Fuzzy Logic for Software Metric Models throughout the Development Life-Cycle

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

Abstract

One problem faced by managers who are using project management models is the elicitation of numerical inputs. Obtaining these with any degree of confidence early in a project is not always feasible. Related to this difficulty is the risk of precisely specified outputs from models leading to overcommitment. These problems can be seen as the collective failure of software measurements to represent the inherent uncertainties in managers' knowledge of the development products, resources, and processes. It is proposed that fuzzy logic techniques can help to overcome some of these difficulties by representing the imprecision in inputs and outputs, as well as providing a more expert-knowledge based approach to model building. The use of fuzzy logic for project management however should not be the same throughout the development life cycle. Different levels of available information and desired precision suggest that it can be used differently depending on the current phase, although a single model can be used for consistency.

1. Introduction

Fuzzy logic modeling techniques have been shown to be a useful addition to the existing statistical and machine-learning techniques used for modeling software development [3]. Aside from theoretical reasons preferring fuzzy logic in some circumstances, several papers have shown favorable empirical comparisons supporting its usefulness by using software metric data sets to compare the predictive accuracy of various techniques [2, 5, 7]. In addition, fuzzy logic modeling software has been especially developed for supporting the project estimation process [6, 7].

A recent survey of New Zealand project managers found them to have considerable interest in using fuzzy logic techniques [7]. This survey found that 31 out of the 44 responding information system managers had heard of fuzzy logic (70.5%). For the 36 managers who were actively involved in managing development projects, 11 (31%) were interested in using fuzzy logic techniques, 23 (64%) stated that they would need to know more about the technique before making a decision, and only two (6%) did not think that fuzzy logic techniques would be useful to them as part of their management activities.

In addition to assessing the managers' perceptions of the worth of fuzzy logic, the survey also investigated which advantages of fuzzy logic were felt to be important. The three choices of being able to use expert knowledge, having linguistic inputs, and producing linguistic outputs were all rated roughly the same (with 19, 19, and 21 respondents citing each as important respectively).

As the level of commercial interest in using this technique "in anger" grows it becomes necessary to provide well-documented and replicable standards for its implementation. The most successful software metric model for effort estimation is Function Point Analysis [1], and a hallmark of this has been its carefully documented procedures (in both standards and many publications), certification, and workshops. While it would be premature to impose such restrictions on the use of fuzzy logic techniques at this early stage of its adoption in this field, it does seem prudent to outline some general skeleton guidelines to encourage the, somewhat inconsistent, goals of experimentation and rigor. In this paper, the selection of input and output precision levels based on the stage of the development life cycle is discussed.

2. The software development life-cycle

Many alternative representations have been proposed for how software is, and ought to be, developed [9]. Different models exist for different types of system, such as object-oriented systems, where the idealized development process is considered by some to be fundamentally different to other types of development. Similarly, the use of prototypes (both low- and high-fidelity), customer reviews, and other activities may differ from one organization to the next.

Here we consider only the fundamental phases of development that are ubiquitous to almost all such models found in practice and the literature. Namely, the analysis, design, coding, testing, and maintenance phases. Each phase is briefly defined below (in deliberately vague terms to encompass common usage as widely as possible). These phases can also be shown graphically as in Figure 1, which also indicates the common notion of iteration and feedback between consecutive phases.

2.1. Analysis

This is the first stage of the development process and starts with the problem being defined and initial user requirements being collected. This phase can include the construction of simple prototypes used to determine or fine-tune the requirements, but should not involve any detailed design or real coding.

Very little information beyond the high-level functions required of the system is available this early in the project's life-cycle. Unfortunately, this is also the phase where planning is most crucial in terms of both time-to-delivery and financial cost estimation. Such information is necessary for contract negotiation (both inter- and intra-organizational) and strategic planning.

In this and the next two stages (design and coding) the main emphasis of software metric models is the prediction of development effort, and from this some estimates of cost and duration. While other dependent variables can be of interest, the focus in this paper is with this primary application of effort estimation. Similar ideas can be easily generalized for other metric applications.

2.2. Design

The requirements must then be translated closer to the actual implementation. This is when the system specifications are developed (for example, Entity-Relationship Diagrams, Data Flow Diagrams, Structure Charts, and pseudo code routines). Ideally, the system should be understood to a high degree at this stage, with the coding stage involving the implementation of the necessary functions.

