Applications of Fuzzy Logic to Software Metric Models for Development Effort Estimation

Andrew Gray
Software Metrics Research Laboratory
Department of Information Science
University of Otago, PO Box 56, Dunedin, New Zealand
Agray@commerce.otago.ac.nz

Stephen MacDonell
Software Metrics Research Laboratory
Department of Information Science
University of Otago, PO Box 56, Dunedin, New Zealand
Stevemac@commerce.otago.ac.nz

Abstract

Software metrics are measurements of the software development process and product that can be used as variables (both dependent and independent) in models for project management. The most common types of these models are those used for predicting the development effort for a software system based on size, complexity, developer characteristics, and other metrics. Despite the financial benefits from developing accurate and usable models, there are a number of problems that have not been overcome using the traditional techniques of formal and linear regression models. These include the non-linearities and interactions inherent in complex real-world development processes, the lack of stationarity in such processes, over-commitment to precisely specified values, the small quantities of data often available, and the inability to use whatever knowledge is available where exact numerical values are unknown. The use of alternative techniques, especially fuzzy logic, is investigated and some usage recommendations are made.

1. Introduction

In order to effectively develop software in an increasingly competitive and complex environment many organizations are making more and more use of software metrics as part of their project management process. Software metrics are aspects of software development (either the software product itself, or the development process producing that product) that can be measured [1]. These measurements can then be used as variables (both dependent and independent) in models for predicting or estimating some aspect(s) of the development process or product that are of interest. Models may also be developed in the same manner and used for classification or control tasks.

The most common application of software metrics is to develop models that predict the effort (often measured in person-hours or person-days) that will be required to complete certain stages of a software system's development. Generally speaking, such models are developed once the users' requirements have been ascertained and the specifications outlining the system are completed. It is at this point that most traditional software metrics can be derived from the available information. Thus, the stages in the development life cycle that are modeled are usually the physical design, programming and testing phases. There has also been some work towards even earlier modeling of effort based on requirements alone, such as [2].

Such models are of considerable importance to a number of diverse stakeholders, for a wide range of reasons. For the developer, manager, and user of any software product the prediction of project effort requirements is an extremely important activity [3,4]. The estimate arrived at frequently forms the basis for contract negotiations, resource and personnel allocation over the schedule, and charging for the project. It enables the project manager to plan, monitor and control the subsequent development process. It may have run-on effects for the user, in that their operations may be planned around the delivery of a particular supporting software product. Clearly then an accurate and robust effort estimation model is desirable from all perspectives.

There are many other applications of software metric models, such as assessing reusability, predicting maintenance costs, and measuring system reliability. The remainder of this paper is concerned with predicting development effort, but the results presented here apply in general to other prediction tasks, and classification and control models as well.

2. Software metric models for effort prediction

As an example of the use of software metrics in an effort prediction system, the size of a computer system could be measured in terms of the number of lines of source code, or the number of screens and reports contained in the specification. Similarly, the complexity may be measured in terms of the structure of the system, such as call paths or, for object-oriented systems, inheritance hierarchies. Other metrics that may be of interest include the quality of developers working on the system, as may perhaps be assessed in terms of years of experience, and the types of development and support tools being used. These descriptive measures of the system and the development process are software metrics. In the same way, the potential variables of interest here such as the required developer effort for programming and testing are also software metrics. Such a model is shown in Figure 1.

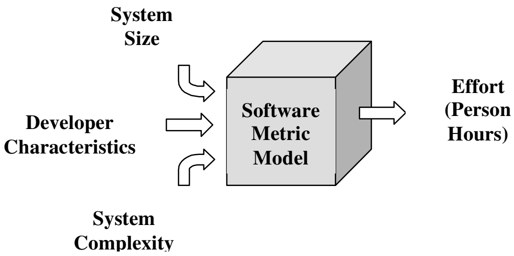

Figure 1. Software metric model for effort estimation

A software metric model may now be developed using data that contains some measures of system size and complexity, and developer ability in order to predict the effort required for various stages of the development life cycle. This is a fairly simple example of a model of what is an immensely complex and dynamic system (the process of developing software). Many other variables could be added, such as the reuse of previous systems, the demands and requirements for the system (such as maximum response time), the tools available, incentive schemes, the size of the team, and even the leadership style of the project manager. It should be noted that effort is the sum of developers' time spent on the project, and is distinct from the duration of the project.

