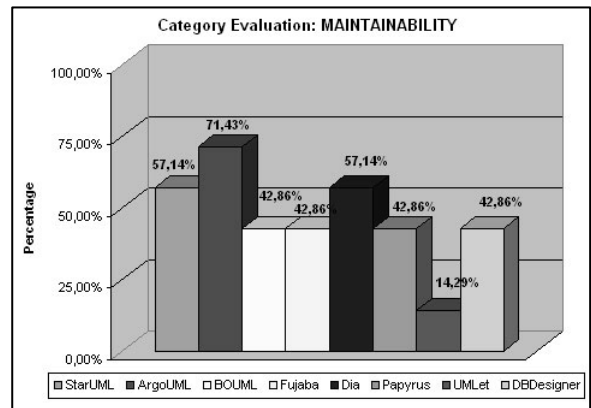


Figure 4: Comparison of FS-based A&D tools according to their Maintainability

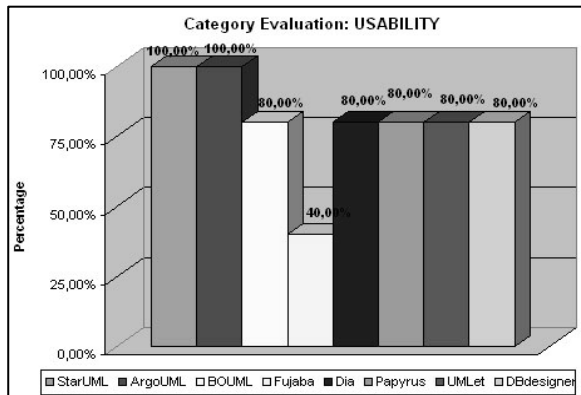


Figure 3: Comparison of FLOSS-based A&D tools according to their Usability

8. Conclusions and recommendations

This work introduces a quality model for FLOSS based A&D tools, which may be useful for software development companies that need to support or explain the selection of the most adequate tool for their environment. In addition, this research allowed making improvements to the Quality Systemic Model (MOSCA). Eight tools supporting A&D of software system were evaluated, namely StarUML, ArgoUML, BOUML, Fujaba, UMLet and Papyrus, which are UML modeling tools. These included DIA, which is a tool used for modeling different types of systems, with different types of languages, including UML. Lastly, we evaluated DBDesigner, a database modeling software. StarUML reached the highest

quality levels of all tool evaluated.

It should be noted that the MOSCA model may be adapted to specify the A&D tools quality in other contexts different than FLOSS, by selecting different categories and features in accordance with specific quality requirements. This is mainly due to the model systemic approach, which allows adapting evaluation to the application environment, taking into account quality required by users. Therefore, for future works, we might broaden the group of tools evaluated to include proprietary tools in order to perform a comparative evaluation of FLOSS-based A&D tools and proprietary tools.

Acknowledgments. This research was financed by FONACIT and DID-USB, Venezuela, through research projects G-2005000165 and S1-IN-CAI-003-07, respectively.

References

- [1] 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
- [2] BOUML. UML2 modeling tool. Version 2.30.2. Accessed July, 2007. <http://bouml.free.fr/>
- [3] Callaos, N. and Callaos, B. (1996) Designing with Systemic Total Quality, *International Conference on Information Systems*, Orlando, Florida, July, 548-560.
- [4] DBDesigner. Database modeling tool. Accessed July, 2007. <http://www.fabforce.net/dbdesigner4/>
- [5] Dromey, G. (1995) A Model for Software Product Quality, *IEEE Transactions on Software Engineering*, February, Vol 21, Nº 2, 146-162.
- [6] FlossMetrics, (2007). Free/Libre and Open Source Software Metrics. Accessed July, 2007 <http://flossmetrics.org/>
- [7] Free Software Foundation (2006). The Free Software Definition. Accessed October, 2006 <http://www.gnu.org/philosophy/free-sw.html>
- [8] Génova, G., Fuentes, J., Valiente, M. "Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional", *Novática*, 181:59-64, May-Jun 2006
- [9] ISO/IEC 9126-1:2001(E) Software engineering - Product quality - Part 1: Quality model. First edition,
- [10] Kendall, K, Kendall, J. (2007). *Systems Analysis and Design*. 7th Edition. Prentice Hall.
- [11] 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.
- [12] Mendoza, L., Pérez, M., Grimán, A. and Rojas, T. (2002) Algoritmo para la Evaluación de la Calidad Sistemica del Software Memorias de las 2das. Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC 2002) Salvador, Brasil.
- [13] Mendoza, L; Pérez, M. and Grimán, A. (2005). Prototipo de Modelo Sistemico de Calidad (MOSCA) del Software: *Computación y Sistemas*, 8, 3, 196-217.
- [14] Netcraft (2006). December 2006 Web Server Survey. Accessed September, 2006 <http://news.netcraft.com/>
- [15] Nichols, D., Thomson, K. Y Yeates, S. (2001). Usability and open-source software development. In *Proceedings of the Symposium on Computer Human Interaction*, (eds.) Kemp, E., Phillips, C., Kinshuk & Haynes, J., 6 July 2001, ACM SIGCHI New Zealand. 49-54. ISBN: 0-473-07559-8.
- [16] Open Source Initiative (2007). The Open Source Definition. Accessed July, 2007 <http://www.opensource.org/docs/osd>
- [17] 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,
- [18] 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
- [19] Ortega, M.; Pérez, M. and Rojas, T. (2002) A Systemic Quality Model for Evaluating Software Products, 6th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, Vol. I, 371-376.
- [20] Papyrus. Eclipse-based UML2.0 modeling tool. Version 1.7.1. Accessed July, 2007. <http://www.papyrusuml.org/>
- [21] Pressman, R. (2005). *Software Engineering: A Practitioner's Approach*. 6th Edition. Mc Graw Hill.
- [22] 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.
- [23] 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, <http://www.lisi.usb.ve/publicaciones>.