2.3. Coding

Actual source code is written during the coding phase (including visually generated code and automatic template code). This may also include ever more sophisticated prototypes that evolve (at least partially) into the final system.

2.4. Testing

Testing can be performed concurrently with development, or may occur after most code has been written. When

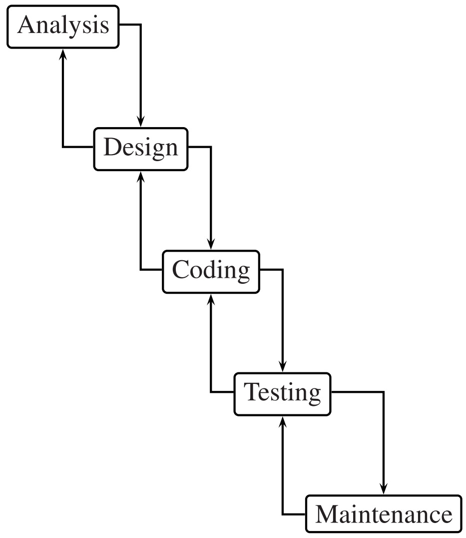

Figure 1. Stages in a generic life-cycle model

the system is tested metric models may be used for estimating testing effort or for predicting the number of defects remaining. This phase includes functionality and usability testing and in many cases this is the first time the user sees the actual system properly executing. At this stage the system should be complete in terms of functionality so a large amount of information about it is available for modeling.

2.5. Maintenance

When the system is modified to add new functionality or correct defects after it has been released to the customer some estimate of maintenance effort may be made. Unlike the effort estimates for the previous four phases, which can all be combined to give the total for the system, effort on maintenance is often treated separately. Estimates can be made for the entire maintenance process if this is sufficiently trivial, or this phase can be treated as a new development process itself.

3. Advantages of fuzzy logic for software metric models

Fuzzy logic modeling techniques offer several potential advantages over more traditional techniques for software metric models. These have already been discussed at considerable length in the literature and so are only briefly mentioned here for reference purposes. The interested reader is referred to [2, 3, 5] for more detailed discussion.

3.1. Data requirements

Fuzzy logic allows model development with little or even no data. This is a considerable boon given the problems with data gathering in software metrics research and practice. The collection of homogeneous data sets is complicated by rapidly changing technologies and a reluctance for inter-organizational sharing of metrics data. Even within a single organization there can be considerable pressure from programmers and managers against measurement collection.

3.2. Robustness

Software metric data sets are likely to contain unusual systems that result from a variety of causes and may reduce the generalisability of any empirically derived model [8]. Some of these problems include different development practices, developer learning, and unmeasured (and perhaps unmeasurable) influences. By developing models with considerable expert involvement, where the model can be interpreted and checked for reasonableness, some of the problems with non-representative data corrupting empirically tuned models can be reduced or perhaps even avoided.

3.3. Organizational process learning and communication

The use of fuzzy logic models provides an opportunity to learn from the resulting models that is less evident with regression and (even more so) neural network models. A linguistically-based model can also be seen as a useful communication tool. For example, a programmer pointing out that complexity for a particular module is *very high* and may need rework may be considerably more meaningful to a manager than them stating that the module's cyclomatic complexity is 55. In addition, since the models are relatively easily understood by management there is a grater chance of management support, which is essential for the success of any metrics program.

4. Fuzzy logic models throughout the development life-cycle

One of the important benefits of fuzzy logic for software engineering project management is the flexibility available in terms of the types of input and output variables. Input variables can be expressed as simple fuzzy labels (a *large* number of entities in the data model), fuzzy numbers (*about 250* entities), or using precise values (*265* entities). Similarly, the output can be expressed in the same way, as a label (a *short* development time), fuzzy number (*about 400* developer-hours), or precise values (*378* developer-hours).

Phase	Inputs	Outputs
Analysis	fuzzy label	fuzzy label
Design	fuzzy number	fuzzy number
Coding	crisp value or fuzzy number	fuzzy number
Testing	crisp value	fuzzy number
Maintenance (small project)	crisp value	fuzzy number

Table 1. Suggested levels of precision across the life-cycle when estimating development effort

Perhaps the greatest benefit from this approach is that the same model (the membership functions and rules) can be used throughout the development process, simply changing the levels of precision as required. This has several advantages over multiple model methods including the improved consistency of predictions, centralized model building and implementation, model building effort minimization, and knowledge gathering over the entire development process.