Such predictive models are usually developed using linear regression analysis on available historical data for sufficiently similar projects, although there has been increasing use of other techniques, most notably fuzzy logic models [5], regression trees [6], neural networks [7], and case-based reasoning [8]. A useful summary of these techniques and their application to software metric modeling can be found in [9]. Predictions can then be made for new system development projects using the resulting model as the independent variables are estimated or become known for certain.

3. Difficulties with current models using software metrics

A number of difficulties with the current use of formal and linear regression models can be identified. Some of the major problems are discussed below.

3.1. Providing exact values for inputs

A major problem that exists with such models is the difficulty project managers face in specifying the exact values for the metrics used as inputs. Often they must use values that they anticipate will eventuate, since for many metrics the actual value is never known with certainty until the project is completed. Accurate specification of such independent variables is next to impossible for lower-level metrics such as source code length. Even higher level metrics such as the number of screens and reports can change greatly during development as the users' requirements are refined and better understood, or for some reason simply change. Using such models therefore demands a level of accuracy in prediction from project managers that is rarely possible early in the project life cycle, the very time that planning is most crucial.

3.2. Over commitment

The outputs from such models are almost always crisp values and this often leads to overconfidence in both the accuracy and precision of the results. When, for example, the dependent variable is predicted as 7120 developer hours (or even worse as 7121.6 developer hours), there is a risk that this value becomes sacred. This can lead to development time being wasted in the event of an overestimate, and requirements remaining unfulfilled or the project going over schedule to an even greater extent where the effort is underestimated. While confidence intervals *can* be developed, this is rarely undertaken, and given the small data sets available, often with skewed distributions, the intervals are often questionable in any event. The practice of rounding such values can be used, but the more precise values are still generally available, and even rounded values can be taken as accurate even though precision is removed.

3.3. The size of data sets

The next problem to be discussed here is that of the small data set size that is generally available for developing such models. While a larger organization may develop hundreds of systems in a year, they will often be so heterogeneous that relatively few can be used for predicting for certain types of systems. In addition, the non-stationarity of the contemporary software development process ensures that new programming languages, development environments, hardware platforms, and methodologies will emerge on a constant basis, making the available data less relevant. When high quality data is available in sufficient quantities, its proprietary nature generally leads to organizations being reluctant to allow its dissemination.

Many of the data sets used for software metrics are collections of projects from different organizations, thereby introducing additional sources of variation. Even where a large number of data sets may have been collected on sufficiently similar systems to permit a proper model development process, the data is often contaminated by outliers (either through measurement error or unusual system or project characteristics) and inaccuracies.

Thus the developer of a software metric model is faced with the need to develop, calibrate, and validate a model of an exceptionally complex process without the benefit of large quantities of accurate or representative data.

3.4. Interpretability of models

The final problem discussed here is that of specifying models in such a way that they retain some intuitiveness while including a sufficient subset of influential independent variables, and their interactions, to provide the required level of accuracy. The intuitiveness of a software metric model is often vital for it to gain acceptance by project managers. In order to provide the necessary level of comprehensibility models in the past have often been linear equations with a small number of independent variables. These models usually ignore the potential non-linearity of the relationships, the interactions between the independent variables, and the less important, but nonetheless influential, variables that cumulatively may be capable of explaining considerable variation in the dependent variable.

4. A fuzzy-logic approach to software metrics and models

The solution that is suggested here to, at least partially, overcome the previously mentioned problems is to use fuzzy logic variables for the metrics and models. In general it is considered that project managers can fairly readily specify independent variables in software metrics models using linguistic labels, such as a *large* number of screens and a *low* level of system complexity, in the early stages of estimation. It is also considered that such models provide considerable benefits in terms of reducing commitment, making full use of knowledge, and improving interpretability. In [5] the bold statement is made, that not only is fuzzy logic useful for effort prediction, but that it is essential in order to improve the quality of current estimating models.

4.1. Fuzzy labels as independent variables

Since many of the independent variables in software metric models are either difficult to quantify (for example complexity), or are only known to a rough degree (such as system size), the use of fuzzy variables seems intuitively appealing. It is our conjecture here that project managers are in fact able to classify systems using fuzzy variables with reasonable levels of both accuracy and consistency.

