SOFTWARE ARCHITECTURE EVALUATION APPROACH

Olfa Lamouchi\textsuperscript{1}, Amar Ramdane-Cherif\textsuperscript{1,2} and Nicole Lévy\textsuperscript{1}

\textsuperscript{1}PRISM, \textsuperscript{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. 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) '\emptyset' empty mode. By considering the example of the Fig. 2 we can give the following expressions:

Quality goal=(FactorA;FactorB)
FactorA=(Criterion4;Criterion6), FactorB=(Criterion5\wedgeCriterion8), Criterion4=(Criterion11\veeCriterion12), Criterion6=(Criterion9\wedgeCriterion10), Criterion9=(Criterion14;Criterion13?), Criterion8=(\emptyset), Criterion10=(\emptyset), Criterion11=(\emptyset), Criterion12=(\emptyset), Criterion13=(\emptyset), Criterion14=(\emptyset), Criterion5=(\emptyset).

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:
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:

\[ M_a = \{M_v_1; M_v_2\}, M_b = \{M_v_3\}, M_c = \{M_v_3; M_v_4\}, M_d = \{M_v_3; M_v_5\}, M_f = \{M_v_3; M_v_4; M_v_5\}. \]

**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, Mv\_2) and (Criterion 11, 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:

\[ \text{Portability}= (\text{Availability}; \text{Co-existence}) \]
\[ \text{Availability}= (\text{Total}_\text{unavailability} \times \text{Vital}_\text{availability} \times \text{Full}_\text{availability}) \]
\[ \text{Co-existence}= (\text{COX}), \text{Vital}_\text{availability}= (\text{VTA}), \text{Full}_\text{availability}= (\text{TUA}), \text{Total}_\text{unavailability}= (\text{TUA}) \]

\[ \text{VTA}= (\text{NYSerH}; \text{NYVitFH}), \text{FA}= (\text{NYSerH}; \text{NYFH}), \text{VTA}= (\text{NYSerH}; \text{NYVitFH}), \text{COX}= (\text{PM}) \]

\[ \text{Availability}= (\text{VTA} \times \text{NYSerH} \times \text{NYVitFH}) \]

\[ \text{Co-existence}= (\text{COX}, \text{PM}) \]

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

The basic evaluation scenario is presented by the function EVALUATION\_BASE (Algorithm 1).

---

**Table 1: Variations of Mv/ criterion.**

<table>
<thead>
<tr>
<th>Mv_1</th>
<th>Mv_2</th>
<th>Mv_3</th>
<th>Mv_4</th>
<th>Mv_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>+</td>
<td>*</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: Co-existence metric.**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>COX</td>
<td>Co-existence</td>
<td>1-yes, 0-not</td>
</tr>
</tbody>
</table>

**Table 3: Availability metrics.**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Computation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA</td>
<td>Full availability</td>
<td>FA=(NYSerH–NYFH)/NYSerH</td>
</tr>
<tr>
<td>VTA</td>
<td>Vital availability</td>
<td>VTA=(NYSerH–NYVitFH)/NYSerH</td>
</tr>
<tr>
<td>TUA</td>
<td>Total unavailability</td>
<td>TUA=NYTFH/NYSerH</td>
</tr>
</tbody>
</table>

\[ \text{NYSerH}: \text{a number of operating hours of the software per year}; \]
\[ \text{NYFH}: \text{a number of hours when at least a function is not available}; \]
\[ \text{NYVitFH}: \text{a number of hours when at least a vital function is not available}; \]
\[ \text{NYTFH}: \text{a number of hours of total stop due to a failure}; \]
\[ \text{PM}: \text{identify the presence of mechanism}. \]
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).

### 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


