Journal of Ubiquitous Systems & Pervasive Networks
Volume 8, No. 1 (2017) pp. 01-09

Software Quality Assessment Algorithm Based on Fuzzy Logic

Madjid Kara[a], Olfa Lamouchi[b], Amar Ramdane-Cherif [a],*
[a] *LISV Laboratory, University of Versailles St-Quentin-en-Yveline, 10-12 avenue de l'Europe, 78140 Velizy, France*
[b] *Engineers National School of Carthage, 45 Rue des entrepreneurs, 2035, Tunisia*

Abstract
In this paper an attempt has been made to provide a new global evaluation approach of a specified software quality model extracted from a generic software quality model using an instantiation procedure. The evaluation is based on data extracted from an ambient distributed system composed of fusion and fission agents connected to input/output services. These data are linked to the appropriate metrics of our software quality model and we use quality factors stated in ISO standards and different models of researchers represented under an ontology. We use equivalent relations to link criteria that have the same meaning and fuzzy logic approach to evaluate the entire software quality model. Our work presents the following contributions: (i) creating a generic software quality model based on several existing software quality standards and formalized under ontology concepts (ii) proposing an instantiation algorithm to extract specified software quality model from a generic software quality models (iii) proposing a new global evaluation approach of the specified software quality model using two processes, the first one executes metrics related to sensors data and the second one uses the result of the first process using fuzzy logic approach evaluating the entire specified software quality model and end up with a final numerical result (iv) adding the variability of metric variables algorithm to determine the impact of a possible variation of one criterion on others and avoid their penalization. This can help to conduct a trade-off-analysis in the proposed quality evaluation approach.

Keywords: *quality models, quality evaluation, quality measurement, metrics*

1. Introduction

Software quality model is a very useful instrument for software quality evaluation, which represents an important step in ensuring sufficiency of software product quality. Reliable software quality model is based on precise, objective and calculable metrics, defined without any ambiguity to provide an incontestable evaluation of quality [1]. The choice of data's appropriate representation is one of the most crucial tasks in the entire system development process [2]. The existing software quality models are generally hierarchical, grouping a set of factors, criteria and sub-criteria [3]. Several research works on software quality models have been completed and many classifications have been developed but the most important are: Mc Call and al-1977, Boehm and al-1978, FURPS Model-1992 and Dromey model-1995 [4]. The first finding was that: the proposed approaches were limited in their use fields, each researcher had its own criteria interpretation which led to having divergence in criterion definition [5], for example, we have nine different definitions for "completeness".

To group different software quality views, ISO/IEC 9126 [6] standard was created in 2003. An update was established as ISO/IEC CD 25010 [7] in 2007. It is used to establish software quality requirements and perform evaluations [8] using ISO/IEC 25023 standard [9] that contain a set of software quality basis measures. ISO/IEC 25010 has also been used as a reference for its reuse or extension. Among these works we can mention: Al-Badareen-2011, Dubey-2012, Al-Qutaish-2010 and Samadhiya-2013 [4]. Even if ISO models provide a solid theoretical basis and a better representation of information, they still too abstract and have some disadvantages, namely: lack guide for use in a global evaluation approach [10], difficulty of implementation, having to adapt it to each scenario without specific methods, no explicit link between criteria and metrics [11] to assign a detailed measure to factors, and Unavailability of some metrics variables does not allow us to evaluate their respective criteria.

The aim of our study is to present a new global evaluation approach of a software quality model extracted from an ambient distributed system. We have proposed a generic software quality models described in [12], based on several quality standards as well as models proposed by others researchers. Equivalence relations will be established between criteria of these standards. We also propose a global evaluation approach to evaluate our instantiated software quality model using two processes; the first one executes metrics related to sensors data and the second one uses these data to evaluate the software model using the principle of fuzzy logic [3, 13]. In addition, the variability of metric variables algorithm was

* Corresponding author. Tel.: +336.63.76.65.49
Fax: +01.39.25.49.85; E-mail: madjid.kara@lisv.uvsq.fr

DOI: 10.5383/JUSPN.08.01.001

proposed to determine the impact of a possible variation of one criterion on others to avoid their penalization. We could apply our approach in systems like Health Monitoring (SHM), Human Health Monitoring (HHM), habitat monitoring, and military surveillance [14]. Evaluation phase is important in the development of a new software quality models as a result of its adoption or rejection [15].

The rest of this article is organized as follows. In the next section, we present the proposed global approach adopted for our model and its different components. Then we define the different components of the software quality model and the algorithms, tools used for both global evaluation and variability of metric variables algorithms. At the end, we apply these algorithms to an example of a specified software quality model then we finish with an analysis results and conclusion.

2. Proposed global approach

The objective of our approach is to evaluate a software quality model involved in the interaction process following the steps 1 to 6 (see Fig.1), our approach is intended to be generally applicable to any software quality model. Data are extracted from an ambient distributed system composed of the fusion and fission agents connected to input/output services. These data are parsed, saved in an XML file and used to associate the sensors resources to the software quality model metrics. The specified software quality model is instantiated from a generic software quality models, taking into account the extracted data and a set of rules to establish an equivalent relationship between criteria of different standards. We will obtain a software model that has a tree structure and consists of several factors, criteria, and sub-criteria represented hierarchically. Each factor is composed of one or several criteria and each criterion is composed of one or more sub-criteria until reaching measurable criteria called leaves. The evaluation approach will use extracted data to calculate metrics of our specified software quality model. To quantify criteria, we use two processes; the first one executes metrics related to sensors data and the second one uses these data to evaluate the software model using fuzzy logic approach to evaluating the entire specified model and end up with a final numerical result.