While complexity can be defined in an algebraic sense, and in fact it has been defined in a large number of ways in the past, such formal definitions are always to some extent arbitrary. The software metrics literature is filled with debates as to the relative merits, and demerits, of various definitions. It is instead suggested here that complexity is a multifaceted concept, but that experienced project managers would be able to make fairly consistent classifications of projects in terms of one or a small number of types of complexity. This has the added advantage of reducing the number of variables used as inputs into the model. This point is discussed further later in the paper in terms of work in progress.

4.2. Reducing commitment through fuzzy outputs

Particularly at the very early stages of a software development project, estimating to within one person-hour or person-day is simply not realistic. Instead, a fuzzy system may be used to transform linguistic labels, or numerical values, indicating system size and complexity, personnel experience, and other factors of influence into an equally imprecise (but adequate for its purpose) label indicating predicted effort, for example *very high*. While this approach may well be imprecise, this is justified and should ensure that personnel associated with the project do not attach unwarranted accuracy to the figures produced. As the project progresses and a greater degree of certainty is established in relation to the scope of the project (and also as data starts to become available), then more precise indicators of effort may be formulated, either through more and smaller membership functions or allowing for numerical defuzzification. There is an inherent trade-off in the development of effort estimation models; is it better to be approximately correct most of the time or precisely inaccurate all of the time?

4.3. Better use of knowledge and data

Since a fuzzy logic model can be initialized with expert rules, and given that the movement of membership functions and rules can be limited, it is possible that such a model will perform significantly better than alternatives such as regression and neural network models given small quantities of data.

4.4. Model interpretability

One final point that deserves mention is that the application of fuzzy logic to the estimation problem allows for *model transparency*. A fuzzy system provides the potential for those involved to view, evaluate, criticize, and even adapt the models. This is not always possible in statistical or other machine learning modeling approaches.

5. Empirical case study

The case study below is based on actual project data from [10]. The data set includes measures of:

- project effort
- project duration
- levels of experience with equipment
- levels of experience in project management
- numbers of basic transactions
- numbers of data entities
- raw and adjusted function point counts, which are a standard method for counting the functional size of a system, with adjustments for complexity also possible.

Although the data is quite real, it is used here mainly to illustrate the capabilities and drawbacks associated with the various analysis methods available. Four approaches are compared: Function Point Analysis (which is a formal model), regression techniques, feedforward neural networks, and finally fuzzy logic. The results for the Function Point Analysis, regression models, and neural network models have been previously reported in more detail in [11].

The issue of making the fullest use of information available, which is one area where fuzzy logic excels, is difficult to illustrate in a post-hoc case study. Since fuzzy variables are not available for earlier in the projects' life cycles, the accuracy of such models can not be shown here. It is regarded that such models would have performed comparatively well, since the other techniques depend more on the availability of data from later in the development life cycle.

Many different methods for estimating a model's goodness for prediction are available. These include the many forms of correlation (R^2, adjusted R^2, R^2 adequate), Akaike Information Criterion, Bayesian Information Criterion, and mean square error. A set of indicators is commonly used in metrics analysis to indicate the adequacy of a predictive model; namely the mean magnitude of relative error and the threshold-oriented pred measure. Both of these are used in this case study on a validation data set (27 of the total 81 observations) for each technique.

The magnitude of relative error (MRE) is a measure of the difference between the actual values of the dependent variable (V_A) and the predictions of the model (V_F):

$$MRE = \frac{|V_A - V_F|}{V_A}$$

The mean MRE (MMRE) is therefore the mean value for this indicator over all observations in the sample. For software development project management relative errors are often a better judge of the model's accuracy.

The pred measure provides an indication of overall fit for a set of data points, based on the MRE values for each data point:

$$pred(l) = \frac{i}{n}$$

where l is the selected threshold value for MRE, i is the number of data point with MRE less than or equal to l, and n is the total number of data points. As an illustration, if pred (0.25) = 30%, then we can say that 30% of the fitted values fall within 25% of their corresponding actual values.

The various techniques for modeling are now discussed in some detail with explanations provided as to how the final models were selected.

5.1. Function points

Function points are currently the most commonly used formal software metric modeling technique. Function Point Analysis (FPA) [12,13,14] provides a well-established method for the relatively early (post specification) assessment of system scope. Several versions of FPA have emerged, with specialized versions also created for particular types of systems. These are all based on various transaction-oriented system requirements characteristics.

