Journal of Positive School Psychology http://journalppw.com
2022, Vol. 6, No. 4, 9126 – 9132

Storage System for Software Quality Metrics Associated with UML Diagrams

[1]Jairo A. Cortes M., [2]Luis Carlos Gutiérrez, [3]Jaime Alberto Paez Paez, [4]Fredys A. Simanca H., [5]Fabian Blanco Garrido

[1]Universidad Cooperativa de Colombia, jairo.cortes@campusucc.edu.co
[2]Universidad Cooperativa de Colombia, lgutier91320@universidadean.edu.co
[3]Universidad Cooperativa de Colombia, jaime.paez@campusucc.edu.co
[4]Universidad Cooperativa de Colombia, fredys.simanca@campusucc.edu.co
[5]Universidad Cooperativa de Colombia, fabian.blanco@campusucc.edu.co

Abstract

The project derived from the research "Software quality metrics system based on Unified Modeling Language - UML diagrams", which is being developed at the Universidad Cooperativa de Colombia, recommends the development of software quality metrics based on UML diagrams to identify early errors in its concept. The final objective is to support these indicators with tools that allow the work of software designers and developers and that they can evaluate the relationship that the diagram and its quality must have with the requirements. This research shows the results of a storage system for software quality metrics based on UML (Unified Modeling Language) diagrams. Information on software quality metrics was obtained mainly on the aforementioned diagrams represented in metamodels with XMI (XML (eXtensible Markup Language) Interchange) formats to facilitate the exchange of metadata between modeling tools based on UML.

Keywords: Metrics, UML, Repository, XMI; Software.

INTRODUCTION

In order to consider a UML (Unified Modeling Language) diagram well elaborated, it must be considered to have implemented correct quality standards and to guarantee these standards, certain quality models or metrics must be followed. This allows considering an efficient software in relation to its results, by means of the study of its attributes. Likewise, there are certain tools to determine the quality, which are focused on generating matrices and comparison diagrams to perform the analysis on these and determine if there are failures in the development. With this in mind, a research will be conducted, focusing on reengineering the rules, codes and guidelines based on UML diagrams, whether they are made by hand or generated by an application. In addition, the study will be focused on the OMG standards on UML integrated to XML-XMI to make a description of these models in OCL (Object Constraint Language) (OMG, 2011) and thus provide a research that expands the measurement base of these diagrams and can use a tool to predict the error in its early development.

This research focuses on the study of metrics and quality standards of UML diagrams, since having a guide on how to qualify the criteria of a software, it is easier to achieve a more efficient development area. By taking into account that the developments are homogeneous and comparable, it is easier to have an optimization of resources and cost estimates than not achieving the fulfillment of the requirements. The adjustment of the

development due to accumulated errors and not having an adequate development guide, implies a greater effort in the conception and implementation of systems. One of the fundamental aspects of software engineering is to generate high quality products, but multiple problems may occur when developing a product that may be associated with not understanding the requirements well, oversizing what is wanted to be done, not understanding the system and other factors (The Standish Group International, 2015).

Some of these drawbacks could become the way in which functionality, usability, safety, security, efficiency and performance can be measured among other factors before development. The adjustment of the development due to accumulated errors and not having an adequate development guide implies a greater effort in the conception and implementation of systems. One of the main objectives of this research is to design and develop a quality metrics storage system of the software based on UML diagrams.

Software Quality Metrics

OCL is known as a standard which is defined by OMG (Murchio, 2014), it is a standard language that allows describing expressions about a UML model or diagram (OMG, 2011). When talking about OCL it refers to a language with pure specification, and, therefore, it will be confirmed that the evaluation of an OCL expression does not have side effects or imply any consequence; when a OCL expression is evaluated, it will simply return a value and nothing is modified in the model; the evaluation of an OCL expression is instantaneous (OMG, 2011).

UML rules are both syntactic and semantic and can be described as follows:

- Names: Indicates names of elements, diagrams and relationships.
- Scope: Determines what each name means.
- Visibility: It is how these names can be visualized and made useful by others.
- Integrity: information about the way some elements are going to be related to others.
- Execution: It is the meaning of simulating or executing any dynamic model.

The models that are built during the development of a system can be abbreviated, incomplete or inconsistent (OMG, 2009).

