SOFTWARE ARCHITECTURE EVALUATION APPROACH

Olfa Lamouchi[1], Amar Ramdane-Cherif[1,2] and Nicole Lévy[1]
[1]PRISM, [2]LISV, UVSQ, 45, Avenue des Etats-Unis, 78035 Versailles Cedex, France

Keywords: Quality evaluation, Fuzzy systems, Quality measurement, Quality model, Evaluation methods.

Abstract: This paper describes our approach for software architecture quality evaluation. The mechanics of this evaluation is based on quality model, metric model and a set of evaluation methods. These models are considered as a hierarchy properties structured set. The final properties need to be measured using metrics. For this purpose a metric measurement-based framework is linked to the defined quality model. In this evaluation approach an indication of overall quality can be determined using a Fuzzy engine.

1 INTRODUCTION

Several researchers and practitioners are interested in quality emerging issues in the context of software system. Several workshops are intended to provide forums for discussions related to software quality. Quality models have long been introduced in literature (Deutsch and al, 1988), mostly as structured sets of properties (such as reliability, maintainability, and so on etc (Kitchenham et al., 1996). These properties are usually presented as a hierarchy of statements. At various levels, properties are denoted as goals or attributes or characteristics, down to sub-characteristics or factors, to criteria and indicators and attributes again; but the point is not yet set, even in standardization activities which are now covering the field (ISO/IEC, 1988-1991). More action oriented quality ideals and principles for evaluation can be found in Cronholm & Goldkuhl (Goldkuhl, 2002).

Some approaches are focused on a direct evaluation of the quality of a software product, and can be implemented using GQM method (Goal-Question-Metric), described for the first time in (Basili, 1984) and developed since that time by NASA. The set of goals or quality characteristics can be the same or similar to the one defined in ISO/IEC 9126 (ISO/IEC, 2001).

The aim of this study is to present a new approach for direct evaluation of the quality of software architectures. In this evaluation approach criteria interdependences can be managed and an indication of overall quality can be determined using a Fuzzy engine. This paper is organized as follow; the following section addresses the evaluation model which is composed of three parts: the quality model, the metric model and criteria interdependences. Following this, we present our evaluation method which composed of two parts: basic evaluation scenario and optional evaluation scenario. In each part, a set of scenarios is developed. The paper ends with a conclusion and some perspectives.

2 THE EVALUATION MODELS

Quality Model. The quality model can be defined by a set of views concerning the product. Each view is decomposed into several factors. A factor is decomposed into several criteria. The factors are in general external attributes (but also internal attributes: testability, effectiveness, etc). Each criterion is defined by a set of metric. In our approach, the quality model is presented in form of tree structure in which each factor can have one or more criteria. Each criterion can have one or more sub-criteria. The different feature properties that link factors, criteria and sub-criteria are represented using the following symbols: (1) ';' alternative, (2) '$\wedge$' mandatory, (3) '$\vee$' or, (4) '?' optional, (5) '$\varnothing$' empty mode. By considering the example of the Fig. 2 we can give the following expressions:

$Quality\ goal=(FactorA;FactorB)$
$FactorA=(Criterion_4;Criterion_6)$, $Factor_B=(Criterion_5\wedge Criterion_8)$, $Criterion_4=(Criterion_{11}\vee Criterion_{12})$, $Criterion_6=(Criterion_9\wedge Criterion_{10})$, $Criterion_9=(Criterion_{14};Criterion_{13}?)$, $Criterion_8=(\varnothing)$, $Criterion_{10}=(\varnothing)$, $Criterion_{11}=(\varnothing)$,$Criterion_{12}=(\varnothing)$, $Criterion_{13}=(\varnothing)$,$Criterion_{14}=(\varnothing)$, $Criterion_5=(\varnothing)$.

