Empirical Software Engineering, 7, 157–175, 2002.

Estimating Maintenance Effort by Analogy

HARETON K. N. LEUNG cshleung@comp.polyu.edu.hk
Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

Abstract. Effort estimation is a key step of any software project. This paper presents a method to estimate project effort using an improved version of analogy. Unlike estimation methods based on case-based reasoning, our method makes use of two *nearest neighbors* of the target project for estimation. An additional refinement based on the relative location of the target project is then applied to generate the effort estimate. We first identify the relationships between cost drivers and project effort, and then determine the number of past project data that should be used in the estimation to provide the best result. Our method is then applied to a set of maintenance projects. Based on a comparison of the estimation results from our estimation method and those of other estimation methods, we conclude that our method can provide more accurate results.

Keywords: Effort estimation, maintenance projects, analogy.

1. Introduction

One of the most important activities in software development is to estimate effort and determine resources for a project. Good project estimation supports effective use of valuable resources. Accurate effort estimates are a means by which the project manager can get a handle on the scale and scope of the project and make important decisions on how it will be tackled.

All estimating methods are prone to error, as they depend to some degree on subjective views of the size and complexity of the project tasks. No single method can give a *right* result all the time, and, in fact, a right result is probably unachievable. Inevitably, factors will arise that will invalidate many of the estimating assumptions and there will be a need to revisit the estimates at later stages of the project.

There are two major types of cost estimation methods: *Algorithmic* and *non-algorithmic*. Algorithmic models vary widely in mathematical sophistication. Some are based on simple arithmetic formulae using such summary statistics as means and standard deviations (Donelson, 1976). Others are based on regression models (Walston and Felix, 1977) and differential equations (Putnam, 1978). To improve the accuracy of algorithmic models, there is a need to adjust or calibrate the model to local circumstances. Even with calibration the accuracy can be quite mixed.

The most popular algorithmic methods are COCOMO and Putnam's models, both are based on power function models of the general form:

$$\text{Effort} = a \times S^b$$

where S is the code-size, and a, b are usually simple functions of other cost factors.

(1) Constructive cost models (COCOMOs): This family of models was proposed by Boehm (1981) and Boehm et al. (1995). The models have been widely accepted in practice. In the COCOMOs, the code-size S is given in thousand line of code (KLOC) and Effort is in person-month.

The original COCOMO81 model exists in three versions (*basic, intermediate* and *detailed*) and consists of formulae for calculating the effort and elapsed time needed to develop software based on an assessment of the size of the system to be developed. Due to the changes in professional software development practice that have occurred in the 70s and 80s, the performance of this model deteriorated.

In the revised cost estimation model, COCOMOII, some of the most important factors contributing to a project's duration and effort are the *scale drivers*, which determine the exponent used in the Effort equation. The scale drivers have replaced the development mode of COCOMO81. Recent work at University of Southern California indicates that this model achieves an accuracy of within 30% of actuals 75% of the time (SCU)[1].

The main disadvantage of the COCOMO is that it depends upon an assessment of code size. The obvious problem with this is that it is not possible to estimate accurately the likely code size until quite late in the project. In addition, it is not entirely clear how the COCOMO formulae should be interpreted for projects using more advanced programming tools and fourth generation programming languages. Also, there is considerable evidence that this approach is not always successful (Kemerer, 1987; Thayer and McGettrick, 1993).

(2) Putnam's model: Putnam derives his model based on Norden/Rayleigh manpower distribution and his finding in analyzing many completed projects (Putnam, 1978). The central part of Putnam's model is called *software equation* as follows:

$$S = E \times (\text{Effort})^{1/3} t_{\text{d}}^{4/3}$$

where t_{d} is the software delivery time; E is the *environment factor* that reflects the development capability, which can be derived from historical data using the software equation. The size S is in LOC and the Effort is in person-year. Another important relation found by Putnam is

$$\text{Effort} = D_0 \times t_{\text{d}}^3$$

where D_0 is a parameter called *manpower build-up*, which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software).

