Full citation: MacDonell, S.G., Gray, A.R., & Calvert, J.M. (1999) FULSOME: a fuzzy logic modeling tool for software metricians, in Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS'99). New York, IEEE Computer Society Press, pp.263-267. http://dx.doi.org/10.1109/NAFIPS.1999.781695

FULSOME: A Fuzzy Logic Modeling Tool for Software Metricians

Stephen G. MacDonell, Andrew R. Gray, and James M. Calvert
Department of Information Science
University of Otago, PO Box 56, Dunedin, New Zealand
{stevemac} {agray} {jmcalvert}@infoscience.otago.ac.nz

Abstract

There has been a growing body of literature suggesting that some of the problems faced by software development project managers can be at least partially overcome by using fuzzy logic techniques. However, one issue that has been generally overlooked in this recommendation is the means by which these "software metricians" can collect data for, develop, and interpret fuzzy logic models in practice. We describe a freely available system that has been built with this in mind called FULSOME (FUzzy Logic for SOftware MEtrics). While there are many tools available for developing fuzzy models, it is suggested that before there will be real adoption of such techniques by project managers there will need to be suitable tools that support their particular work-flows and that use appropriate terminology. Another requirement will be the development of some standard procedures and definitions for such models. Issues involved with membership function elicitation and extraction are also discussed as a first step towards this second goal.

1. SOFTWARE METRIC MODELS

Software metrics is the field of research and practice that involves investigating the characteristics of and relationships between sets of attributes associated with software development projects, usually in terms of products, processes, and resources [1]. Such analyses can be exploratory or model building in nature and applications can be classified as predicting, monitoring, controlling, or assessing some aspect of development. Here the focus is on predictive models since these form the majority of applications and support the other three varieties of analyses.

By far, the most common application for software metric models has been predicting development effort based on system characteristics from the development specifications (available relatively early in the project life-cycle). The emphasis that has been placed on this particular application originates from the considerable financial and strategic implications of accurately timing and costing software development projects. For example, a common scenario would be to predict development effort (perhaps in person-months) based on measures of system size, system complexity, and developer competence. Such models are most useful in the early stages of development where the uncertainty is at its greatest and such estimates are essential for contract negotiation and strategic planning.

1.1 Difficulties with current approaches

In the past several problems have been almost ubiquitous for "software metricians" and these have proved to be considerable impediments to both the adoption of techniques by practitioners and the advancement of researchers' theoretical understanding. Primary difficulties experienced with developing and calibrating models for development effort prediction include data acquisition, data purity, model expression, and knowledge gathering issues.

The first of these, acquiring data, is especially problematic since the most commonly used formal techniques for modeling development effort are regression based (generally linear least-squares). The process of capturing meaningful data is made more difficult by both rapidly changing technologies (rendering data sets quickly outdated) and the reluctance of organizations to share their data. The problem is further compounded by the influence of developers disliking such close monitoring of their performance.

Even once sufficient quantities of data are available, data purity is generally difficult to ascertain – necessitating some treatment of unusual observations [4]. Such points

may originate from unusual systems, unexpected developer performance, or recording errors. Alternatively, such cases may be due to factors that are, for all intents and purposes, unrecordable, for example personality characteristics of team members. The use of expert knowledge based estimates has dominated all formal techniques in software metric modeling, with the considerable influence of human factors perhaps necessitating some form of expert intervention in any model development or deployment process.

Since software metric models are generally expressed as simple linear equations, the inherent nonlinearities and interactions intrinsic to the development process are disregarded, leading to inferior models in cases where the true underlying relationship is significantly more complex than this simplified view of reality. Some basic transformations are used on occasions to remedy this, but there has been little use of more general nonlinear approaches, aside from some elementary use of neural network models [3].

Finally, process maturation within an organization can be impeded by the inability of management to extract a sufficiently meaningful understanding of the underlying mechanisms behind the development process from such simple, and often ineffectual, models. As development projects become increasingly large, such "data mining" applications will become ever more imperative in order to optimize development practices.