Common mechanisms in UML, are applied in a consistent way by means of the language, such ways are classified into:

- Specifications.
- Adornments.
- Common divisions.

Software quality metrics have evolved through process, product and resource standards. There are traditional metrics such as Cyclomatic Complexity, McCabe and McCall metrics, there are also object-oriented metrics such as MOOSE, MOOD and those of Loren and Kidd, defined at the design level and there are others such as metrics for UML diagrams that are defined more at the conceptual level (Piattini et al, 2008, pp 68-119).

To describe some of these for example the MOOSE Metrics or known by CK Chidamber and Kemerer are class oriented metrics, they measure individual classes, inheritance and collaborations, it is one of the most referenced set of metrics.

Lorenz and Kidd metrics are class-oriented and measure size, values, inheritance, externals and internals. The UML or conceptual diagram metrics are divided according to the diagram, among the most used are the class diagram, cases of use and transition and states (Piattini et al, 2008, pp 68-119).

Materials and Methods

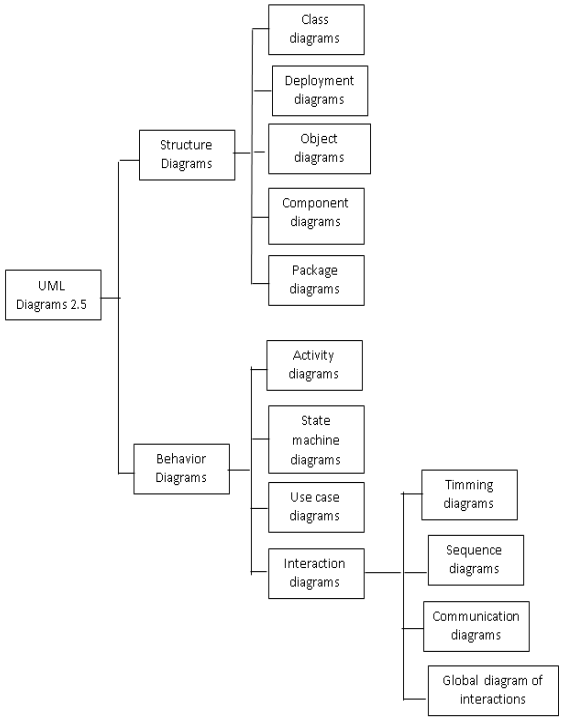
The methodology used is SCRUM, which aims to deliver values in very short periods of time and for this it is necessary to take into account

three fundamental pillars which are transparency, inspection and finally adaptation, this makes the project deliverables totally reliable and thus reviewing constantly the process or the development of the project (Sutherland, 2013). The SCRUM methodology is based on flexibility, on the adoption of modifications and the addition of new requirements, in the execution of a complex project; it is also based on the interaction with the client, the human factor and the iterative development with the intention of having good results (Sutherland, 2013). In the initial planning stages, the requirements were organized in the blacklog product as research was conducted on each of the UML diagrams (Booch et al, 2006) and the way in which metrics are associated to guarantee the accuracy of the requirements. Sprint is generated for the development of each of the diagrams and their associated metrics, and then sprint is generated for the storage process of the metrics and their validation with the requirements. For the organization and development of the project there were two main phases, the first one related to the collection of specific information on the UML diagrams and their associated metrics and the second phase, with the elaboration of the repository with its validation and verification process.

Results and Discussion

An identification of each of the UML diagrams is made according to the standard found in version 2.5, see figure 1.

Figure 1. *UML 2.5 Diagrams*



Today, UML modeling is found in UML version 2.5.1 (OMG, 2009), in which there can be two classifications of diagrams such as structural diagrams and behavioral diagrams.

Structure Diagrams

A structure diagram represents the internal architecture of a system at different levels of abstraction, with a special focus on the classes, the connections and/or interactions that they have, its general objective is to represent in a modular graphical way the components that a system has, leaving the input and output parameters defined.

The rules that are applied to the structure diagrams are not applied to follow some kind of standard but rather to reduce the complexity in case some modification arises where it is necessary to modify some structure. Some of the rules for structure diagrams are:

- When setting up a module, do not detail the internal logic of the module, only the inputs and outputs.