Others have tried discrete models with a tabular form, which usually relates the effort, duration, difficulty and other cost factors. This class of models contains Aron model (1969), Wolverton model (1974), and Boeing model (Black et al., 1977). These models gained some popularity in the early days of cost estimation, as they were easy to use.

The common non-algorithmic methods are expert judgment, Parkinson, and analogy-based methods.

(1) Expert judgement: The expert judgement method involves consulting one or more experts. Staff members with extensive experience in the specific area are asked to provide the effort estimation. This method is fast and accurate if the estimators are truly experts in the specific area. However, it is not a systematic and objective method.

(2) Parkinson method: According to Parkinson's principle, 'work expands to fill the available volume' (Parkinson, 1957). The cost is determined (not estimated) by the available resources rather than based on an objective assessment. If the software has to be delivered in 12 months and five people are available, the effort is estimated to be 60 person-months. Although it sometimes gives good estimation, this method is not recommended as it may provide very unrealistic estimates.

(3) Analogy-based: The analogy method is one of the oldest and most reliable methods and depends on finding a project similar to the current one that has been undertaken in the organization before. The *similarity dimensions* may include application type, language used, development method, and functionalities. The organization will need good historical data. The accuracy will depend whether a similar project or a group of similar projects can be found in the project database.

Where there are differences in any of the similarity dimensions, suitable adjustments must be made. For example, if the historical project was developed in COBOL, but the new project is to use a fourth-generation language, then the programming effort for the new project should be adjusted downward.

More recently, attention has turned to a variety of machine learning methods to predict software development effort. Artificial neural nets, case based reasoning (CBR), and rule induction are examples of such methods (Gray and MacDonell, 1997; Srinivasan and Fisher, 1995).

For example, Wittig and Finnie (1994) used neural nets with a back propagation learning algorithm to obtain an error rate of 29%, which compares favorably with other methods. However, it must be stressed that they required a large data set of 136 projects, and that only a very small number of projects were used for validation purposes. Neural nets seem to require large training sets in order to give good predictions.

Recently, Shepperd and Schofield (1997) used a pure CBR strategy to estimate project effort. In the CBR method, historical project metrics and their features are recorded. A project represents a point in an 'n-dimensional' space using the recorded

parameters and the project development effort. For new project estimation, the *location* of the new project, i.e. the values in each of the n dimensions, is first determined. Then, the nearest neighbor in the n-dimensional space of this location is computed. The project effort of the nearest neighbor is used as the basis of estimation. Shepperd and Schofield achieve a 50% error rate using data from only 12 past projects. This indicates that it is possible to obtain a reasonable estimation result based on a small set of historical data. Note that estimating project effort using analogies can be viewed as a form of CBR.

Mukhopadhyay et al. (1992) described some early work using a hybrid CBR and rule-based system. They reported encouraging results of a 43% error rate when their method was applied to the data set collected by Kemerer (1987).

Others have reported that regression based analysis generated more accurate models than using CBR (Briand et al., 2000). There is also some evidence that economies and diseconomies of scale prevail in new software development. This implies that simple linear models may not be suitable for estimating project effort (Banker et al., 1994).

Gray and MacDonell (1997) have reported a detailed comparison of various methods for estimating effort.

The analogy-based method offers some advantages when compared to many other approaches:

- Since the data are derived from a form of reasoning more similar to human problem solving, as opposed to the somewhat arcane chains of rules or neural nets, users may be more willing to accept solutions from the analogy-based system.
- Analogy is able to deal with poorly understood and multi-variable domains, since the solution is based upon what has actually happened as opposed to chains of rules in the case of rule-based systems.
- Analogy-based systems avoid the problems associated with both knowledge elicitation and extracting. There is no need to codify the knowledge.
- Analogy-based systems can be applied in the very early phase of project development. It enables a *broad-brush* estimate for a whole project during the preparation of a bid or a proposal.
- Analogy-based systems can deal with differently scaled similarity dimensions.

However, there are also some areas of improvement when using the analogy-based method.

- The accuracy of the method depends on finding a similar project.
- Typical analogy-based methods assign equal weight to all similarity dimensions. However, for critical and important dimensions, a 'heavier' weighting factor seems to make more sense.