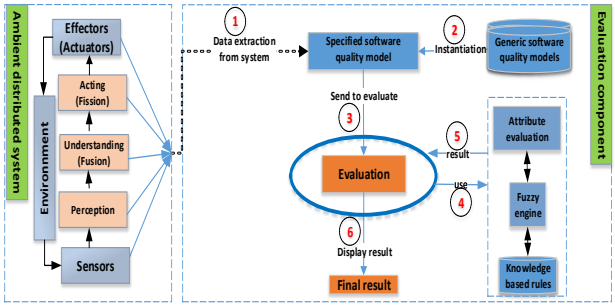

Fig. 1. Quality evaluation approach

3. Composition of software quality model

Composed of four main components (see Fig.2): criteria model, metric model, interdependence model and equivalence relations models. To formalize our model, we use an ontology that becomes a necessary and an important component of professional language in the computer field and knowledge representation. It is widely recognized as an appropriate knowledge representation technology; therefore, research on ontology is becoming more in demand for developing knowledge-based information systems [16]. By definition, ontology is a structure of concepts and relations representing the meaning of a given domain. It allows the representation of knowledge and it is used in the semantic web and artificial intelligence field. In the literature, we can find several definitions or meanings attributed to this concept. In 1993, Gruber proposed the definition most cited, it defines ontology as: "an explicit specification of a conceptualization" [17].

There are various ontology languages, based on different knowledge representation formalisms, and for our evaluation, we use the OWL (Web Ontology Language) ontology language. It is specified by classes, relations, and individuals [18]. Our software quality allows us to integrate software quality criteria, interoperate between quality factors and metrics in various fields and relate the different relations between these criteria. Users can introduce and modify all necessary information about software quality model

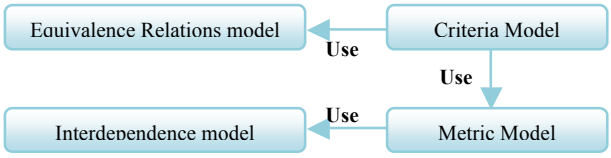

Fig. 2. Composition of software quality model

3.1. Criteria model

We define a criteria model by a list of factors, criteria, and sub-criteria, represented under a hierarchical form, each factor is composed of one or several criteria and each criterion is composed by one or several sub-criteria until reaching the lowest level of criteria called leaves criteria. We use the ¨composedBy¨ relationship to link each criterion with its direct sub-criteria and "equivalentTo" relationship to make links between criteria of different models. We note that a sub-criterion can be an intermediate criterion or a leaf criterion.

3.2. Metric model

A metric model is composed of metrics used to quantify the software quality models. Each metric contains a list of metrics variables that represent measurement functions or calculation procedures to attribute numerical values. A metric variable can be used by more than one metric and a metric can also be composed of another metric.

3.3. Interdependence model

This model describes the variation of metrics variables that allow us to satisfy the criteria quality of our model. The metrics variables variation is represented with the following variation signs:

Variation sign (+), the more the metric variable value is high, the more the criterion level is high (critical criterion).

Variation sign (-), the more the metric variable value is low, the more the criterion level is high (non-critical criterion).

Variation sign (*), the metric variable is neutral; its variation has no impact on criterion level (neutral criterion).

3.4. Equivalence relations model

This model is composed of equivalence relations between criteria that compose the generic software quality models. These relations link criteria that have the same meaning, they

are used to calculate metrics of a specific criterion that does not have metric variables. Each equivalence relation contains the equivalents criteria and an equivalence order, which define the priority of the chosen criterion when using the instantiation procedure.

3.5. Example of software quality model

In this example, the software quality model (see Table 1) is composed of three models (Model-01, Model-02, and Model-03). This model is based on the four components listed above. We use equivalence relations between metrics to specify the relations between criteria of different software quality models that have the same meaning. Using these equivalence relations and data coming from the distributed ambient system, we can instantiate from these models a specified software quality model related to the target system. In this example, the specified software quality model in our example is composed of Fact3, Fact4 and Fact7 factors (see Table 2). These factors are extracted from software quality models 2 and 3, taking into account the following equivalence relations Equiv1, Equiv2 and Equiv3 (see Fig. 3): Equiv1 (Ct6 ^ Ct3; 1): equivalence relation between Ct6 and Ct3 criteria with first order relationship, Equiv2 (Ct8 ^ Ct2; 1): equivalence relation between Ct8 and Ct2 criteria with first order relationship and Equiv3 (Ct8 ^ Ct11; 2): equivalence relation between Ct8 and Ct11 criteria with second order relationship.