- Modules must be small with a concise action name.
- Modules must be independent of each other
- Modules have to perform a clear and simple function.
- There must be a data store to represent the system values
- Modules can be decomposed into other sub-modules to make the system easier to understand and modify
- There should be hierarchy between modules so that higher levels can more easily handle information from lower levels
- There must be a clear design strategy on which the information will be worked in order to choose the appropriate diagram.

One of the diagrams associated with this classification is the component diagram, which details the static design view of the components in a system, in addition to showing the organization and their respective dependencies between them. Some of the rules for object diagrams are:

- There is no need to confuse the object diagram with the class diagram, since the class diagram has operations.
- There is no need to add multiplicity between objects.
- Attributes must have assigned values within the software.
- A UML object diagram represents a specific instance of a class diagram at any given point in time.

An UML object diagram represents a specific single instance of a class diagram at any given time.

Behavioral Diagrams

A behavioral diagram is defined as a diagram that represents the sequence of states that an object undergoes in response to events throughout its life cycle. In other words, these diagrams show the different states of the process. Through these charts, the process to be programmed is shown graphically.

These charts are used to show, standardize and record the dynamics of the system. When we talk about software development, these dynamic aspects can be generated messages, data input operations, events, etc. These diagrams can be very simple or very complex, depending on the process they represent. They refer to objects because they are widely used in object-oriented programming. An object can be any entity with a specific state and behavior.

Some of the rules for use case diagrams are:

- Use cases must describe what the system to be developed should do when interacting with participants, not how it should do it. In other words, it must only describe externally observable behavior without going into the internal functions of the system.
- The name of the use case should describe what the participant intends to achieve by executing it.
- The use case should describe the interaction with the participants without explicitly referring to the specific user interface elements of the system to be developed.
- Calling certain use cases from other use cases should only be used as a mechanism to avoid repeating a certain sequence of steps that are repeated in multiple use cases. Never use it to express possible user interface menus.
- Try to ensure that all use cases in the same software requirements specification - ERS are described to the same level of detail.
- In a use case diagram, you should avoid crossing the connecting lines between the participants and the use case.

In Use Case Diagrams, system functions are symbolized from a user's point of view, which is called an "actor". For the case of activity diagrams, it is a special case of a state diagram, where the state is an action state and most transactions are sent at the end of the action performed with the previous state. This type of

chart allows parallelism of operations to show the decision paths that exist in the overall process.

Some of the rules for activity diagrams are:

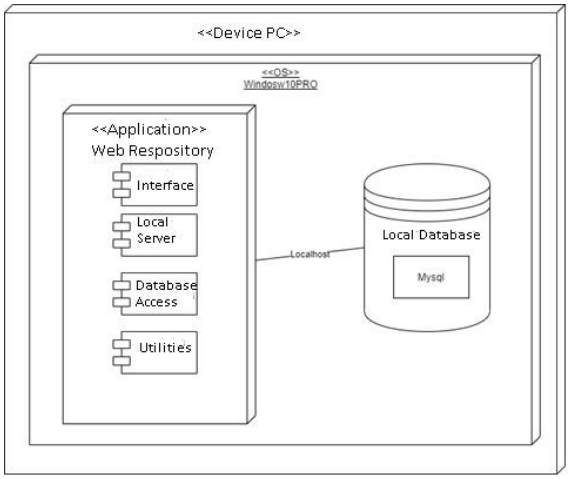
- "Opaque actions" act as a kind of item allocators or are known by a specific text syntax.
- Invoking an action is a step that directly or indirectly leads to a certain behavior.
- Object operations modify the state of the object (and the instance of the class). You can originate or delete them, relate them to other instances, read them, and then assign them to a class.
- The link operation will change the association (usually the relationship between the classifiers of the two classes) and their instances, and the behavior of the link.
- Structural feature actions determine the behavior of structural features in activity diagrams. To do this, they need an input pin because they are usually assigned an aesthetically specified object and a structural feature of the classifier.
- Variable operations affect statically specified variables defined by activities or structured activity nodes.
- Accept that even actions are hypothetical support points. This means that the activity is waiting for an event to occur in the event group of the context object. Actions have triggers, which can cause actions when one or more specified states occur.