[24] Quintero, J. B., Anaya de Paez, R. "Marco de Referencia para la Evaluación de Herramientas basadas en MDA". X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software.

[25] 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.

[26] SQO-OSS (2006) Software Quality Observatory for Open Source Software. Accessed 09/23/2007 <http://www.sqo-oss.eu/>

[27] Sourcepyme (2006). Los institutos tecnológicos AIMME, AIMPLAS e ITI reúnen a más de 200 expertos en la Jornada Sourcepyme. Accessed December, 2006

<http://www.sourcepyme.org/?q=node/97>

[28] Wheeler, D. (2005). Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!. Accessed October, 2006 http://www.dwheeler.com/oss_fs_why.html

[29] ArgoUML. UML1.4 modeling tool. Version 0.24. Accessed July 2007 <http://argouml.tigris.org>

[30] DIA. Design tool including UML Version 0.96.1. Accessed July, 2007 <http://www.gnome.org/projects/dia/>

[31] StarUML UML2.0 modeling tool. Multilingual project. Version 5.0.2.1570. Accessed July, 2007 <http://staruml.sourceforge.net/en/>

[32] UMLet. UML 2.0 modeling tool. Version 7.1. Accessed July 2007 <http://www.umlet.com/>

[33] Fujaba. UML modeling tool. Version 4.1.0. Accessed July, 2007. <http://wwwcs.uni-paderborn.de/cs/fujaba/downloads/index.html>

Appendix A: New sub-features and metrics related to Functionality

Features	Sub-features	Questions	Formulation	Targeted to
Suitability (FUN 1)	Diagrams (FUN 1.1)	Does the tool allow drawing class diagrams?	1 yes - 0 no	User
		Does the tool allow drawing use case diagrams?	1 yes - 0 no	User
		Does the tool allow drawing WAE diagrams?	1 yes - 0 no	User
		Does the tool allow drawing E-R diagrams?	1 yes - 0 no	User
		Does the tool allow generating diagrams from a different one? (to support Traceability)	1 yes - 0 no	User
		Does the tool allow exporting diagrams to other extensions or formats?	1 yes - 0 no	User
	Documentation (FUN 1.2)	Does the tool allow generating documentation from diagrams?	1 yes - 0 no	User
Link A&D and implementation (FUN 1.3)	Does the tool allow generating code from diagrams?	1 yes - 0 no	User	
	Does the tool support reverse engineering of a diagram from code?	1 yes - 0 no	User	
Accuracy (FUN 2)	Abstraction details (FUN 2.1)	Does the tool allow specifying abstractions' names?	1 yes - 0 no	User
		Does the tool support abstractions' representation?(i.e. attributes with data types and class initial values, stimulus arguments in sequence diagrams, association cardinality)	1 No - 2 Regular - 3 Good - 4 Very good - 5 Excellent	User
Interoperability (FUN 3)	Interoperability (FUN 3)	Does the tool provide functionalities that belong to other software?	1 yes - 0 no	Developer
		Does the tool allow data exchange with other software?	1 yes - 0 no	Developer User
Correctness (FUN 5)	Complete (FUN 5.1)	Does the tool allow UML 2.1 abstractions and relationships?	1 No - 2 Regular - 3 Good - 4 Very good - 5 Excellent	Developer User
	Consistent (FUN 5.2)	Do the abstractions represented keep their properties?	1 No - 2 Regular - 3 Good - 4 Very good - 5 Excellent	Developer User
		Are the generated diagrams consistent with the initial ones?	1 No - 2 Regular - 3 Good - 4 Very good - 5 Excellent	User
Encapsulated (FUN 7)	Taxonomy (FUN 7.1)	Does the tool support diagrams grouped by known taxonomies as views, types (i. e. structure, behavior, interaction diagrams)	1 No - 2 Regular - 3 Good - 4 Very good - 5 Excellent	User

Appendix B: New sub-features and metrics related to Maintainability

Feature	Sub - feature	Query	Formulation	Targeted to
Changeability (MAB 2)	License (MAB 2.1)	Does the license allow access to the code?	1 yes - 0 no	Leader
		Does the license allow modifying the code?	1 yes - 0 no	Leader
		Does the license allow distributing the code?	1 yes - 0 no	Leader
		Does the license allow distributing the modified code?	1 yes - 0 no	Leader
		Does the license allow using the product without limitations of purpose, time and equipment number?	1 yes - 0 no	Leader
		Is the system documentation available?	1 yes - 0 no	Leader
Stability (MAB 3)	Stability	Number of security patches found and solved by the community.	1 From zero to 5 - 2 From six to ten - 3 From 11 to twenty - 4 ten (over twenty) - 5 hundred or thousands	Developer Leader
Software maturity attributes (MAB 8)	Software maturity attributes	Number of individuals involved in the development process.	1 From zero to five - 2 From six to ten - 3 From 11 to twenty - 4 ten (over twenty) - 5 hundred or thousands	Leader
		Number of downloads per year	1 From zero to five - 2 From six to ten - 3 From 11 to twenty - 4 ten (over twenty) - 5 hundred or thousands	Leader