Table 1. Composition of the generic software quality models

Software quality	Criteria model	Metrics model	Interdependence model	Equivalence relations
Model-01	Quality={Fact1;Fact2}. Fact1={Ct1; Ct2}; Fact2={Ct3}. Ct1={Ct4; Ct5} . Leaves list={Ct4;Ct5;Ct2;Ct3}.	M1={Mv2}, M2={Mv3}, M3={Mv1; Mv8}, M4={Mv4; Mv5}.	Ct4={M1;(Mv2,+)}, Ct5={M2; (Mv3,-)}, Ct2={M3;(Mv1,+);(Mv8,-)}, Ct3={M4;(Mv4,+);(Mv5,-)} .	Equiv1 (Ct6 ^ Ct3; 1). Equiv2 (Ct8 ^ Ct2; 1).
Model-02	Quality ={Fact3;Fact4}. Fact3={Ct6}; Fact4={Ct7}. Ct7={Ct8;Ct9}. Leaves list ={Ct6;Ct8;Ct9}.	M5={Mv5 ; Mv6}, M6={Mv7;Mv8 ; Mv9}, M7={Mv3}.	Ct6={M5; (Mv5,+);(Mv6,-)}, Ct8={M6 ; (Mv7,*);(Mv8,-) ; (Mv9,+)}, Ct9={M7 ; (Mv3,+)}.	Equiv1 (Ct6 ^ Ct3; 1). Equiv2 (Ct8 ^ Ct2; 1). Equiv3 (Ct8 ^ Ct11; 2).
Model-03	Quality ={Fact5;Fact6;Fact7}. Fact5={Ct10}, Fact6={Ct11}, Fact7={Ct12,Ct13}. Ct13={Ct14,Ct15}. Leaves list ={Ct10;Ct 11; Ct12;Ct14;Ct15}.	M8={Mv11}, M9={Mv9;Mv13}, M10={Mv10, Mv12}, M11={Mv10}, M12={Mv12, Mv14;Mv15}.	Ct10={M8; (Mv11,+)}, Ct11={M9;(Mv9,+);(Mv13,-)}, Ct12={M10;(Mv10,-);(Mv12,+)}, Ct14={M11;(Mv10,+)}, Ct15={M12;(Mv12,-);(Mv14,-); (Mv15,+);}.	Equiv3 (Ct8 ^ Ct11; 2).

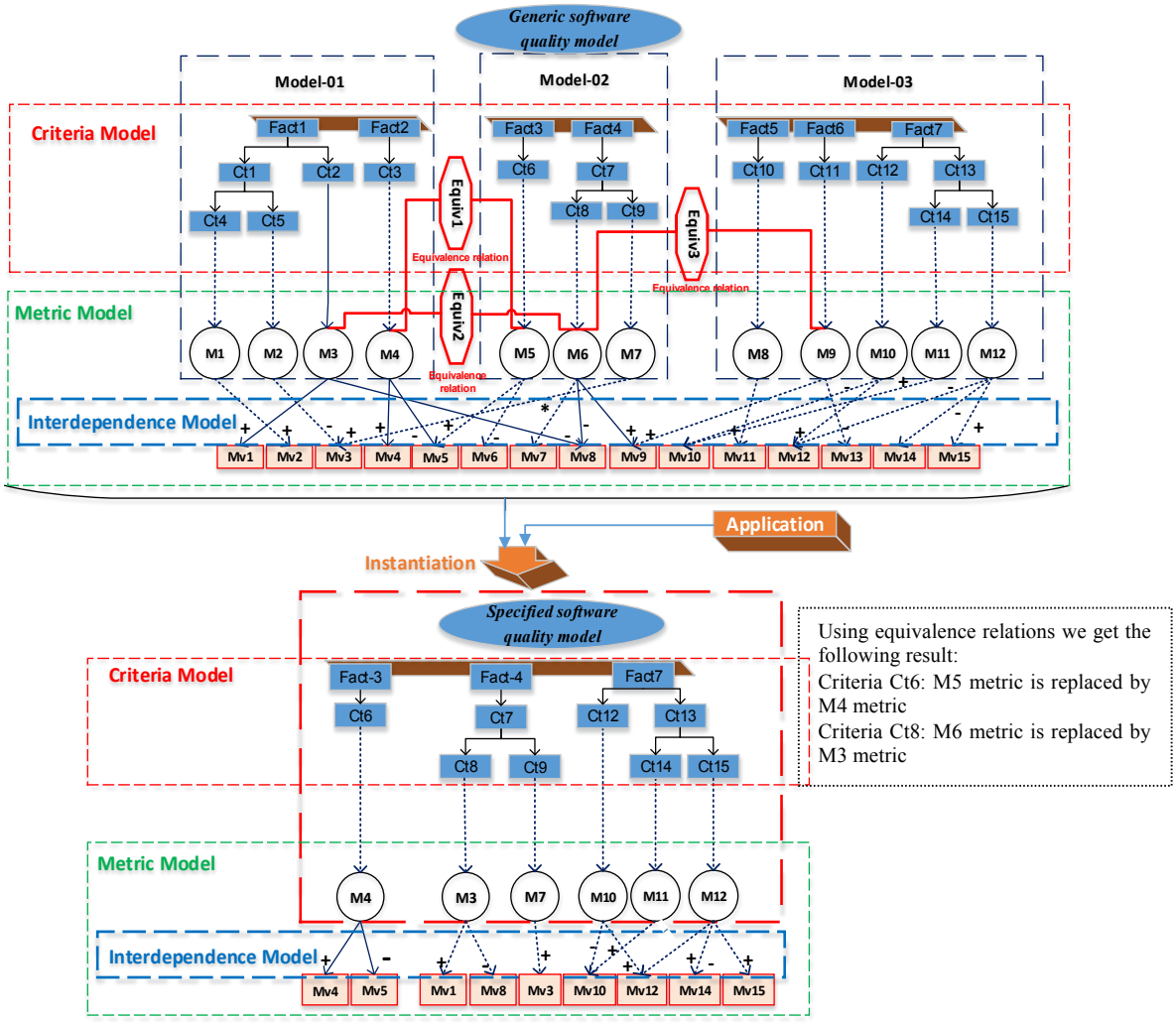

Fig .3. Software quality model

Table2. Composition of the specified software quality model

Model	Criteria model	Metrics model	Interdependence model
Specified software quality model	Quality ={Fact3;Fact4;Fact7}. Fact3={Ct6},Fact4={Ct7}, Fact7={Ct12;Ct13}. Ct7={Ct8;Ct9},Ct13={Ct14;Ct15}. Leaves list ={Ct6;Ct8; Ct9;Ct12;Ct14;Ct15}.	M4={Mv4; Mv5}, M3={Mv1; Mv8}, M7={Mv3},M10={Mv10, Mv12}, M11={Mv10}, M12={Mv12, Mv14, Mv15}.	Ct6={M4;(Mv4,+);(Mv5,-)}, Ct8={M3;(Mv1,+);(Mv8,-)}, Ct9={M7;(Mv3,+)}, Ct12={M10;(Mv10,-);(Mv12,+)}, Ct14={M11 ;(Mv10,+)}, Ct15={M12; (Mv12,-); (Mv14,-); (Mv15,+)}