In a UML behavior diagram, the control flow or object flow is shown, with special emphasis on the sequence and conditions of this flow.

In the research, each of the UML diagrams were described with their respective associated metrics (Genero et al, 2000), for example, the result for the class diagrams identified metrics such as the number of attributes, methods, associations, depth and width lengths, number of objects, the result for the use case diagrams identified metrics such as the number of actors and actions, number of interactions, extensions and includes. Thus, with each of the UML diagrams their metrics were identified in order to validate and store them in the repository that was designed, verifying them with the requirements of the applications.

According to the previous results, detailing each one of the diagrams and establishing the associated rules, it is possible to gather enough information to create a storage and control system of quality metrics associated to the UML diagrams. Likewise, it allows to fulfill the objectives proposed in the research. The development of a storage system is structured according to the architecture represented (Clements et al, 2012) in Figure 2.

Figure 2. *Metrics Storage System Architecture*



The description of the architecture is described in Table 1.

Table 1. *Software Architecture description*

Process or flow	Description	Result
Registration	Process by which the user about to log in (either common user or administrator user), provides his data to the application in order to generate a correct login to the application, thus safeguarding the user's personal information in the database.	User´s accessibility
Verification	In the process of validation by the user, the information	Verified information

	provided by the user will go through the administrator in order to make the correct validation and implementation of this within the system.	
CRUD	This process is exclusive to the administrator, since he/she is in charge of ensuring that the information uploaded to the application is accurate and consistent.	Verified information
Reading (Common user)	This process is simple since the administrator verifies the information that can be read.	Verified Information

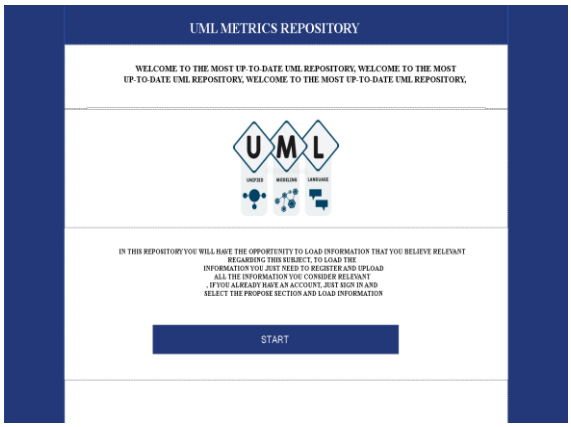
The variables used in the architecture (Clements et al, 2010) are detailed in Table 2.

Table 2. *Architecture Variables*

Input	Process	Output
Data (Personal Information)	Registration	User´s accessibility
UML Rule (Common user)	Verification	Verified Information
UML Rule (administrator user)	CRUD	Verified Information
Verified Information	Reading (Common user)	Verified Information

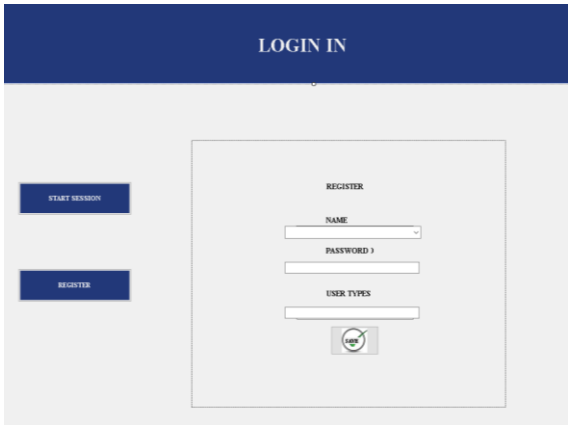
Some of the most representative interfaces, which are part of the storage system, are the following:

Figure 3. *Software presentation*



After entering the system, the user can enter and validate the metrics associated with each UML diagram. Then the user will find this window, in which he/she can enter the system, or alternatively he/she can register, and to do so he/she only has to fill in or complete the following form.

