A Fuzzy Logic Based Set of Measures for Software Project Similarity: Validation and Possible Improvements

Ali I DRI, ENSIAS, Rabat, Morocco
Alain ABRAN, UQAM, Montreal, Canada

7th IEEE International Software Metrics Symposium, London, 4-6 April, 2001
Plan

- Introduction
- Fuzzy Logic
- Software Project Similarity Measures
- Validation
- Possible Improvements
- Conclusions and future work
Introduction

- Software Project similarity is one of the most important process attribute
- It is often used when estimating software development effort by analogy
- Intuitively, two software projects are not similar if the differences between their sets of attributes are obvious
- Analogy

<table>
<thead>
<tr>
<th>Human</th>
<th>Software Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Size</td>
</tr>
<tr>
<td>Color of skin</td>
<td>Complexity</td>
</tr>
<tr>
<td>Color of eyes</td>
<td>Reliability</td>
</tr>
<tr>
<td>Color of hair</td>
<td>Analyst competence</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Sheppered et al. (1997)

$$d(P_1, P_2, V) = \frac{1}{\sum_{v_j} d_{v_j}(P_1, P_2)}$$

$$d_{v_j}(P_1, P_2) = \begin{cases} 
(v_j(P_1) - v_j(P_2))^2 & \text{if } v_j(P_1) = v_j(P_2) \\
1 & \text{if } v_j(P_1) \neq v_j(P_2) 
\end{cases}$$

Critics

- Euclidean distance is used when the attributes are measured in at least an interval scale.
- Most of the software attributes are measured in an ordinal or nominal scales (COCOMO’81, COCOMOII, Function Points, ...).
- The equality distance is used when the values are classical intervals rather than fuzzy sets.
- The equality distance is not precise and can give great difference when estimating effort for two similar projects (Idri, Abran. 7th FT&T, Atlantic City, 2000).
Objective

- To measure the similarity between software projects when they are described by linguistic values such as ‘very low’, ‘high’, ‘complex’, ‘excellent’...

- Software projects are described by linguistic values => software projects are described by vagueness informations

- This is always the case in software measurement for an ordinal or nominal scale! But it can be presented in ratio or interval scale (fuzzy numerical number).

Our comprehension of the software is still limited
Fuzzy Logic

- Values between ‘TRUE’ and ‘FALSE’?

‘The main motivation of fuzzy logic is the desire to build up a formal, quantitative framework that captures the vagueness of human knowledge via natural languages’ Dubois and Prade 1991

- 1965, Zadeh : Fuzzy Set

- 1994, Zadeh : Fuzzy Logic = Fuzzy Set Theory

- Fuzzy Set

  - A person X is ‘young’?
  - Among 100 answers:
    - 50 ==> [21, 30]
    - 30 ==> [25, 30]
    - 20 ==> [22, 35]

  I strongly disagree!!
Operations on fuzzy sets

Intersection
- $\mu_{A \cap B}(x) = \min (\mu_A(x), \mu_B(x))$

Union
- $\mu_{A \cup B}(x) = \max (\mu_A(x), \mu_B(x))$

Complement
- $\mu_{\neg A}(x) = 1 - \mu_A(x)$

Fuzzy Rules, Fuzzy Relations, Fuzzy Reasoning, Fuzzy Control, ...
Fuzzy Logic Based Measures for Software Project Similarity

Objective:
- Evaluating similarity between two software projects $P_1$ and $P_2$ described by M linguistic variables:

$$d(P_1, P_2) = ?$$

Hypothesis:
- $P_1$ and $P_2$ are described by M linguistic variables
- Each linguistic variable, $V_j$, is measured by linguistic values, $A_k^j$
- Each linguistic value is represented by a fuzzy set with a membership function $\mu_{V_j}^{A_k}$

How?
- The computing process is organized in two steps:
First Step:

- $d_{vj}(P_1, P_2)$ must express the fuzzy equality according to $V_j$ of $P_1$ and $P_2$

- The fuzzy equality can be represented by the following fuzzy relation:

  $R^{vj}_{\approx} = P_1$ and $P_2$ are approximately equal according to $V_j$

- Problem:

  $\hat{r}_{\approx}^{vj}(P_1, P_2) = ?$
\( \mathbf{R}^v_j = \text{Combination} \quad \mathbf{R}^v_j \)

\( \mathbf{R}^v_j \) is the fuzzy if-then rule:

**if** \( v_j(P_1) \) is \( A_k \) **then** \( v_j(P_2) \) is \( A_k \)

- Combination of \( \mathbf{R}^v_j \) is called **aggregation**
- The way this is done is different for the various types of **fuzzy implication functions** adopted for the fuzzy rules \( \mathbf{R}^v_j \)

\[
\mathbf{R}^v_j = \bigcup \mathbf{R}^v_j = \bigcup (A_{k}^j \cap A_{k}^j) \quad (A \rightarrow B \text{ is } A \cap B)
\]

\[
\mathbf{R}^v_j = \cap \mathbf{R}^v_j = \cap (\neg A_{k}^j \cup A_{k}^j) \quad (A \rightarrow B \text{ is } \neg A \cup B)
\]
\[ \mu_{R_{\alpha}^{v_j}}(P_1, P_2) = \begin{cases} 
\max \min_{k} (\mu_{A_k}^{v_j}(P_1), \mu_{A_k}^{v_j}(P_2)) \\
\max - \min \text{ aggregation n} \\
\text{ou} \\
\sum_{k} \mu_{A_k}^{v_j}(P_1) \times \mu_{A_k}^{v_j}(P_2) \\
\text{sum - product aggregation n} \\
\end{cases} \]

\[ \mu_{R_{\alpha}^{v_j}}(P_1, P_2) = \begin{cases} 
\min \max_{k} (1 - \mu_{A_k}^{v_j}(P_1), \mu_{A_k}^{v_j}(P_2)) \\
\min - \text{Kleene Dienes aggregation n} \\
\end{cases} \]
**Second Step:**

- $d(P_1, P_2)$ is calculated from the various individual distances $d_{v_j}(P_1, P_2)$

$$d(P_1, P_2) = F(d_{v_1}(P_1, P_2), ..., d_{v_M}(P_1, P_2))$$

- The function $F$ is one of the three operators: **min, max, i-or**

$$d(P_1, P_2) = \begin{cases} 
\min(d_{v_1}(P_1, P_2), ..., d_{v_M}(P_1, P_2)) \\
\max(d_{v_1}(P_1, P_2), ..., d_{v_M}(P_1, P_2)) \\
i - or(d_{v_1}(P_1, P_2), ..., d_{v_M}(P_1, P_2)) = \begin{cases} 
0 & \exists k, h/d_{v_k}(P_1, P_2) = 1 \text{ and } d_{v_h}(P_1, P_2) \\
\prod_{j=1}^{M} d_{v_j}(P_1, P_2) & \text{otherwise} \\
\prod_{j=1}^{M} (1 - d_{v_j}(P_1, P_2)) + \prod_{j=1}^{M} d_{v_j}(P_1, P_2) 
\end{cases} 
\end{cases}$$
Software Measurement Validation

- Software measurement validation is an important step in software metrics building process
- It allows us to choose the best measures from a large number of software measures for a given attribute
- No common definition: What is a valid measure?
- Fenton’s approach (1997):
  - Validation in the narrow sense
    « The measure is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied »
  - Validation in the wide sense
    « The measure is valid in the narrow sense and it is a component of a valid prediction system »
Axiomatic Validation

- Similarity measures satisfy the representation condition if they do not contradict any intuitive notions about the similarity of $P_1$ and $P_2$

- Our initial understanding will be codified by four axioms.

- We check whether or not the two measures satisfy these axioms.