1.2 Fuzzy logic for software metric models

Fuzzy logic has recently gained a greater amount of attention in the software metrics literature as a means for solving some of these long-standing problems [2, 3]. Obviously fuzzy logic is not a panacea for all of project management's ills. In fact it seems quite likely that the improvements from better modeling would be less than those from better management and coding practices. However, the use of fuzzy logic techniques to support other existing techniques (allowing each to operate within its own niche) does appear to offer some potential benefits that are worth pursuing.

The data acquisition problem is considerably alleviated under conditions where expert knowledge can be used to augment or even supplant empirical data. Similarly, the lessened reliance on data for the initial specification of the model can be seen as a more *robust* approach in terms of treatment for potentially outlying observations.

Since fuzzy logic models can be specified in a piecewise manner, the restrictions of linearity are easily overcome. In the same way, interactions are implicitly included in multiple antecedent models.

Finally, the transparency of fuzzy logic models provides both a check on the model's reasonableness (even where data-driven calibration techniques are used) and can assist organizational learning via process awareness. Discussions with several New Zealand software houses have verified that this is a useful outcome in itself.

2. FULSOME

FULSOME (FUzzy Logic for SOftware MEtrics) is a Microsoft Windows application (although the code libraries are written in standard C++ and should be easily ported to any operating system) that provides project managers with access to fuzzy logic modeling techniques in a manner that reflects their particular needs and workflows.

Figure 1 shows the basic structure of the FULSOME program – "systems" containing data sets, membership functions, rules, and the output of the inference process. The figure also illustrates the basic functionality of the system, and the approach that was taken to make the system as user-friendly as possible.

Two additional modules are also available, for fuzzy clustering and model evaluation, although they are not technically part of the FULSOME core system and are correspondingly absent from Figure 1. The fuzzy clustering module allows for the automatic extraction of membership functions from data sets (using one-dimensional fuzzy c-means clustering), and the extraction of rules based on a set of membership functions (using multi-dimensional fuzzy c-means clustering). The model evaluation module provides several measures of goodness of fit for a pre-specified model on different data sets.

Since one of the applications discussed here is the extraction of membership functions some further detail is provided on the first of these two supplementary modules. The basic algorithm for membership function extraction is as follows.

1. select an appropriate mathematically defined function for the membership functions of the variable of interest (i) say $f_i(x)$

2. select the number of membership functions that are desired for that particular variable, m_i functions for variable i.

3. call each of the m_i functions $f_{ij}([x])$ where $j=1...\ m_i$ and $[x]$ is an array of parameters defining that particular function (usually a center and width parameter are defined, either explicitly or implicitly)

4. using one-dimensional fuzzy c-means clustering on the data set find the m_i cluster centers, c_{ij} from the available data

5. sort the cluster centers c_{ij} into monotonic (generally ascending) order for the given i

6. set the membership function center for $f_{ij,}$ generally represented as one of the parameters in the array [x], to the cluster center c_{ij}

7. set the membership function widths for f_{ij} in [x] such that, $\sum_{n=1}^{m_i} f_{in}([c_{in},\ldots]) = 1$ or as close as possible for the chosen $f(x)$ where this cannot be achieved exactly (for example for triangular membership functions each function can be defined using three points, a, b and c where a is the center of the next smaller functions and c is the center of the next larger function)

Similarly, rules can be extracted from data by using the same process of clustering with multiple dimensions (the number of dimensions matching the number of antecedents plus the single consequent).

1. start with known membership functions f_{ij} ([x]) for all variables, both input and output, where j represents the number of functions for variable i and [x] is the set of parameters for the particular family of function curves

2. select the number of clusters k (which represents the number of rules involving the k – 1 independent variables to estimate the single output variable)

3. perform fuzzy c-means clustering to find the centers (i dimensional) for each of the k clusters

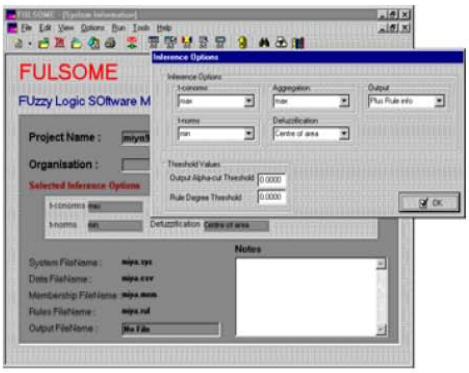
4. for each cluster k with center c_k

(a) determine the k^{th} rule to have the antecedents and consequent f_{ij} for each variable i where f_{ij} (c_k) is maximized over all j.