- In case certain similarity dimensions are missing, there is a need to determine appropriate adaptation rules.

A method that can provide *better* estimation of project effort, especially one that can be applied relatively early in the development cycle, would be of considerable interest to the software development community. The objective of this study is to investigate an estimation method based on analogy with refinements. Our aim is to develop a method that can provide highly accurate estimation at the requirement stage. We also aim to determine the number of past data that will be required to provide an accurate result. Our primary interest is the effort estimation of maintenance projects which may require a few days to several weeks to complete. Section 2 presents the estimation by the *analogy with virtual neighbor* (AVN) method. The enhancement made on top of the traditional analogy-based method will be described. Section 3 highlights some empirical experiments applying the AVN method. The accuracy of the AVN method will be compared to three other methods. The conclusion and idea for future work are provided in Section 4.

2. Estimation by Analogy with Virtual Neighbor

The estimation by AVN method consists of two stages. Stage 1 develops the cost-driver and effort relationship, while Stage 2 uses the result of Stage 1 to perform project estimation. Stage 1 involves 3 steps, while Stage 2 involves 2 steps.

Stage 1 first identifies all possible *dimensions* or characteristics that may serve as cost drivers. As most projects reuse code from previous projects or software library, a key cost factor is the reuse percentage. The actual effort with reuse will be different from the project effort without reuse. During the effort estimation, we need to factor in the impact of reuse.

We next discard those dimensions that are not contributing to the estimation or are unavailable in practice. Finally, we apply linear regression to relate these contributing dimensions to the project effort.

When an estimate of the effort for a new project is required, Stage 2 of the AVN is invoked. We introduce two refinements to improve on the analogy-based approach:

(1) Use of two nearest neighbors: We use two nearest neighbors as the basis of estimation, as they should have a higher referencing value than other project points and overcome some of the problems associated with using just one nearest neighbor.

(2) Proportional refinement: The AVN method further adjusts the effort estimation by compensating for the relative location of the target project with respect to its two nearest neighbors.

Figure 1 shows the key steps of the AVN method.

Step 1: Identify a standard set of cost drivers. We first identify those cost drivers that may influence the software development schedule and effort. It is critical that the organization collects a *sufficient* set of key costs drivers.

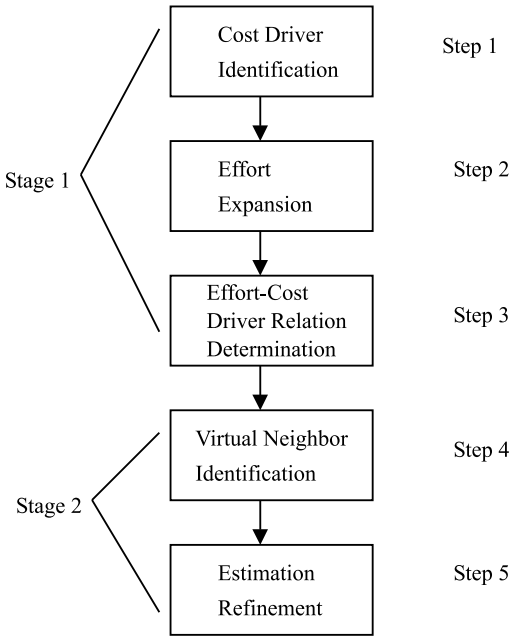

Figure 1. Key steps of the AVN method.

The cost drivers are selected based on the following criteria:

- They must be measurable and collectable in real life. Since the aim of this study is to estimate project effort early in the system development life cycle, a key selection criterion of cost drivers is that they should be collectable in the requirement phase.
- They are considered as important by experienced staff. We asked three project leaders to propose cost drivers that may affect the effort. They indicated that the factors should be classified into simple, medium and complex, based on 'complexity'.
- They are not cost drivers used in the COCOMO, as we would like to see whether other factors can be used for estimation.