- A tuple of fuzzy sets $(A_1, A_2, ..., A_n)$ satisfy the **normal condition** if $(A_1, A_2, ..., A_n)$ is a fuzzy partition and each $A_i$ is **normal** and **convex**
Axiom 0

\[ d_{v_j}(P_1, P_2) \neq 0 \iff \exists A_k / \mu_{A_k}^{V_j}(P_1) \neq 0 \text{ and } \mu_{A_k}^{V_j}(P_2) \neq 0 \]

Fuzzy Sets for \( V_j \)

Axiom 1

\[ d(P_1, P_2) \geq 0; \ d(P, P) > 0 \]
Axiom 2

\[ d(P, P_i) \leq d(P, P) \]

Axiom 3

\[ d(P_1, P_2) = d(P_2, P_1) \]

Results of the axiomatic validation

<table>
<thead>
<tr>
<th></th>
<th>Max-min</th>
<th>Sum-product</th>
<th>Kleene-Dienes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Axiom 0</strong></td>
<td>Yes/</td>
<td>Yes/</td>
<td>No/</td>
</tr>
<tr>
<td><strong>Axiom 1</strong></td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
</tr>
<tr>
<td><strong>Axiom 2</strong></td>
<td>Yes/Yes</td>
<td>No/No</td>
<td>Yes/Yes if NC</td>
</tr>
<tr>
<td><strong>Axiom 3</strong></td>
<td>Yes/Yes</td>
<td>Yes/Yes</td>
<td>No/No</td>
</tr>
</tbody>
</table>
Towards an empirical validation

- The measures will be valid in the wide sense if they are both valid in the narrow sense and a component of a valid prediction system.

- The prediction system adopted is the estimation of software development effort by analogy.

- Estimation by analogy is composed by:
  - Characterization of the projects by a set of attributes such as Reliability, Complexity, Analysts competence ...
  - Evaluation of the similarity between the candidate project and each project in the database
  - Adaptation
Intermediate COCOMO’81 database is used as an historical data

- Each project is described by 17 attributes:
  - Software size measured in KDSI
  - Project Mode is defined as Organic, semi-detached or embedded
  - 15 cost drivers related to the software environment

- Each cost driver is measured using rating scale of six linguistic values:
  Vey low, Low, Nominal, High, Very high, Extra-high

- The assignment of linguistic values uses the conventional quantization where the values are classical intervals
Example: DATA cost driver

\[
\frac{D}{P} = \frac{\text{Database size in bytes or characters}}{\text{Program size in DSI}}
\]

<table>
<thead>
<tr>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very high</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/P&lt;10</td>
<td>10\leq D/P&lt;100</td>
<td>100\leq D/P&lt;1000</td>
<td>D/P\geq1000</td>
</tr>
</tbody>
</table>

- It is more general
- It mimics the way in which humans interpret linguistic values
- The transition from one linguistic value to a contiguous linguistic value is **gradual** rather than **abrupt**
### Results for max-min and sum-product aggregations

<table>
<thead>
<tr>
<th></th>
<th>( d_{ij}(P_m, P_n) )</th>
<th>( d(P_m, P_n) )</th>
<th>( i\lor )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min</strong></td>
<td>( P_1 )</td>
<td>( P_2 )</td>
<td>( P_3 )</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>.521</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>0</td>
<td>.742</td>
<td>0</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>0</td>
<td>0</td>
<td>.659</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( d_{ij}(P_m, P_n) )</th>
<th>( d(P_m, P_n) )</th>
<th>( i\lor )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min</strong></td>
<td>( P_1 )</td>
<td>( P_2 )</td>
<td>( P_3 )</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>.501</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>0</td>
<td>.660</td>
<td>0</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>0</td>
<td>0</td>
<td>.550</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Min operator

It expresses the 'and' logical operation

\[
d(P_m, P_n) = 0 \iff \exists v_j / d_{v_j}(P_1, P_2) = 0
\]

\( d(P_m, P_n) = 0 \) if \( n \neq m \)

Explication: often in the COCOMO’81 database, two projects have at least one variable for which the associated linguistic values are different

\( d(P_m, P_m) \neq 1!! \)

If Normal Condition \( d(P_m, P_m) \geq 0.5 \)