As mentioned in the interdependence model (3.3), the metric variables variation is representing by (+), (-) or (*) symbol. We illustrate these variations by taking the specified software quality model (see Fig.3) and we will get the result represented in the interdependence table (see Table3). We get an interdependent couple (Ct14, Mv10) and (Ct12, Mv10), when we increase the value of Mv10, the value of Ct14 increase and when we decrease the value of Mv10, the value of Ct12 increase.

Table3. Interdependence table

	Mv1	Mv3	Mv4	Mv5	Mv8	Mv10	Mv12	Mv14	Mv15
Ct6			+	-					
Ct8	+				-				
Ct9		+							
Ct12						-	+		
Ct14						+			
Ct15							-	-	+

4. Instantiation procedure

The specified software quality model will be instantiated from a generic software quality models, taking into account data captured from sensors resources and equivalence relations defined between metrics of different software quality models applying algorithm 1 and algorithm 2. First, we need to identify and collect data captured from sensors. These data will be parsed, saved to an XML file, the extracted data is associated with generic software quality models to define metrics and criteria we want to evaluate. Then, we use equivalence relations between metrics of different software quality models to disambiguate some criteria of different models that have different names but the same meaning. They are used to calculate metrics that do not have complete metrics variables. We take an example to calculate M6 metric (see Fig. 3). In case we do not have Mv7 value metric variable, we use Equiv2 equivalence relation between Ct8 and Ct2 criteria. If all metrics variables (Mv1 and Mv8) are available, we calculate M3 metric value and return it to M6 metric. But if it is not the case, then we check for another relation with another order that gives us Equiv3 in our example and the M9 metric value will be affected automatically to M6 metric. Following this approach, M6 metric of Model-02 is replaced by M3 metric of Model-01. The same procedure was applied using equivalence relation Equiv1where M5 metric of Model-02 is replaced by M4 metric of Model-01. We use also default values for some neutral metrics variables which have no impact on criteria. In case, we do not have complete metrics variables for a specified metric and also no equivalence relation; we can use these default values only for neutral metrics variables.

Algorithm1: Instantiation algorithm

```
Start Program: Instantiation_algorithm
result=true
Read Ontology,
Extraction_leaves_list,
  For each Leaves_list
    Do
      If  verification_ procedure_Mv (leaf) then
        Mv (leaf) complete,
        Select (list_Mv)
        Continue
      Else
        leaf<- Equiv (leaf, result)
        continue
      EndIf
  End For
End Program
```

Algorithm2: Verification_procedure_Mv (leaf)

```
Start Verification_procedure_Mv (leaf)
result=false
list_Mv <- Search_Mv (leaf)
list_Mv_empty <- list_Mv_null (list_Mv)
  If number (list_Mv_empty) != 0 then
   Default_value <- Search_default_value(list_Mv_empty)
      If not (Mv_has_default_value) then
          result=false
      EndIf
  EndIf
 return result
End Procedure
```

5. Global evaluation

To evaluate our software quality model, we propose a global evaluation approach that evaluates all factors of the specified software quality model. Once interesting and relevant criteria are chosen and their metrics defined, we pass to the global evaluation of our instantiated model. The evaluation approach is represented in the following figure (see Fig. 4). Using data captured from sensors, variable metrics, which correspond to leaves criteria, are defined. The metrics are quantified with formulas and procedures and the value is stocked in a file. The result is routed to the part "fuzzy interpreter" that uses knowledge-based rules to evaluate the rest of criteria and factors and end up with final numerical values

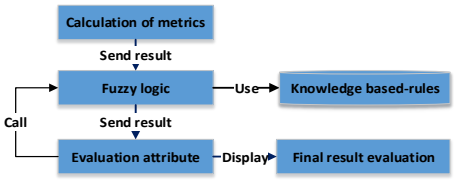

Fig .4. Global evaluation approach

5.1. Fuzzy Interpreter

Software quality models criteria are not very easily to measure and to quantify. Many attempts have been made to exactly quantify software quality parameters using various models. In this paper, an attempt has been made to provide a tool for precisely quantifying software quality factors. From which value, we can consider that criterion is very good, good, medium or weak? The difficulty is more as this value has an impact on the final evaluation of quality factors. In order to counter this difficulty, we propose the use of a fuzzy threshold. The concept of the fuzzy system was first conceived by Lofti Zadeh in 1965, who presented it as a way of processing data by allowing a partial membership.

5.1.1. Fuzzification step

The first step is to determine the degree to which these inputs belong to each of the appropriate fuzzy sets. The membership function is a graphical representation of the inputs participation degree describing the system. In the example (see fig. 5) we use the same linguistic variables model for Ct14 and Ct15 criteria. Its set of values can be: T = [Very Low, Low, Medium, High and Very High] where each term T is characterized by a fuzzy set in an interval U = [0, 1]. We will linguistically express the validation levels of criteria then project them on [0, 1] interval. Fuzzy logic allows a degree of truth to these criteria. For this example, Ct14 criterion is medium with a degree of truth of 0,4 and high with a degree of truth of 0,6; the Ct15 criterion is very high with a degree of truth of 0,4 and high with a degree of truth of 0,57.