Metric Model. The metric model is a set of metric which is used to quantify an aspect of software architectures. The utility of these metrics is double:

Lamouchi O., Ramdane-Cherif A. and Lévy N. (2009).
SOFTWARE ARCHITECTURE EVALUATION APPROACH .
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 353-356
DOI: 10.5220/0001665503530356

on the one hand, they make it possible to anticipate the needs and to envisage the consequence resources; in addition, they can help the designer or the developer to better understanding the architecture of his system.
The calculation of metrics implies the notion of the metric-variable (Mv). The metric-variable is basic measurement function extracted from software architecture or data collected by designers. The metric-variable is used in the calculation of one or more metrics, and a metric can be used in the calculation of one or more other metrics.
By considering the example (Fig. 2) we can give the following expressions:

Ma={Mv_1; Mv_2}, Mb={ Mv_4}, Mc={Ma; Mv_4}, Md={Me; Mv_2}, Me={Mv_3; Mv_5}, Mf={Me ;Mb}.

Interdependence Model. As we already mentioned, factors and criteria are evaluated by metrics, and each metric is composed by one or more metric-variables (Mv). In consequence, the level of quality depends on the variation of Mv. In our approach, we seek to present the variation of metric-variables that permits the satisfaction of all qualities presented in the quality model. The variation of metric-variables in each metric is represented by the variation-sign (Vs): (+), more the result of the Mv is high more the criterion is satisfied. (-), more the result of the Mv is low more the criterion is satisfied. (*), the result of the Mv is neutral; its variation does not impact the evaluation of the criterion.
We can find certain couples (Criterion, Mv) which are interdependent, as in the case of (Criterion_{10}, $\mathbf{Mv_2}$) and (Criterion_{11}, $\mathbf{Mv_2}$) of the Table 1.

3 THE EVALUATION METHODS

For the basic evaluation scenario, we seek to evaluate the factor "Portability". The evaluation model is:

Portability=(Availability;Co-existence)
Availability =(Total_unavailability^Vital_availability^Full_availability);
Co-existence =(Ø), Vital_availability=(Ø), Full_availability=(Ø).Total_unavailability=(Ø),

VITA={NYSerH; NYVitFH}, FA ={NYSerH; NYFH}, TUA={NYTFH ; NYSerH}, COX={PM},

Full_availability ={FA,(*NYFH ,-), (NYSerH,+)} ;*
Vital_availability ={ViTA, *(NYSerH,+), (NyVitFH,-)};*
Total_unavailability ={ TUA, (NYTFH ,-), (NYSerH,+) ;
Co-existence ={ COX, (PM,+)}.

Table 1: Variations of Mv/ criterion.

	Mv_1	Mv_2	Mv_3	Mv_4	Mv_5
Criterion_5			-		*
Criterion_8				+	
Criterion_{10}		-	*		*
Criterion_{11}	+	+			
Criterion_{12}			+	*	-
Criterion_{13}	+	*		+	

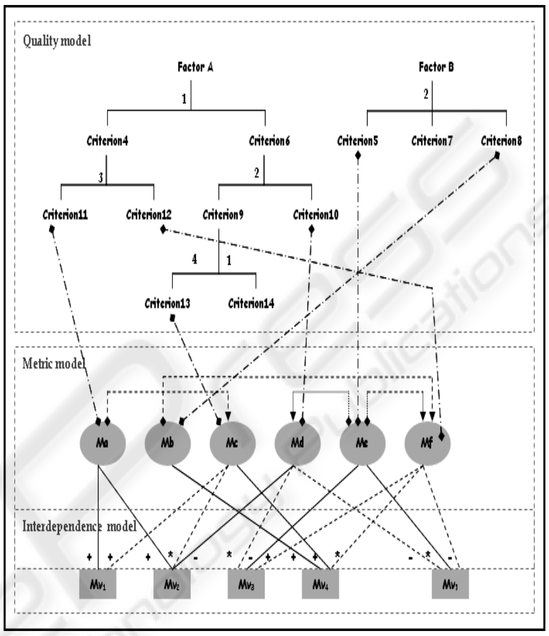

Figure 1: Structure of the evaluation model.

Table 2: Co-existence metric.

Measure of Availability		
Code	Name	Unit
COX	Co-existence	1-yes, 0-not

Table 3: Availability metrics.

Availability Metrics		
Code	Name	Computation formula
FA	Full availability	FA=(NYSerH –NYFH)/ NYSerH
ViTA	Vital availability	ViTA= (NYSerH – NyVitFH) / NYSerH
TUA	Total unavailability	TUA=NYTFH / NYSerH

***NYSerH**: a number of operating hours of the software per year; **NYFH**: a number of hours when at least a function is not available; **NYVitFH**: a number of hours when at least a vital function is not available, **NYTFH**: a number of hours of total stop due to a failure; **PM**: identify the presence of mechanism.*

The basic evaluation scenario is presented by the function EVALUATION_BASE (Algorithm 1).