\( d(P_m, P_n) \) using max-min aggregation is different of \( d(P_m, P_n) \) using sum-product aggregation

\[
\left| d(P_m, P_n)_{\text{max-min}} - d(P_m, P_n)_{\text{sum-product}} \right| \leq \frac{1}{8}
\]
Max operator

- It expresses the ‘or’ logical operation

\[
d(P_m, P_n) \neq 0 \Leftrightarrow \exists v_j / \ d_{v_j}(P_1, P_2) \neq 0
\]

- \(d(P_m, P_n) \neq 0\) if \(n \neq m\)

**Explication**: often in the COCOMO’81 database, two projects have at least one variable for which the associated linguistic values are the same

- \(d(P_m, P_m) = 1\)

- \(d(P_m, P_n)\) using max-min aggregation is different of \(d(P_m, P_n)\) using sum-product aggregation

\[
\left| d(P_m, P_n)_{\text{max-min}} - d(P_m, P_n)_{\text{sum-product}} \right| \leq \frac{1}{8}
\]
I-or operator

Between the ‘and’ and the ‘or’ logical operations

\[ d(P_m, P_n) = 0 \iff \exists v_j / \quad d_{v_j}(P_1, P_2) = 0 \]

\[ d(P_m, P_n) = 1 \iff \begin{cases} 
\exists v_j / \quad d_{v_j}(P_1, P_2) = 1 \\
\forall v_j / \quad d_{v_j}(P_1, P_2) \neq 0
\end{cases} \]

- \( d(P_m, P_n) = 0 \) if \( n \neq m \)

*Explication*: Like the case of min operator

- \( d(P_m, P_m) = 1 \)

- \( d(P_m, P_n) \) using max-min aggregation is different of \( d(P_m, P_n) \) using sum-product aggregation

This difference is not obvious because

- The \( \left| d(P_m, P_n)_{\text{max-min}} - d(P_m, P_n)_{\text{sum-product}} \right| \leq \frac{1}{8} \)

- The i-or function is continuous
Human discourse uses a large number of linguistic quantifiers.

Zadeh distinguishes between two classes:
- Absolute linguistic quantifiers
- Proportional linguistic quantifiers (most, few, at least, at most, ...)

Yager has distinguished three categories of proportional quantifiers:
- RIM quantifiers (most, at least $\alpha$, ...)
- RDM quantifiers (few, at most $\alpha$, ...)
- RUN quantifiers (about $\alpha$)
In this work, we have used only two RIM quantifiers ‘all’ and ‘there exist’ to combine the individual distances $d_{ij}(P_m, P_n)$.

Critics:

- The ‘all’ and the ‘there exist’ quantifiers are not always a good combination.
- In many situations, other linguistic quantifiers can be useful such that ‘most’, ‘many’, and ‘at least $\alpha$’.
- We must take into account the importance of the variables describing the projects.
- The i-or operator has no clear natural interpretation.
Solution

Evaluation of the $d(P_m, P_n)$ by aggregating the individual distances using RIM linguistic quantifiers


\[
d(P_m, P_n) = \begin{cases} 
\text{most of } d_{v_j}(P_m, P_n) \\
\text{many of } d_{v_j}(P_m, P_n) \\
\text{at least four of } d_{v_j}(P_m, P_n) \\
. 
\end{cases}
\]
Conclusions and Future work

- We have developed and validated a set of similarity measures. These measures are also applicable when the variables are numeric.

- From an axiomatic validation, we have retained two measures for the individual similarities:
  - Are the four axioms represent an exhaustive list of all required properties?
  - What about the transitivity of \( d(P_1, P_2) \)?

- The empirical validation of estimation effort by analogy must be achieved:
  - For the individual distance, we use the two retained measures
  - For the overall distance, we use RIM linguistic quantifiers
Can I use our measures for prediction of Size, Reliability, Maintenability,…?

Fuzzification of the software measurement theory

Building prediction systems that satisfy Soft Computing:
- Tolerance of imprecision (Fuzzy Logic)
- Learning (Neural Networks)
- Uncertainty (Belief networks, genetic algorithms,…)
Welcome to Rabat, Morocco

The royal city with a prestigious past