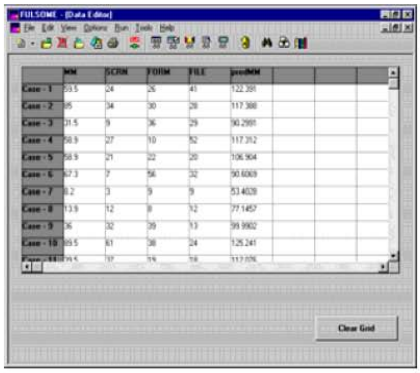
(b) weight the rule, possibly as $\prod_{n=1}^{i} f_{ij}\ (c_k)$ or $\sum_{n=1}^{i} f_{ij}(c_k)$

5. combine rules with same antecedents and consequents, either summing, multiplying, or bounded summing rule weights together

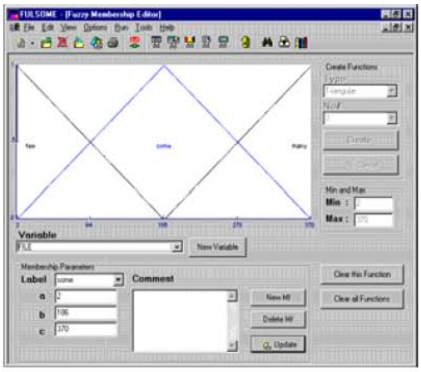
6. (optionally) ratio scale all weights so that the mean weight is equal to 1.0 to aid interpretability



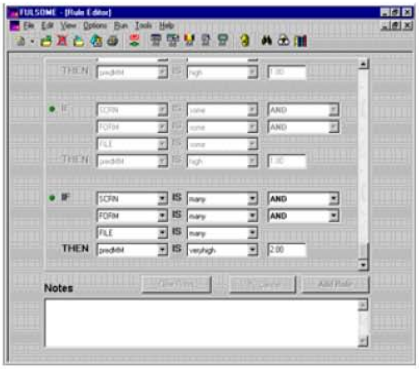
(a) System screen



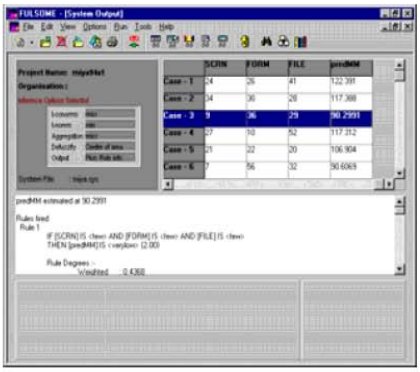
(b) Data editor



(c) Membership function editor



(d) Rule editor



(e) Output screen

Figure 1. Structure of the FULSOME system – reading from left to right, top to bottom reflects the usual sequence of work

3. MEMBERSHIP FUNCTIONS

Three means for deriving membership functions have been examined for suitability to the metrics application domain, namely derivation through polling of classifications, combining expert drawn membership functions, and automated data-driven methods. The most useful of these approaches for general software metric models, in our experience, has been that of polling where a sufficient number of experts are available. Since fuzzy logic models are intended to be used as a communication vehicle it is important that they reflect, as well as possible, the consensus view of stakeholders.

Two separate surveys of New Zealand software development project managers were used for this approach, with surprisingly different results. The differences observed suggest that the two seemingly similar approaches are in fact quite different in terms of the responses generated.

In the first survey, the respondents were asked to provide ranges of values that they felt best described fuzzy labels (where the ranges were non-overlapping). For example, a manager may have replied that *small* ranged between 0 and 30 entities, *medium* between 30 and 60 entities, and *large* was 60 or more entities. The membership functions were then plotted using the proportion of respondents who ascribed each label to equally spaced intervals throughout the ranges.