Step 2: Determine the expanded actual effort. Next, we *expand* the actual effort. In most cases, the recorded effort data of past projects represent the project effort with some degree of reuse. As the amount of reuse will affect the actual effort for a project, we need to first estimate the project effort if there was no reuse. The formula to determine the effort without reuse is:

$$\text{Expanded actual effort} = \text{Actual effort}/(1 - \text{reuse}\%) \quad (1)$$

For example, suppose that the actual effort for a project is 30 man-days and 70% of its source code comes from routines in a standard library. Then, if there is no standard library, the effort for developing this project should be 100 man-days.

We use the simplified formula (1) mainly because we are dealing with maintenance projects. From our experience, the percentage of code reuse corresponds to the percentage of effort savings to a high degree. This formula would not hold for new development projects.

Step 3: Determine the cost drivers–effort relationship. We then perform a linear regression analysis relating the expanded actual effort and the cost drivers. The linear regression analysis provides the weighting factor for each of the cost drivers. The result of this step is a normalized equation, with the general form:

$$\mathrm{ND(P)} = C_1 \times D_1 + C_2 \times D_2 + \cdots + C_k \times D_k$$

where ND(P) is the normalized effort value of project P,
C_i is the weighting factor of the ith cost driver,
D_i is the ith cost driver, and
k is the total number of cost drivers.

The method of least square is used to determine the best-fit line (BFL). As more and more project data are used in the above steps, the best fit line will shift toward the location where there is a high concentration of project points, and the weighting factors will change. It may seem that the more project data available, the better is the estimation. However, as the older projects may be developed under conditions and constraints that were very different from the current environment, these project data may not truly reflect the current practice. Thus, the inclusion of these project data may actually *corrupt* the estimation result! We believe that there is a fixed number of past project data that can provide the best estimation.

When using the AVN method for project estimation, we do not just use ND(P) as the project estimation. Instead, we apply two additional refinement steps to improve on the accuracy of the estimation.

Step 4: Determine the virtual neighbor. When we need to estimate the effort of a project P in a traditional rule-based system, the effort estimation is determined as follows: determine ND(P) from the normalized equation, and read off the corresponding effort estimation of P, E(P) in Figure 2. Each $\times$ in Figure 2 represents a project data point.

Instead, AVN uses two *nearest neighbors* of ND(P) as the basis of the estimation. The two nearest neighbors are determined as follows: calculate the absolute difference between point ND(P) and each of the normalized effort values of other projects in the database. The project points with the two smallest absolute differences are the two nearest neighbors of point ND(P). Let $\mathrm{ND(NN_1)}$ and $\mathrm{ND(NN_2)}$ be the two nearest neighbors of point ND(P).

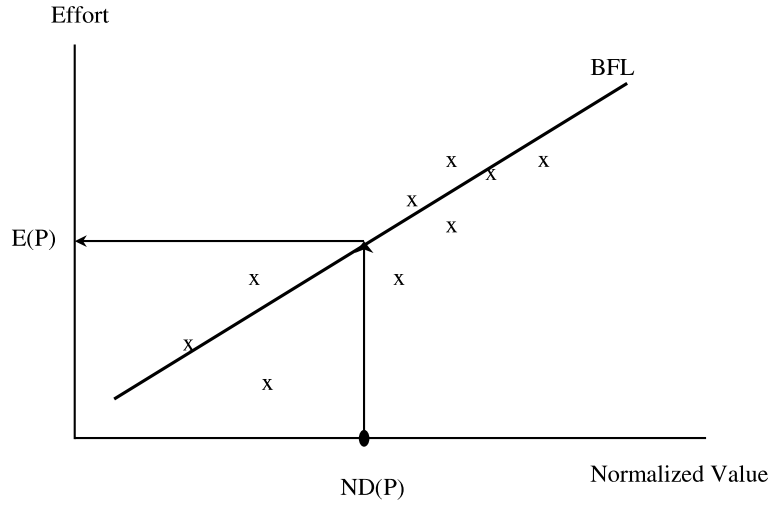

Figure 2. Traditional rule-based estimation method.