In many respects FPA already contains some degree of fuzziness, with levels of complexity recognized for functions. Although FPA is not without its potential problems, especially inter-rater subjectivity which tends to be very high, it remains one of the most widely used methods for modeling software development.

Two versions of FPA are shown here. The standard use of FPA for effort prediction is to use the historical mean effort per function point. This is complemented here by the use of the median effort per function point, which is a more robust estimator.

5.2. Linear regression

Two forms of linear regression are illustrated here. Firstly, standard least squares (LS) is used as a demonstration of the most commonly used technique for developing software metric models. Secondly, least median squares (LMS, a robust regression technique that uses weighted regression to remove outliers) is illustrated. The two cases differ only in terms of the goal function that they attempt to minimize, namely the mean square error and median square error respectively.

In both cases stepwise procedures were used to select linearly influential variables, with no interaction terms entered. Both models resulted in a simple linear equation involving a constant term and the unadjusted function points.

5.3. Feed-forward neural network

Neural networks have been applied to software metric modeling in a large number of papers and the results have, in general, been favorable to this particular technique where sufficiently large data sets have been available.

The Multi-Layer Perceptron (MLP) networks were trained using two-thirds of the 54 model-development observations for training, and one-third for a testing set. Training was stopped when the testing error was minimized, and the lowest testing error was also used to select the particular network architecture.

5.4. Fuzzy logic

A significant motivation for using fuzzy logic is not under the circumstances of the following post-hoc analysis, but rather in the ability to estimate required effort much earlier in the development process.

For this analysis, the two most influential variables were selected as the raw function points count and the associated complexity adjustment factor. These were divided into three equi-spaced triangular membership functions for small, medium, and large size or complexity. The same procedure was then carried out for the effort data. An experienced software developer was then asked to provide the initial set of nine rules, which were then hand-adjusted in consultation with the expert to achieve a better fit to the training data set.

5.5. Comparison of techniques

Each technique's best model was then used to predict for the remaining 27 observations in the validation set. As can be seen in the results in Table 1 and Figure 2, the most accurate model in terms of MMRE is the neural network, followed by the fuzzy logic model. This is to a large extent due to the non-linearities and interactions present in the data set, which is barely large enough for such features to be taken into account with regression analysis. However, in terms of classification accuracy, the neural network model is fairly comparable to the least squares regression model after outlier removal from the training and testing data based on residual analysis.

Table 1. Comparison of results

Method	MMRE	pred(10)	pred(25)
FPA estimation (mean-based)	0.70	4%	22%
FPA estimation (median-based)	0.89	19%	41%
LS regression	0.86	15%	41%
LS regression (no outliers)	0.88	30%	56%
LMS regression	0.85	7%	41%
Neural network	0.44	26%	63%
Fuzzy logic	0.54	7%	30%

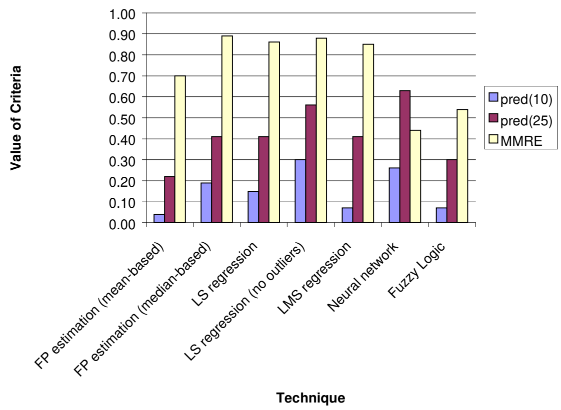

Figure 2. Comparison of results

Clearly these performance indicators are not in themselves very encouraging and one would hope for much more accurate predictions in order to effectively manage the development process. The objective of this case study, however, was to compare a selection of analysis methods using the same data set, so as to emphasize the potential of the various analysis options and their capacity to provide effective general models for estimation.

Again it is stressed that the numerical accuracy of such models is not the only selection criteria that should be used. The ability of fuzzy logic to avoid over-commitment to particular predictions and its interpretability needs to be kept in mind, as does the simplicity of the regression models.

6. Other aspects of using fuzzy variables for software metric models under investigation

The current study represents only the initial stage in a longer-term project to assess the suitability of, and develop models using, fuzzy logic in software metric modeling. A range of data sets has been collected and is currently being analyzed using a variety of modeling techniques including fuzzy logic.