In the second survey, the respondents were provided with a list of values for each variable and asked to classify these as best belonging to one of several labels. The functions were again derived by using the proportion of respondents classifying each value as belonging to a particular function, although this time the points evaluated were pre-specified.

The first survey involved 36 respondents, and the second produced 34 responses. Both surveys were the result of mailings to major New Zealand software development organizations.

In the two surveys the results for the data model size questions are shown in Figures 2 and 3. The smoothed functions are produced using Bezier curves, since this removes (in most cases) the incongruity of "bumpy" membership functions with several maxima and minima. As can be seen the second approach has led to much *cleaner* looking functions, with significantly less ambiguity regarding the middle function. Similar, and in some cases much more extreme, results emerged with the graphs of other variables.

It could be suggested from this result that providing (and thereby limiting) survey respondents' freedom has produced better results. Allowing the managers to specify their own ranges has seemingly led to lower cut-off points (the two sets of membership functions are plotted on the same x-range for comparability). Alternatively, it could be said that using the predefined list of values has led managers into delimiting the range into the number of membership functions. However, we believe that the former is a much more likely proposition here. In order to determine which explanation best reflects the results, a follow-up survey will be used where managers will be presented with a survey in the manner that they were not originally exposed to.

4. PROCESS FOR DEVELOPING FUZZY LOGIC MODELS FOR SOFTWARE METRIC PREDICTION

As part of encouraging fuzzy logic models for software project management it is important to provide some form of standard. The procedure that we recommend for developing fuzzy logic models for predictive software metric applications consists of the following (not necessary mandatory nor sequential) steps. This is only a skeleton of the actual guideline document that we are providing to commercial organizations, however it illustrates the basic procedures.

1. Define the goal of the metrics (sub)program.

2. Define the dependent variable (including scale).

3. Using some process, such as a standard model or GQM (or one of its variants) determine the *n* independent variables and similarly define them in an unambiguous manner (including scale).

4. Obtain membership functions for all variables

- If a sufficiently large number of experts are available then use polling techniques to determine membership functions for all variables *after* providing a fuzzy logic tutorial if necessary.

- Otherwise, if a small number of sufficiently qualified experts are available for drawing membership functions then have them each draw membership functions and combine these.

- Otherwise, using data automatically extract cluster centers for standard shaped membership functions.

- Otherwise, use a standard defined set of membership functions (provided).

5. Validate the set of membership functions using a measure of inter-reliability. If the results are too low then re-negotiate the function(s) in question.

6. Obtain rules

- If a sufficiently large number of experts are available then use voting techniques to determine rules for all combinations of independent variables *after* providing a fuzzy logic tutorial if necessary.

- Otherwise, if a small number of sufficiently qualified experts are available for deriving rules, then have them each provide rules and use Delphi methods to arrive at a final set of rules.

- Otherwise, using data automatically extract rules for the currently defined membership functions.

- Otherwise, use a standard defined set of rules (provided).

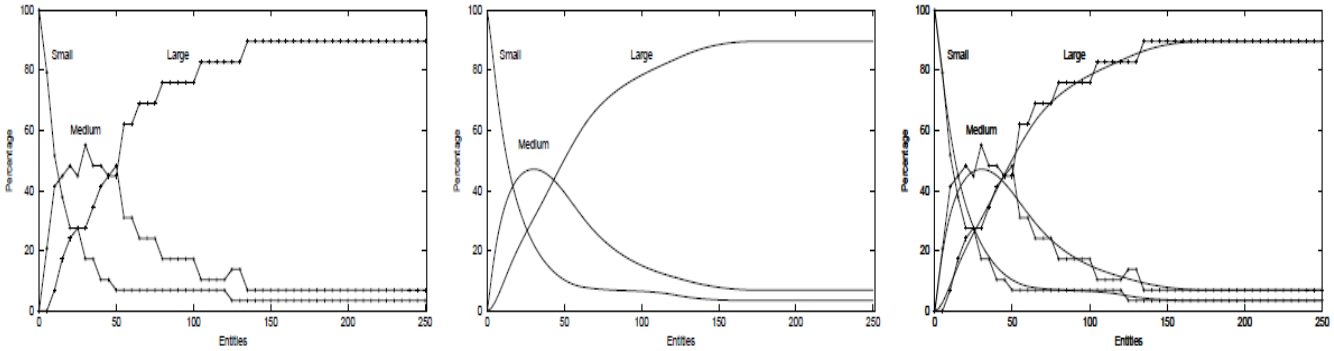
7. Validate the set of rules using a measure of inter-reliability. If the results are too low then re-negotiate the rules(s) in question.