Next, we determine the *virtual neighbor* (VN) of P. When $ND(NN_1)$ and $ND(NN_2)$ are both larger than ND(P), or both smaller than ND(P), then VN is NN_1 if it is nearest to P, else VN is NN_2. Otherwise, the virtual neighbor is determined as shown in Figure 3. NN_1 and NN_2 are shown as dots in Figure 3. VN is the midpoint of the line joining NN_1 and NN_2. ND(VN) is determined by adding the values of $ND(NN_1)$ and $ND(NN_2)$ and dividing the sum by two.

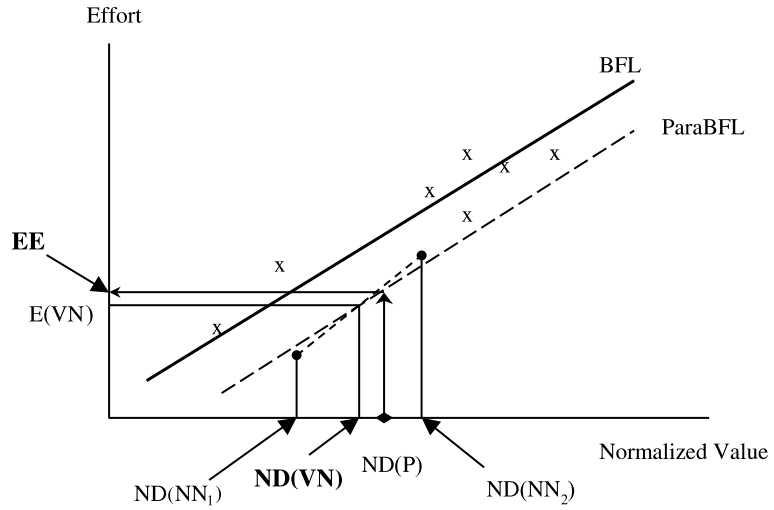

Figure 3. Estimation using virtual neighbor.

The reason for introducing the virtual neighbor is that it acquires an *averaging effect* and is superior than using a single neighbor (i.e. the closest neighbor). In other strategies that rely on a single neighbor (Shepperd and Schofield, 1997), if this neighbor happens to be a non-typical project, then the estimation will be biased and may not provide an accurate result.

Step 5: Proportional refinement. The next step is to adjust the estimation effort $E(\mathrm{VN})$ of the virtual neighbor by compensating for the relative location of ND(P) with respect to its two nearest neighbors. We do not just use the estimated effort of the virtual neighbor as the estimate of the effort for project P. We use the slope of the best-fit line (δ_{BFL}) to obtain the estimation effort EE. In Figure 3, ParaBFL is a line with the same slope as that of BFL, which meets VN.

EE is computed as follow:

$$\mathrm{EE} = E(\mathrm{VN}) + (\delta_{\mathrm{BFL}}) \times (\mathrm{ND(P)} - \mathrm{ND(VN)}) \tag{2}$$

where δ_{BFL} is the slope of BFL.

3. Validation of Model

This section highlights the empirical experiment using the AVN approach. First, we describe the characteristics of the project samples. Next, we present the results of applying AVN to four project estimations. Lastly, we compare the performance of the AVN method with three other estimation methods.

3.1. Organization and Data Collection

The project data come from the Hong Kong branch office of an international financial institution (Wong, 1999). The information technology department is divided into a front-end development team and a mainframe development team. The empirical data was collected from projects of the mainframe development team. This team has adopted the Waterfall model as its software engineering paradigm. As most of the computer systems were developed many years ago, most current projects are maintenance projects. This kind of environment is quite common in other financial institutions.

A Project Information System was developed in July 1997 for collecting the values of the cost drivers and the actual effort used for various projects. The data used for the study were collected over a period of 16 months. The company has been using the original COCOMO model for effort estimation for many years and relevant data were available for comparison.

A total of 101 project data were collected, of which 85 represented live projects. In these 85 live projects, 24 of them are selected for analysis because they have complete data for all the cost drivers and are maintenance projects. We will call this set the *experimental set*. The experimental set includes projects of different nature and size. In fact, they are typical projects of the team. The project data of the experimental set is listed according to the live date of the project in Table 1. Note that cost drivers for the COCOMO model are not listed.