Figure 4. *System Access*



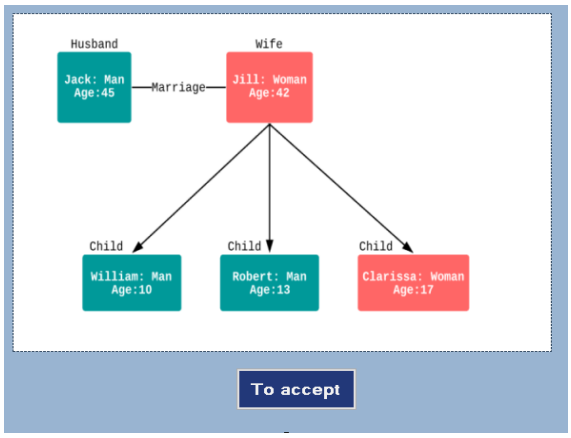
It should be clarified that this is the view of a user's registration from the perspective of the administrator user, since this will be the only one able to assign the type of user or role that will be assigned to a new user of the system.

Figure 5. *Types of diagrams to be entered and validated*

Once the diagram types section is selected and we choose the type of diagram we want to analyze, we can see in first instance the metrics already entered into the system, also with two

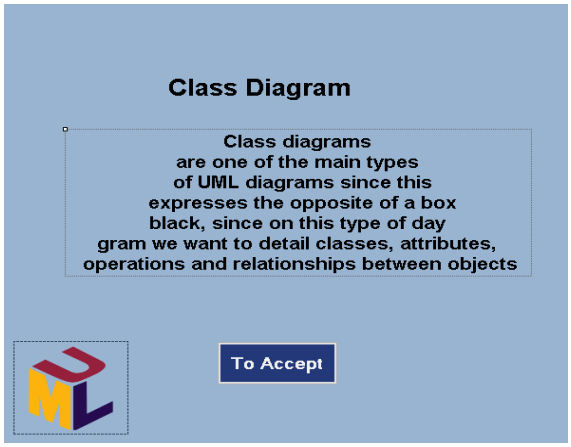
buttons in which interacting with them is validated with the requirements and in turn see a reference image of the diagram stipulated or otherwise the description of this.

Figure 6. *Diagram Examples*



Then a description of each diagram with its rules is made. See figure 7.

Figure 7. *Description of each diagram*



Conclusions

The design and development of a storage system for software quality metrics based on Unified Modeling Language - UML diagrams was completed.

It was possible to identify each UML diagram with its rules to register them in the storage system represented in metamodels with XMI formats developed by the Object Management Group - OMG.

A metrics storage system was designed for software quality using UML diagrams according to the OMG standard. Students and teachers developed tests on software usability, obtaining favorable results. The system allows to support the metrics indicators, applied in the tools that are used by software designers and developers, it also allows to evaluate the relationship between the diagram and its standard quality.

Reference

[1] Booch, G., Rumgaugh, J., Jacobson, I. (2006). El lenguaje unificado de modelado guía del usuario. Addison Wesley.

[2] Clements, P., Bass, L. (2012). Software Architecture in Practice. SEI: Series in Software Engineering.

[3] Clements, P., Bachmann, F. (2010).Documenting Software Architectures: Views and Beyond. SEI: Series in Software Engineering.

[4] Genero, M., Piattini, M. y Calero, C. Early Measures For UML class diagrams. L´Objet. 6(4), Hermes Science Publications, 489-515, 2000

[5] Murchio, S. (2014). Refinando UML: Object Constraint Language. Recuperado de: https://folderit.net/es/blog/refinando-uml-object-constraint-language-es/

[6] OMG (Object Mangement Group) 2009. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2. Recuperado de: http://www.omg.org

[7] OMG (Object Mangement Group) 2006. OMG Diagram Interchange. Recuperado de: http://www.omg.org

[8] OMG (Object Management Group) 2021. Lenguaje de Restricción de Objetos. Recuperado de https://www.omg.org/spec/OCL/2.3.1/

[9] Piattini, M., García, F., Garzas J., Genero, M. (2008). Medición y estimación del software. Técnicas y métodos para mejorar la calidad y la productividad. Alfaomega-Ra-Ma. pág. 332.

[10] Standishgroup(2015). Chaos Report 2015. Recuperado de: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

[11] Sutherland, J. (2013). Scrum. El Arte de Hacer el Doble de Trabajo en la Mitad de Tiempo. Editorial Oceano. Pag. 220.