The following subsections discuss the representation of independent and dependent metric variables using fuzzy logic. Table 1 and Figure 2 show a summary of these suggestions.

Where a context is necessary in the discussion below, the most common task for metric models is used. That is development effort prediction based on system characteristics (including the product, process, and associated resources). The *standard* inputs into such a model are generally onee or more size and complexity measures, with perhaps some developer productivity adjustment. Developer effort may be estimated on the basis of the entire system, a specified component of the system, for a particular phase, or for both a component and phase together.

It should also be noted, that as the development process is enacted, more is known about the *actual* effort which is generally a component of the effort being estimated. As such the expectation is that model performance will improve roughly monotonically irrespective of the modeling technique used.

4.1. Analysis

The analysis phase is one of the most difficult times to make predictions. This is because almost all existing software metric models assume that precise values relating to the system specification are available as independent variables. For example, Function Point Analysis assumes that the numbers of external inputs, external outputs, external inquiries, external files, and internal files are all known. It also requires each to be individually rated as simple, average, or

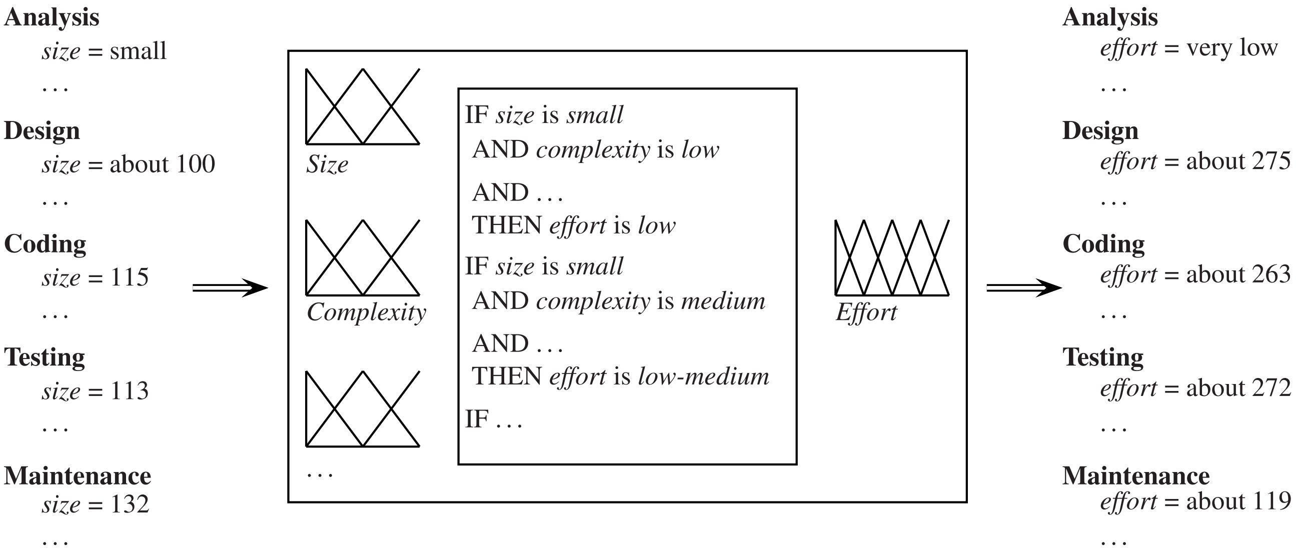

Figure 2. Different levels of input and output precision with the same set of membership functions and rules

complex. Optionally, some subjective technical complexity factors can be used to further refine the estimated size (which is usually translated into an effort estimate using a developer hour per function point approach) [1].

However, in many cases the numbers of these system components, and even more so their specific complexities, are not known with any degree of certainty until the design phase is almost finished, let alone during the analysis phase itself. This is one area where the natural uncertainty of fuzzy logic provides a useful means of representing the approximate numbers of components and their average complexity (or whatever other metrics are used). While managers can try to provide precise values for standard models, such results are unlikely to be taken seriously given the obvious guesswork and there is a natural reluctance to provide values to a level of precision beyond the estimator's capabilities.

Of course, different models could be used at each phase of development with regression or neural network techniques. Earlier models could use categorical labels for the number of functions, for example. However, this then leads to the problems with multiple models that have already been discussed above in §4.