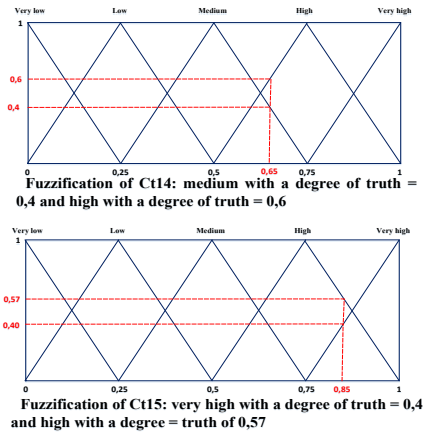

Fig. 5. Membership functions of Ct14 and Ct15 criteria

5.1.2. Inference step

Fuzzy Logic incorporates a simple, rule-based "If X and Y then Z" approach for solving the problem rather than solving it mathematically. The set of fuzzy inference rules is the knowledge base of the fuzzy controller. These rules are made by experts and we have 13 rules in our example (See fig. 6).

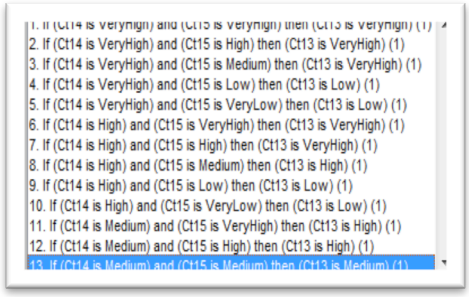

Fig. 6. fuzzy inference rules (Ct14 and Ct15 criteria

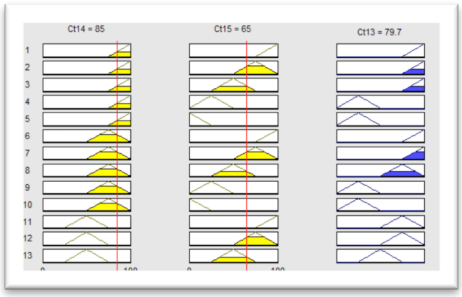

Fig. 7. Combination of fuzzified inputs and fuzzy rules

Then we combine the fuzzified inputs according to the fuzzy rules to get the result. We apply Mandeni's minimum operator [19] (See Fig 7) and we aggregate the result using the Min/Max technique (see Fig. 8).

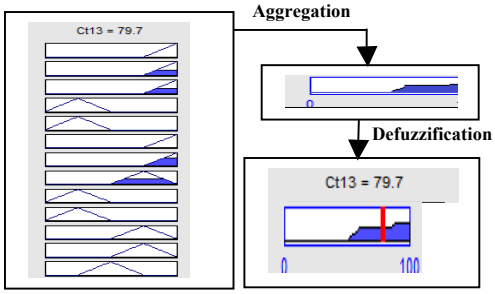

Fig. 8. Aggregation & Defuzification

5.1.3. Defuzzification step

Defuzzification is the process of converting the fuzzy sets into real-time data. The Centroid Method [20] has been adopted in this paper to defuzzify the triangular fuzzy sets. It is characterized by a red line with a projection on the x-axis (see Fig.9). In the end, we get a numerical result for Ct13 criterion=79.7.

5.2. Global evaluation algorithm

Our evaluation algorithm (see Algorithm 3), allows us to read and extract criteria and sub-criteria from the specified software model. Leaves criteria will be defined and evaluated with their own metrics or inherited from other standards. We call node each criterion with its direct sub-criteria. The second part of our algorithm breaks down the specified software model into nodes segments with a respected order. Nodes segments evaluation will use fuzzy logic to have a final numerical result. The implementation of the fuzzy controller will be performed using scripts to define Fuzzy Inference System (FIS) file for each node and get a final numerical result of factors.

Algorithm3: Global evaluation

Start Program Global Evaluation algorithm
  If leaves_list is Not__Empty **then**

```
   For each element of leaves_list do
    father←subClass(1ST_element)
    sons_list_←Superclass(true) (father)
    If sons_list Exists in leaves_list then
      Delete sons_list ( leaves_list)
      Add father(leaves_list), Order←file_order
    Else Move sons_list to the end of leaves_list
    EndIf
  EndIf
  Evaluate_leaves, Create_FIS_files
  Fuzzy_evaluation, Display_result
End Programm
```

6. Variability of metric variables algorithm

It is to be noted that by getting a "high score" in any of the criteria does not simply mean that a high quality of software has been achieved. Somme criteria may be important only in conjunction with others criteria, we could say that they are a complement to each other. In this algorithm, we will determine the impact of a possible variation level of one criterion on other's and propose a solution to avoid the penalization of the other criteria. In fact, we seek to optimize a criterion by modifying its metric variables and by doing this we could affect and penalize other criteria. In our example (see figure 15), (Ct14, Mv10) and (Ct12, Mv10) are interdependent and by varying Mv10 metric variable of Ct14 criterion we will affect M12 criterion. The purpose of this evaluation approach is to find all metric variables concerned by this variation to maintain a high-quality level of all criteria without any penalization.
This approach is described by the following algorithms:

Algorithm4. Variability of metric variables (parameter)

```
Start Program Variability of metric variables (Param)
If not empty (parameter) then
  Read (interdependence_table)
  Extraction (interdependence_parameter(parameter))
  Manage_interdependences (parameter,interdependence_table)
  Sort (result_table)
EndIf
End Program
```