ALGORITHM EVALUATION_BASE	First iteration	Second iteration	Final iteration
list_ct<-Extract_criterion_leaf (QM)	list_ct=(full_availability, vital_availability, total_unavailability,Co-existence)		
WHILE** NOT EMPTY (list_ct) **DO	yes	yes	yes
ct <-Criterion_folow (liste_ct)	ct = full_availability	ct = co-existence	ct=portability
father_ct<-Father (ct, MQ)	father_ct=availability	father_ct=portability	father_ct=' '
IF** NOT EMPTY (father_ct) **THEN	not	not	yes
under_ct<- Under_criterion (father_ct,MQ)	under_ct = (full_availability, vital_availability, total_unavailability)	under_ct= (co-existence, availability)	-
***IF** Belongs (under_ct,list_ct)*	yes	yes	-
Execution_of_metrics(under_ct)	full_availability =0.8, vital_availability= 0.9 total_unavailability =0.1	co-existence=1, availability=0.797	-
Remove (under_ct,list_ct)	list_ct=(co-existence)	list_ct=()	-
Fuzzy_execution (father_ct, under_ct)	Availability=0.797	portability=0.86	-
Add (father_ct,list_ct)	list_ct= (co-existence, availability)	list_ct=(portability)	-
ELSE	ELSE	ELSE	ELSE
Remove (ct, list_ct)	-	-	-
Add (ct, list_ct)	-	-	-
ENDIF	ENDIF	ENDIF	ENDIF
ENDIF	ENDIF	ENDIF	ENDIF
ENDWHILE	ENDWHILE	ENDWHILE	ENDWHILE
END	END	END	END

Figure 2: Example of basic evaluation scenario ct: criterion; list_ct: list of criteria (fifo); Under_ct: list of criteria; Extract_criterion_leaf(QM): seeks the whole of criteria-leafs of quality model (QM); Criterion_folow(liste_ct): recover a copy of the first criterion being in list_ct; Farther(ct,MQ): seeks the father of criterion ct; Remove(under_ct,list_ct): removes all criteria belonging to under_ct of list_ct; Under_criterion(father_ct,QM): returns the under criteria set of father_ct; Belongs(sous_ct,liste_ct): returns true if all criteria of list under_ct belong to liste_ct; Execution_of_metrics(under_ct): executes metrics of criteria-sheets belong to under_ct; Add(father_ct,list_ct): add father_ct at the end of list_ct. Fuzzy_execution(father_ct,under_ct): launches the evaluation of father_ct using evaluation results of its under criteria.

In many quality standard, the evaluation of their levels of validation by metrics, poses the problem of the definition of thresholds values. 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 objective associated with a criterion will be described like a fuzzy set. For example the criterion "Full_availability" has a very strong quality level if the result of its metric is equal or higher than 0.81, whereas having 0.80 tightened it. This fixed cut does not make it possible to consider a good evaluation. A more realistic approach consists in defining intervals. These intervals make it possible to introduce uncertainty on the thresholds.

We will linguistically express the levels of quality factors, and project them on [0,1]. Thus, fuzzy logic controller makes it possible to express this concept by allotting a degree of truth ranging between 0 and 1. Thus, the "Full_availability" criterion is very strong with a degree of truth of 0.20 and strong with degrees of truth of 0.80 (Figure 5). The rules of fuzzy inferences base, take the form IF [conditions] THEN [actions], where conditions and actions are linguistic labels applied to input and output variables respectively (e.g. IF "Total_unavailability" is Weak AND "Vital_availability" is VStrong AND "Full_availability" is VStrong THEN "Stability" is VStrong). A set of such fuzzy rules constitutes the fuzzy rule –base of the fuzzy logic. The system uses this rule-base to produce precise output values according to actual input values. This control process is divided into three steps:

- Fuzzification: calculate fuzzy input, i.e. evaluate input variables with respect to the corresponding linguistic terms in the condition side (Fig. 3, Fig. 4 & Fig. 5).
- Fuzzy interference: calculate fuzzy output, i.e. evaluate activation strength of every rule and combine their action sides (Fig. 6: A&B).
- Defuzzification: calculate actual output, i.e. convert fuzzy output into a precise numerical value. The result of the treatment is: the quality level of "Stability" is equal to 79,7% (Fig. 6: C).