Similarly, the outputs from software metric models are usually numerical values, such as *5251* developer hours. This can introduce problems with over-commitment where the estimate becomes a *sacred number*. When the estimate is later revised, perhaps due to more knowledge becoming available, changes can be seen as reflecting instability in the estimation process or the project itself. Increasing the estimated cost and duration is often associated with problematic projects, which could lead to a politically-motivated reluctance to update estimates as frequently as may be beneficial to the organization. If estimates were instead expressed as fuzzy labels, such as this is a *high effort* project, then commitment to precise values can be delayed until these values can be estimates with enough accuracy to be meaningful.

Of course, predictions from standard models can be rounded if this is desired. However, rounding a value to the nearest thousand hours does not guarantee that it will be seen as an estimate plus or minus 500 hours.

It is this stage, and the following design phase, that seem most suited to fuzzy logic modeling since actual numerical values are generally not available and exact estimates of effort can potentially even be harmful.

4.2. Design

During design the system specifications should be drafted and reviewed. This may include Entity-Relationship Diagrams showing the numbers of entities, relationships, and elements; Data Flow Diagrams showing the data sources, sinks, and flows; Functional Decomposition Charts showing the breakdown of the system into actual functions; and pseudo-code showing the algorithms and program logic from a higher level perspective [9]. All of these are commonly used as part of software metric models [1].

Since the specifications, and thus the measurements, are not available in their final form until the end of this phase, and are subject to subsequent changes due to the dynamism of customer requirements or for technical reasons, it makes sense to represent the measures as approximate values. It is suggested that fuzzy numbers are useful for this purpose, allowing both the center (best estimate) and the degree of confidence (spread) to be represented. For example, the data model complexity may be assessed as *about 50* 1-m relationships or *very close to 55* 1-m relationships depending on the manager's confidence in the estimate.

In the same way, estimates of development effort can be made using fuzzy numbers, with any desired level of linguistic precision. For example, *approximately 5000* developer-hours or *almost exactly 5125* developer hours.

4.3. Coding

By the coding stage most common metric models can be used with actual values from the system specification and the use of fuzzy logic seems less useful. At this point, using exact values as inputs into the model seems far more sensible if they are indeed available.

However, some use of fuzzy logic estimates for subjective concepts, such as program complexity, can still be used [4]. Such concepts are often captured by a series of arbitrary measurements that can be difficult to obtain, whereas human experts may be able to quickly and accurately ascribe a fuzzy value to them.

Fortunately, as noted earlier the same fuzzy logic model can still be used here irrespective of the actual precisions used. Fuzzy numbers may still be preferred for outputs even though exact inputs are being used in order to maintain the appearance of uncertainty.

4.4. Testing

Testing effort metric models can depend on either specification or code-based measurements, so such predictions may be performed using either fuzzy logic or crisp values depending on availability. As with coding, fuzzy logic estimates may be superior to numerical estimates for some concepts, such as complexity. Again, fuzzy numbers for outputs may be a useful way to keep the uncertainty in the estimate clear.

4.5. Maintenance

As was mentioned above, maintenance projects can be either run as single entities, or may be treated as new development projects (with the other phases included within them). The choice of precision will obviously depend on the scale of the project.

5. Conclusions

Give its ability to represent differing levels of uncertainty for inputs and outputs whilst still basing inference on the same model, fuzzy logic is well suited to a life-cycle approach to software metric modeling. This is a unique opportunity not previously available from standard software metric models that are primarily developed using measurements from a single phase in a project's life. The ensuing consistency, communicatability, and economy make this an attractive modeling technique for such applications as effort estimation.

The other advantages of data-free (or data-poor) model building, more robust models, and improved communication further enhance the opportunities from using fuzzy logic for software metrics. We are currently entering a phase of industrial collaboration with several large New Zealand commercial organizations where the ideas discussed in this paper will be trialed and refined.

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] R. Kilgour, A. Gray, P. Sallis, and S. MacDonell. A fuzzy logic approach to computer software source code authorship analysis. In *Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems*, pages 865–868. Springer-Verlag, 1997.

[5] S. MacDonell and A. Gray. A comparison of modeling techniques for software development effort prediction. In *Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems*, pages 869–872. Springer-Verlag, 1997.

[6] S. G. MacDonell and A. R. Gray. Fulsome: a fuzzy logic modeling tool for software metricians. In this volume.

[7] S. G. MacDonell, A. R. Gray, and J. Calvert. Fulsome: A fuzzy logic toolbox for software metric practitioners and researchers. Submitted to ICONIP'99.

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

[9] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, fourth edition, 1997.