Algorithm5. Procedure manage interdependences

```
Start Procedure manage interdependences
      (parameter, interdependence_table)
Line<-Extract_Line(table)
While Not End Processing(table) do
 If Treated_Criterion(line) then
    Update_Table(table, line(1), result_table)
    Line<-Extract_Line(table)
 Else
 If Interdependence_Criterion(parameter,line(2)) then
    Result(result_table, Equivalent(line), 'No')
    Treated_line(table.line(3))
    Update_Table(table, line(2), result_table)
    Line<-Extract_Line(table)
  ElseIf Not Dependant_Criterion (table, line(2)) then
    Result(result_table, Equivalent(line), 'Yes')
    Treated_Line(table.line(3))
    Update(table, line(2), result_table)
    Line<-Extraire Line(table)
   Else Line<-Extract_Dependant_Criterion(table,line(2))
  EndIf
 EndIf
EndIf
End While
Update_Result(result_table,table)
End Procedure
```

Algorithm6. Procedure Update result (result_table,table)

```
Start procedure Update_resul (result_table,table)
While Not End Processing(table) do
  Line<-extract_line(table)
If (Line_Result(line)='No') then
  If (Other_Result_Dependent_Criterion(ligne(2))='Yes')
    then Update_Result(result_table,line)
 EndIf
EndIf
End While
End Procedure
```

Applying the variability of metric variables algorithm on the specified software model in the example shown in Figure 3, to optimize Ct14 criterion at its metric variable Mv10, we will get the result presented in Table 4.

Table 4. Result of variability of metric variables algorithm

Factor	Leaf criterion	Criteria	metric variables
Fact 7	(Ct14, Mv10, +)	Ct12	(Mv12, +),
		Ct15	(Mv14, -) (Mv15, +)

To update the metric variable Mv10 of Ct14 criterion without penalizing Ct12 and Ct15, we have two solutions (see Table 4). The first solution: (Ct14, Mv10) (Ct12, Mv12) (Ct15, Mv14) : we intervene in Mv12 and Mv14 metric variables and the second solution: (Ct14, Mv10) (Ct12, Mv12) (Ct15, Mv15): we intervene in Mv12 and Mv15 metric variables.

7. Case study

This is an example of a specified software quality model to evaluate a representative Client/Server software architecture used as a system that receives data from an ambient distributed system. In this example, we will focus on the evaluation method without relating the instantiation step from the generic software quality models. For modeling, we use the ontology concept [21]; it offers many optional components such as reasoning and graphical interface. Data extracted will be adapted to leaves criteria of the specified software quality model. Our model is composed of Performance, Availability and Security factors (see Fig. 9). Performance is composed of Latency, and Jitter, Availability is composed of Software Availability and Hardware Availability, Security is composed of Attack Level1, Attack Level2, and Attack Level3, Latency is composed of Worst latency and Best latency. The leaves list of our model is composed of Worst latency, Best latency, Jitter, Attack Level1, Attack Level2, Attack Level3, Software Availability and Hardware Availability. Each leaf criterion has one metric variable (See Table 5) From this model we extract an interdependence table (see Table 6) which describes the interdependence between criteria using the variable metrics signs. The relation between two metric variables Mva and Mvb

can be neutral (*: the variation of Mva does not affect the variation of Mvb), direct (+: a positive variation of Mva causes a positive variation of Mvb) or inverse (-): a positive variation of Mva causes a negative variation of Mvb.