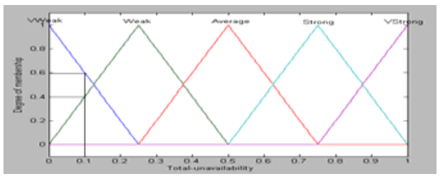

Figure 3: Fuzzification of "Total_unavailability": *Total_unavailability =0,10;VWeak (0,60); Weak (0,40).*

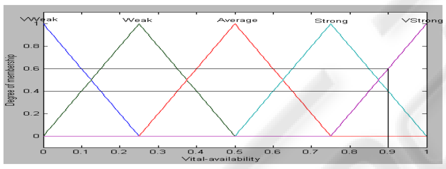

Figure 4: Fuzzification of "Vital _availability": *Vital _availability =0,90;VStrongt (0,60);Strong (0,40).*

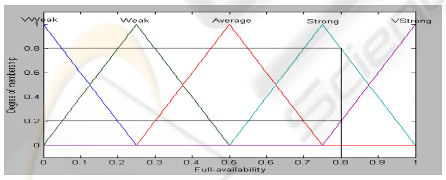

Figure 5: Fuzzification of "Full_availability": *Full_availability =0,80; Strong (0,80); VStrong (0,20).*

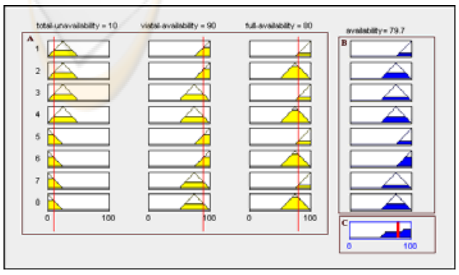

Figure 6: Fuzzy logic controller process.

4 CONCLUSIONS

In this study we have presented a framework for understanding architecture software evaluation. The mechanics of the evaluation is based on quality model. This quality model comes out as a collection of desired properties which can be divided into sub properties at various levels. The last level is linked to various software metrics and measurement techniques that an organisation uses. This hierarchical model appears in more deductive way than those presented in literature. In this evaluation approach interdependences can be managed and an indication of overall quality can be determined. In our work, the objective associated with a criterion will be described like a fuzzy set. The use of a fuzzy threshold permits a more realistic approach. A fuzzy interpreter is used in our basic evaluation scenario and optional evaluation scenario.

REFERENCES

Deutsch, M. S., Willis, R. R, 1988. Software Quality Engineering, Randall W. Jensen.

Kitchenham, B. and. Plfleeger, S. L., 1996. Software Quality. The Elusive Target, IEEE Software, pp. 12-21.

ISO/IEC 9126: (IS), (1988, 1991). Information technology - Software product evaluation - Quality characteristics and guidelines for their use.

ISO/IEC 9126-1,9001, 2001. Quality management systems–Requirements. ISO. Software engineering – Product quality. ISO/IEC.

Basili, V. R., Weiss, D. M., 1984.A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering.*

Cronholm, S. & Goldkuhl, G., 2002. Actable Information Systems - Quality Ideals Put Into Practice. Presented at the Eleventh Conference On Information Systems (ISD 2002). 12-14.

Cox, 1997. La logique floue pour les affaires et l'industrie. International Thomson Publishing France, Paris, France.

Ramdane-cherif A., Lamouchi, O., Lévy, N., 2007. One quality software evaluation approach. CAINE 2007, ISCA 20th international Conference on Computer Application in Industry and Engineering. San Francisco, California USA.

Akoka J. Wattiau I, 2002. La Qualité du logiciel.

Losavio, F., Chirinos, L., Matteo, A., Lévy, N., Ramdane-Cherif A , 2004. ISO quality standards for measuring architectures. The journal of Systems and Software 72, 209-223.

Mamdani, E., Assilian, S., 1975. An experiment in linguistic synthesis with a fuzzy logic controller. International Journal of Man-Machine Studies. vol. 7, pp. 1–13.