An important requirement for using models with linguistic inputs is that project managers are indeed able to specify such inputs in a reliable and consistent manner. In order to assess the truth of this assumption a number of experienced developers are being asked to describe aspects of systems based on specification documentation. The only known study of the consistency of such rankings is [15] where on a very small data set they found some consistency in metrics for source code analysis.

Work is also currently underway to ascertain how best to gather this information from software development and project management experts. Three main areas arise in this problem: the selection of appropriate experts, the techniques used to gather information from such experts, and the approach to performing the knowledge acquisition task [16]. The technique currently favored is providing the experts with some understanding of fuzzy logic, and then walking them through the creation of membership functions and matrices of rules.

Finally, work on automatically extracting the fuzzy rules from the available data and on fine-tuning expert-provided rules using such data is being investigated.

7. Conclusions

The idea of using fuzzy logic for defining software metrics as linguistic variables and for the modeling process has been outlined in this paper. The motivation for this has been the difficulties faced by software metricians in terms of avoiding premature and costly commitment, using all available knowledge, having only small data sets to work with, and also the need for transparent models.

Compared to other techniques the fuzzy logic model developed for the case study shows good performance, being out-performed in terms of accuracy only by the neural network model with considerably more input variables. Given the other advantages of a fuzzy logic model this suggests that there is a place in the field of software metrics for such models.

References

[1] N. Fenton. *Software Metrics, a Rigorous Approach.* Chapman & Hall, London, 1991.

[2] T. Mukhopadhyay and S. Kekre. Software effort models for early estimation of process control applications. *IEEE Transactions on Software Engineering*, 18(10):915-924, 1992.

[3] A.L. Lederer, R. Mirani, B.S. Neo, C. Pollard, J. Prasad, and K. Ramamurthy. Information system cost estimating: a management perspective. *MIS Quarterly*, 159-176, June, 1990.

[4] B. Londeix. Deploying realistic estimation (field situation analysis). *Information and Software Technology*, 37:665-670, 1995.

[5] S. Kumar, B.A. Krishna, and P.S. Satsangi. Fuzzy systems and neural networks in software engineering project management. *Journal of Applied Intelligence*, 4:31-52, 1994.

[6] R.W. Selby and A.A. Porter. Learning from examples: generation and evaluation of decision trees for software resource analysis. *IEEE Transactions on Software Engineering*, 14:1743-1757, 1988.

[7] K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort, *IEEE Transactions on Software Engineering*, 21:126-137, 1995.

[8] T. Mukhopadhyay, S.S. Vicinanza, and M.J. Prietula. Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS Quarterly*, 16:155-171, 1992.

[9] A.R. Gray, and S.G. MacDonell. A comparison of model building techniques to develop predictive equations for software metrics. *Information and Software Technology*, to appear, 1997.

[10] J.-M. Desharnais. *Analyse statistique de la productivitie des projects de development en informatique apartir de la technique des points des fontion.* Master's Thesis, Universite du Montreal, 1989.

[11] S.G. MacDonell and A.R. Gray. Alternatives to regression models for estimating software projects. *In Proceedings of the IFPUG Fall Conference*, Dallas TX, IFPUG 279.1-279.15, 1996.

[12] A.J. Albrecht. Measuring application development productivity. In *Proceedings of the IBM Applications Development Joint SHARE/GUIDE Symposium*, Monterey, CA, 83-92, 1979.

[13] J.E. Matson, B.E. Barrett, and J.M. Mellichamp. Software development cost estimation using function points. *IEEE Transactions on Software Engineering*, 20(4):275-287, 1994.

[14] J.J. Dolado. A study of the relationships among Albrecht and Mark II function points, lines of code 4GL and effort. *Journal of Systems and Software*, 37:161-173, 1997.

[15] R.I. Kilgour, A.R. Gray, P.J. Sallis, and S.G. MacDonell. A fuzzy logic approach to computer software source code authorship analysis. Submitted to The Fourth International Conference on Neural Information Processing -- The Annual Conference of the Asian Pacific Neural Network Assembly (ICONIP'97).

[16] Y.I. Liou. Knowledge acquisition: issues, techniques and methodology. *DATABASE*, 59-64, Winter 1992.