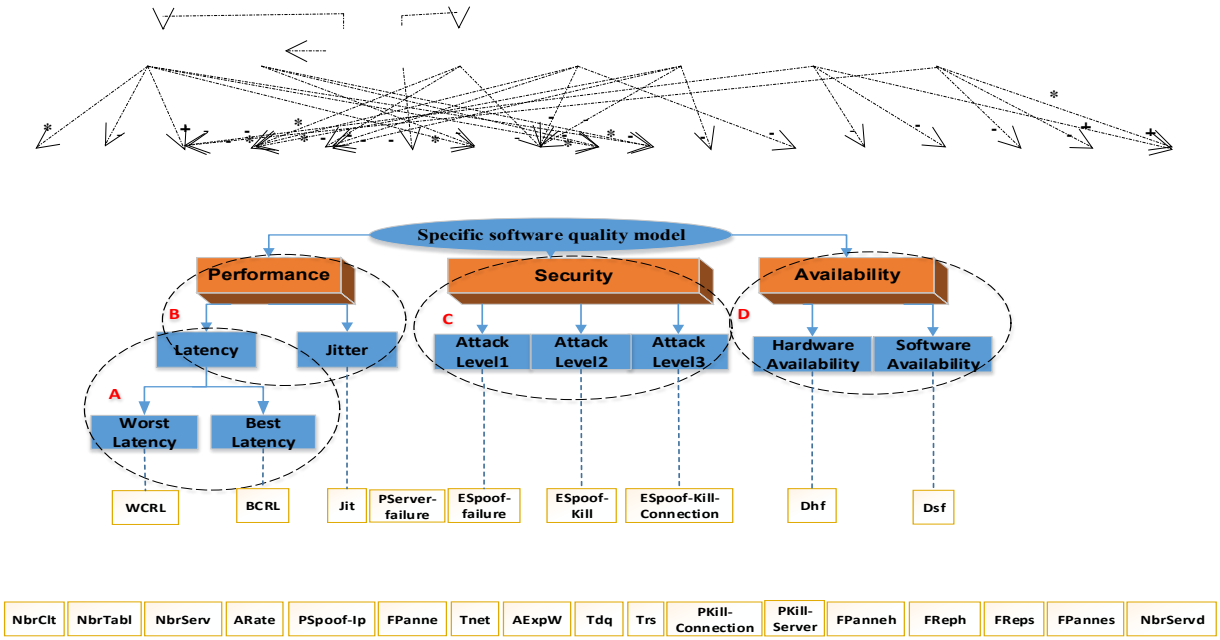

Fig. 9. Specific software quality model

Table 5. Composition of metric model

Metrics model	Description of the used metrics variables
WCRL={NbrClt; NbrServ; NbrTabl;Tnet; Tdq; TRs}, BCRL={Tnet; Tdq; TRs}, Dhf={NbrServ, Fpanneh, FReph, NbrServd} Dsf={NbrServ, Fpannes, FReps} JiT={ WCRL, BCRL } PServer-failure={NbrServ, AExpW, FPanne}, ESpoof-failure={PServer-failure, AExpW, ARate, PSpoof-IP}, ESpoof-Kill={AExpW, ARate, PKill-Server, PSpoof-Ip}, ESpoof-Kill-Connection={AExpW, ARate, PKill-Connection, PSpoof-Ip}.	NbrClt: Nbr of client, NbrServ: Nbr of server, NbrServd: Nbr of available server, Tnet: passage time of a request by the network, Tdq: waiting time of the request into the input queue, TRs: time to search for the query in the table, FPanneh: Software fault frequency, FPannes: Equipment failure frequency, FPanne: Failure frequency, FReph: Software repair frequency, FReps: Equipment repair frequency, AExpW: time in which the system is vulnerable to attack, ARate:frequency of piracy attack, NbrTabl: Table numbers, PSpoof IP: probability of a successful IP spoofing, PKill Server:probability of inhibiting a server, PKill

Table 6. Interdependence table:

	NbrClt	NbrServ	NbrTabl	Tnet	Tdq	TRs	Fpannes	FReps	FReph	FPanneh	ARate	PS-Poof-IP	FPanne	AExpW	PKill-Connection	PKill-Server	NbrServd
Worst latency	*	+	-	*	*	*											
Best latency				-	-	-											
Jitter	*	*	*	*	*	*											
Attack Level1		-									*	-	-	-			
Attack Level2											*	-		-		-	
Attack Level3											*	-		-	-		
S- Availability		-					-		-								
H- Availability		-						-		-							+

To evaluate the specified software quality model we apply the global evaluation algorithm. First, we calculate metrics of leaves criteria using formulas (See Table 7).

Table 7. Metric calculation

Leaves & Metrric=formulas & value
WorstLatency: WCRL= NbrClt / NbrServ * NbrTabl * (2*Tnet + 2*Tdq + Trs) **= 71**
BestLatency: BCRL= 2*Tnet + 2*Tdq + Trs **= 61**
H-Availability: Dhf=(NbrServ!(FPanneh/ FReph)NbrServ)/((1+ NbrServerd=1(FPanneh/ FReph)NbrServd) *(NbrServ!/(NbrServ-NbrServd)!)) **=60**
S-Availability:Dsf=(NbrServ!(FPannes/ FReps)NbrServ)/((1+NbrServerd=1(FPannes/ FReps)NbrServd) *(NbrServ!/(NbrServ-NbrServd)!)) **= 90.1**
Jitter: Jit= WCRL – BCRL **= 36.8**
AttackLevel1: ESpoof-failure= AExpW * ARate * PServer-failure*PSpoof-IP**=12,**PServer-failure=1-exp-FPanne.NbrServ.AExpW
AttackLevel2: ESpoof-Kill = AExpW * ARate * PKill-Server*PSpoof-Ip **= 25**
AttackLevel3: ESpoof-Kill-Connection= AExpW * ARate * PKill-Connection*PSpoof-IP **= 36**

The node segments evaluation (shown by dotted circles in Fig. 10) and all variants execution order will be established. In our example, node "A" must be evaluated before "B" because Latency criterion must be evaluated before Performance factor.

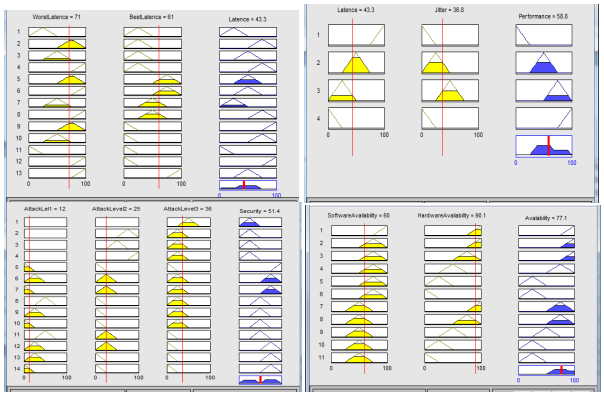

Fig. 10. Final result

To create inference system, we use Matlab TOOLBOX fuzzy logic. This tool generates Fuzzy Inference System files: security.fis, availability.fis, latency.fis and performance.fis. Using the gravity center method, characterized by a red line with a projection on x-axis, we get the final evaluation result of the specified software quality model (See Fig.10) (Security=51.4%, Availability=77,1%, Performance=58,8%)

During the tests we have done, when we increase the number of servers (NbrServ) the performance increase and the security decrease but if we decrease the number of servers the security increase and the performance decrease. Increasing the number of servers is essential to improve the performance of the system, by using the variability of metric variables algorithm; we will determine the impact of a possible variation level of the metric variable "NbrServ" to avoid the penalization of the security or the availability of the system. The execution result of this algorithm is resumed in Table 8. To increase number of servers without penalizing security and availability factors, we have to intervene on one of the Hardware Availability metric variables ((Fpanneh, -) or (Freph, -) or (NbrServd, +)), on one of the Software Availability metric variables ((Fpannes, -) or (FReps, -) or (NbrServd, +)) and on one of the Attack Level1 metric variables ((FPanne, -) or (AExpW, -) or (PS_Poof-IP,-)).