8. Assess model performance on any data used for membership function and/or rule derivation above. This provides some measure of the ability of the model to fit to the data, but should not be seen as indicative of generalization performance.

9. If additional holdout data is available, then use this to assess the model's ability to generalize.

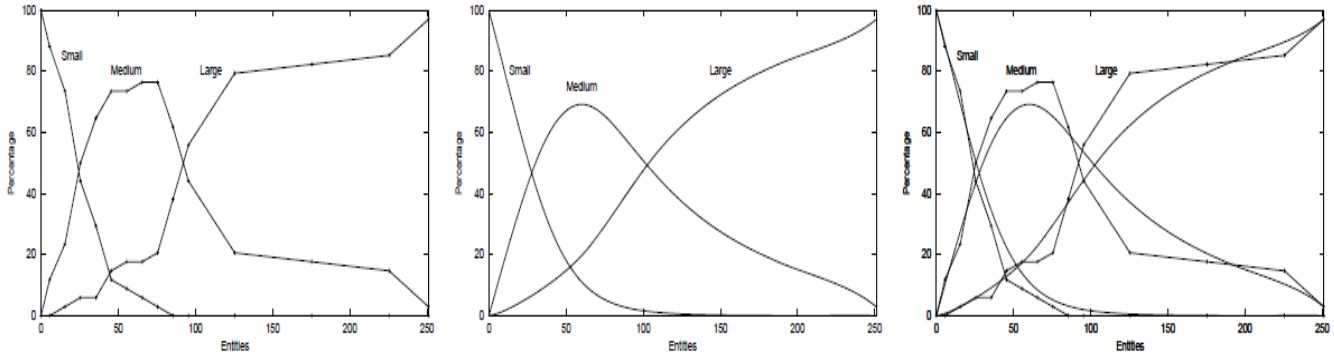
10. If model's performance is acceptable then implement model.

11. As managerial experience and empirical data allow revise functions and rules.



(a) Entities – original (b) Entities – smoothed (c) Entities – combined

Figure 2. Graphs of "entities" size membership functions derived from the first survey



(a) Entities – original (b) Entities – smoothed (c) Entities – combined

Figure 3. Graphs of "entities" size membership functions derived from the second survey

5. CONCLUSIONS

The use of fuzzy logic models appears to be a promising addition to the current suite of methods used for the empirical support of software development processes. In particular, the ability of this technique to operate without large quantities of data and to provide a basis for the acquisition of further organizational process knowledge is especially appealing.

By providing a simple set of procedures and supporting software we have been able to attract the interest of several commercial organizations, leading to our next phase of implementing the techniques in *live* projects (as opposed to completed projects or where the use of fuzzy logic was limited to one phase of development).

REFERENCES

[1] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous & Practical Approach*. PWS, 1997.

[2] A. Gray and S. MacDonell. Applications of fuzzy logic to software metric models for development effort estimation. In *Proceedings of the 1997 Annual meeting of the North American Fuzzy Information Processing Society -NAFIPS'97*, pages 394–399. IEEE, 1997.

[3] A. Gray and S. MacDonell. A comparison of model building techniques to develop predictive equations for software metrics. *Information and Software Technology*, 39:425–437, 1997.

[4] Y. Miyazaki, M. Terakado, K. Ozaki, and N. Nozaki. Robust regression for developing software estimation models. *Journal of System and Software*, 27:35–16, 1994.