Many projects in the experimental set were *enhancement projects*. For example, project A introduced a regular saving plan into the current mutual fund system. Originally, a customer could only buy a mutual fund by making a subscription order by phone. With the implementation of this project, a customer could debit his savings account or credit card account monthly and use the amount to buy a mutual fund in the mutual fund system. The subscription order transaction is automatically generated by the mutual fund system. Projects such as D, F, H, and N were also this kind of project but on different systems. *Corrective projects* are also included in the experimental set. For example, projects B and G were Year 2000 conversion projects, and project J involved changing a customer's code.

The experimental set is divided into two sets: a *testing* data set and a *validation* data set. The testing data set is the set of project data that will be used for finding the relationship between relevant cost drivers and project effort, and the validation data set is the set of project data that will be used for determining the accuracy of the AVN method. The first 20 projects will be used as the testing data set, and the last four projects will be used as the validation data set.

3.2. Project Effort Estimation

Our initial set of cost drivers includes a total of seven groups of factors. Some are suggested by previous studies (Gaffney and Werling, 1993), while others represent what can be collected in the organization. We do not include the skills and expertise of the development team, as there is a low turnover rate and all team members have been with the organization for several years. Table 2 lists the initial set of cost drivers.

Drivers such as *#input*, *#output*, *#inquiry*, *#update*, *#int-itf and #ext-itf* are further classified into simple, medium or complex, according to the normal practice of the organization. If the user requirement requires the program to handle one to nine objects, then it is a *simple* item. If it requires to handle 10 to 39 objects, then it is a *medium* item. If it requires to handle more than 40 objects, then it is a *complex* item.

By eliminating those cost drivers that always carry zero value, we identify 14 contributing cost drivers:

D_1 : Number of simple input, #input(s)
D_2 : Number of medium input, #input(m)
D_3 : Number of complex input, #input(c)

Table 1. Project data.

Project	#input(s)	#input(m)	#input(c)	#output(s)	#output(m)	#inquiry(s)	#inquiry(m)	#update(s)	#update(m)	#int-itf(s)	#int-itf(m)	#int-itf(c)	#ext-itf(s)	#ext-itf(m)	%reuse	Actual man days	Expanded actual effort
A	2	1	1	4	0	1	0	2	0	0	3	0	0	0	0	21	42
B	0	0	0	0	0	0	0	0	0	30	0	0	0	0	40	19	63
C	1	0	0	3	0	0	2	0	0	1	0	0	1	0	0	13	13
D	2	2	0	3	0	0	0	1	2	3	0	0	1	0	0	20	20
E	0	0	0	2	0	1	0	0	0	1	0	0	0	0	0	4	4
F	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	4	4
G	2	0	0	1	0	0	0	0	0	109	1	0	0	0	75	25	50
H	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	3	6
I	0	0	0	3	0	0	0	1	0	1	0	0	1	0	0	7	7
J	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
K	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1
L	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	3
M	0	0	0	1	0	0	0	0	0	13	1	0	0	9	50	21	42
N	0	0	0	0	1	0	0	0	0	2	0	0	0	0	0	4	4
O	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
P	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	13	13
Q	0	1	0	3	0	0	0	1	0	0	1	0	2	5	50	11	12
R	1	0	0	1	0	1	0	1	1	2	0	0	3	0	0	16	16
S	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2	2
T	0	0	0	0	2	0	0	0	0	0	0	0	0	0	20	5	6
U	17	4	0	2	0	0	0	0	1	7	1	0	0	0	20	25	31.25
V	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
W	1	0	0	2	0	0	1	1	0	2	0	0	1	0	0	8	8
X	0	0	0	0	2	0	0	0	2	1	4	0	0	0	0	17	17

Table 2. Initial set of cost drivers.