Table 8. Result of the variability of metric variables

Factor	Leaf criterion	Criteria	metric variable
			(Fpanneh, -)
		Hardware Availability	(Freph, -)
			(NbrServd, +)
Performance	(Worst Latency, NbrServ, +)		
			(Fpannes, -)
		Software Availability	(FReps, -)
			(NbrServd, +)
			(FPanne, -)
		Attack level1	(AExpW, -)
			(PS_Poof-IP,-)

9. Conclusion

Modeling and evaluating software quality model is an important step of decision making. The main purpose of software engineering is to find the best solutions to improve software quality. In this paper, we have proposed a generic software quality model for an ambient distributed system. A specified software quality model was extracted from a generic software quality models. The instantiated model comes out as a collection of factors, criteria, and sub-criteria until leaves criteria. The last level of criteria is linked to different software metrics and measurement procedures. The model is based on ontology where we can add equivalence relations between different attributes belonging to several software quality models. To integrate these models, we have proposed an instantiation algorithm that allows us to derive a specified software quality model from the generic software quality models. To evaluate the rest of criteria starting up from leaves to factors, we use an evaluation approach based on fuzzy logic. We have also presented a variability of metric variables algorithm to determine the impact of a possible variation of one criterion on others and avoid their penalization. Then we have shown our approach through a Client/Server architecture. We plan to study the problem of interaction between models criteria, enrich evaluation methodology taking into account the sign of metrics variables.

References

[1] Grubb, P. etTakang, A. A. (2003). Software Maintenance : Concepts and Practice. World Scientific, 2nd edition. https://doi.org/10.1142/5318

[2] Moody, D.L., 2003. Measuring the quality of data models: an empirical evaluation of the use of quality metrics in practice. ECIS 2003 Proceedings 78.

[3] Lamouchi, O., Cherif, A.R., Lévy, N., 2008. A framework based measurements for evaluating an IS quality, in: Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling-Volume 79. Australian Computer Society, Inc., pp. 39–47.

[4] P. Miguel, J., Mauricio, D., Rodríguez, G., 2014. A Review of Software Quality Models for the Evaluation of Software Products. International Journal of Software Engineering & Applications 5, 31–53. https://doi.org/10.5121/ijsea.2014.5603

[5] K. Mehmood, A Quality Pattern Based Approach for the Analysis and Design of Information Systems, Citeseer, 2014.

[6] ISO/IEC IS 9126-1. (2001). Software Engineering - Product Quality – Part 1: Quality Model. International Organization for Standarization, Geneva, Switzerland.

[7] ISO/IEC 25010-JTC1/SC7/WG6 Software Engineering Software product Quality Requirements and Evaluation (SQuaRE)-Software and quality in use models. 2008.
[8] Lew,P, Olsina,L, Zhang, L,2010. Quality, quality in use, actual usability and user experience as key drivers for web application evaluation. Springer.
[9] ISO/IEC DIS 25023 - JTC1/SC7/WG6 Systems and software engineering -Systems and software Quality Requirements and E valuation (SQuaRE) - Measurement of system and software product quality. Date: 14-nov.-16. https://doi.org/10.1007/978-3-642-13911-6_15
[10] Bouzeghoub, M., Calabretto, S., Denos, N., Harrathi, R., Kostadinov, D., Nguyen, A.-T., Peralta, V., 2007. Accès personnalisé aux informations: approche dirigée par la qualité., in: INFORSID. pp. 105–120.
[11] Mordal, K., n.d. Analyse et conception d'un modèle de qualité logiciel.
[12] Madjid Kara, Olfa Lamouchi, Amar Ramdane-Cherif "Ontology Software Quality Model for Fuzzy Logic Evaluation Approach". The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / . Volume 83, 2016, Pages 637–641. https://doi.org/10.1016/j.procs.2016.04.143
[13] Challa J.S., Paul, A., Dada, Y., Nerella, V., Srivastava, P.R., Singh, A.P., 2011. Integrated Software Quality Evaluation: A Fuzzy Multi-Criteria Approach. Journal of Information Processing Systems 7, 473–518. https://doi.org/10.3745/JIPS.2011.7.3.473
[14] Mahyoub, M., Al-Roubaiey, A., Ahmed, G., 2016. Content-based Filter Publish Subscribe Model for Real-time WSN applications. Journal of Ubiquitous Systems & Pervasive Networks 7, 19–27.
[15] Osman, F.A., 2015. Healthcare Providers' Attitudes toward Using the Technology of Smart Health Cards. Journal of Ubiquitous Systems & Pervasive Networks 6, 11–17.
[16] Liu J.N.K., He, Y.-L., Lim, E.H.Y., Wang, X.-Z., 2013.A New Method for Knowledge and Information Management Domain Ontology Graph Model. IEEE Transactions on Systems, Man, and Cybernetics: Systems 43, 115–127. https://doi.org/10.1109/TSMCA.2012.2196431
[17] Boudra, M., Hina, M.D., Ramdane-Cherif, A., Tadj, C., 2015. Architecture and Ontological Modelling for Assisted Driving and Interaction. International Journal of Advanced Computer Research 5, 270.
[18] http://ontogenesis.knowledgeblog.org/514.
[19] Iancu, I., 2012. A Mamdani type fuzzy logic controller. INTECH Open Access Publisher. https://doi.org/10.5772/36321
[20] T.J. Ross, Fuzzy Logic with Engineering Applications,2nd Ed, Wiley India Pvt. Ltd, New Delhi, India, 2004.
[21] Liu J.N.K., He, Y.-L., Lim, E.H.Y., Wang, X.-Z., 2013.A New Method for Knowledge and Information Management Domain Ontology Graph Model. IEEE Transactions on Systems, Man, and Cybernetics: Systems 43, 115–127. https://doi.org/10.1109/TSMCA.2012.2196431