1	#input(s) #input(m) #input(c)	Number of input interface into the system (e.g. screen, parameter, and transaction types), items that enter the boundary of the system that cause processing to take place.
2	#output(s) #output(m) #output(c)	Number of output from the system (e.g. screen, report), items that leave the system boundary after processing has occurred.
3	#inquiry(s) #inquiry(m) #inquiry (c)	Number of inquiry of entity by program in the system (e.g. program call a routine to inquire the status of an account), unique inquiries that require an immediate response.
4	#update(s) #update(m) #update(c)	Number of update of entity by program in the system (e.g. program call a routine to update the status of an account), unique updating that updates the value of an entity immediately.
5	#int-itf(s) #int-itf(m) #int-itf(c)	Number of internal interface file formatting (e.g. make a report file), including files creation by the system for internal processing.
6	#ext-itf(s) #ext-itf(m) #ext-itf(c)	Number of external interface file formatting (e.g. file for other system), including files accessed by the system but not modified by it.
7	% reuse	Percentage of code reuse, this is an adjustment dimension used for adjusting the overall size of the project when all new codes are used.

Each item (1–6) can be classified into simple(s), medium(m) and complex(c).

D_4 : Number of simple output, #output(s)
D_5 : Number of medium output, #output(m)
D_6 : Number of simple inquiry, #inquiry(s)
D_7 : Number of medium inquiry, #inquiry(m)
D_8 : Number of simple update, #update(s)
D_9 : Number of medium update, #update(m)
D_{10} : Number of simple internal interface, #int-itf(s)
D_{11} : Number of medium internal interface, #int-itf(m)
D_{12} : Number of complex internal interface, #int-itf(c)
D_{13} : Number of simple external interface, #ext-itf(s)
D_{14} : Number of medium external interface. #ext-itf(m)

As there are 14 cost drivers, the minimum number of past data points for the linear regression analysis should be at least 14.

After expanding the actual effort and carrying out the linear regression analysis, we obtain the normalized equation:

$$\begin{aligned} \mathrm{ND(P)} = {} & 0.73D_1 + 0.93D_2 + 1.62D_3 + 0.99D_4 + 1.94D_5 \\ & + 1.29D_6 + 2.45D_7 + 1.51D_8 + 3.00D_9 + 0.97D_{10} \\ & + 2.23D_{11} + 1.69D_{12} + 0.42D_{13} + 2.25D_{14} \end{aligned} \tag{3}$$

For each project of the testing set, we substitute the cost drivers into Equation (3) to determine its normalized value.

We then plot the expanded actual effort against the normalized effort value using the testing data. The data points are lying on or near a straight line. Hence, we have great confidence that the expanded actual effort is directly proportional to the normalized value. The method of least square is used to calculate δ_{BFL}, which is 0.99.

Next, we present an example to illustrate the AVN method. Consider project U with the following raw data:

Actual effort = 25 man-days
Number of simple update = 17
Number of simple internal interface = 7
Number of medium input = 4
Number of medium update = 1
Number of simple output = 2
Number of medium interval interface = 1
Percent of reuse = 20%

The expanded actual effort is $25/(1 - 20\%) = 31.3$ man-days.
Substituting the above values into Equation (2), we obtain the normalized value for project U:

$$\begin{aligned}\mathrm{ND(U)} &= 17 \times 0.73 + 7 \times 0.97 + 4 \times 0.93 + 1 \times 3.00 + 1 \times 0.99 + 1 \times 2.23 \\ &= 30.1 \text{ man-days}\end{aligned}$$

Now, based on the ND(U) value, we locate its two nearest neighbors, which are ND(B) of project B and ND(M) of project M:

$$\mathrm{ND(B)} = 29.1 \text{ and } E(\mathrm{B}) = 29$$
$$\mathrm{ND(M)} = 36.1 \text{ and } E(\mathrm{M}) = 36$$

The virtual neighbor is the mid-point between ND(B) and ND(M). Thus,

$$\mathrm{ND(VN)} = (29.1 + 36.1)/2 = 32.6 \text{ and}$$
$$E(\mathrm{VN}) = (29 + 36)/2 = 32.5$$

Using these values and substituting into Equation 1, we obtain the estimated effort for project U:

$$\mathrm{EE} = 32.5 + 0.99 \times (30.1 - 32.5) = 30.1 \text{ man-days}$$

The relative error of this estimation is 3.7%.

3.3. Optimal Number of Past Projects

By repeating the estimation procedure for each of the validation data set using different numbers of past project data, we generate many project effort estimates. By

Table 3. Estimation error (%) of AVN using different number of past project data.

	Number of past project data sets used										
Project	23	22	21	20	19	18	17	16	15	14	Data set with the best est.
U	N/A	N/A	N/A	**3.7**	3.8	4.7	8.9	7.9	19.3	51.2	20
W	N/A	N/A	5.3	4.7	**4.5**	4.6	19.3	28.7	45.2	46.6	19
X	N/A	18.8	10.3	**10.2**	22.7	26.7	32.6	37	34.2	36.8	20
V	5.7	5.7	5.0	4.7	**3.9**	3.9	7.3	19.1	43.8	84.0	19
MARE	N/A	N/A	N/A	**5.8**	8.7	10.0	17.0	23.2	35.6	54.7	20

comparing these estimation results with the actual project effort, we can determine the number of past project data that will give the best estimation.

Various researchers have used different error measurements. A popular error measure is *Mean absolute relative error* (MARE) (Conte et al., 1986):

$$\text{MARE} = \sum_{i=1}^{n} (|(\text{estimate}_i - \text{actual}_i)/\text{actual}_i|)/n$$

where estimate_i is the estimated effort from the model, actual_i is the actual effort, and n is the number of projects.

Table 3 shows the result. For example, using 20 past project data, we obtain a relative error of 3.7% for project U. If we use 16 past project data, the relative error increases to 7.9%.

In Figure 4, we plot the error against the number of past project data used for estimation. It can be seen that the curve for Project X has a relatively larger error. Across all four projects, the relative error drops very quickly when the number of past project data used approaches 20.

From the results, it can be concluded that the number of past project data that should be used for optimal results is either 19 or 20, which provide a MARE of 6–9%. This is a very encouraging result.

3.4. Comparison with Other Approaches

To check the performance of AVN relative to other methods, we compare the results from AVN with those of three other estimation methods: The normalized dimension value (NDE) method, the closest neighbor (CN) method (Shepperd and Schofield, 1997), and the original COCOMO81 method (Boehm, 1981).

For the AVN estimation, the results based on 20 past projects are used for the comparison. The effort estimates from the COCOMO81 were retrieved from the company database. Table 4 summarizes the comparison.

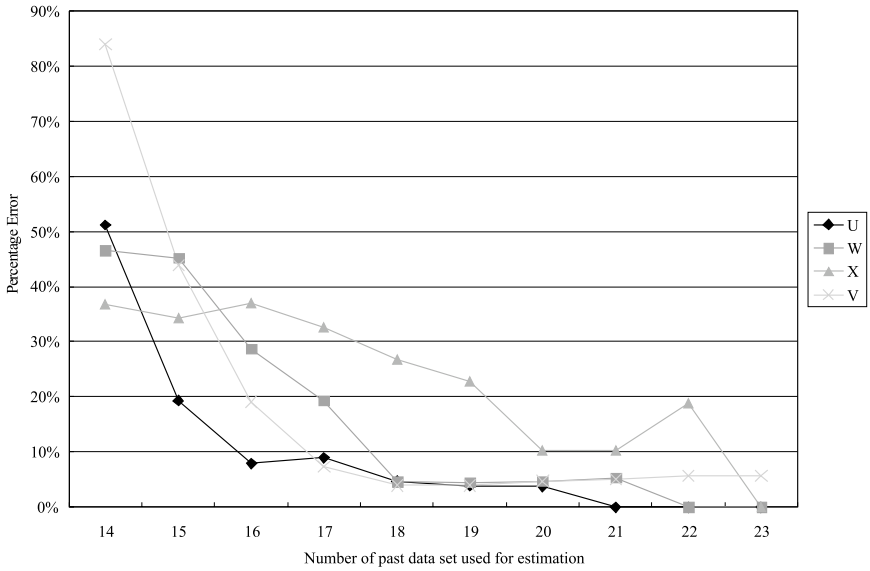

Figure 4. Estimation error relative to number of past project data.

Table 4. Estimation error (%) using different methods.

Project	AVN	NDE	CN	COCOMO81
U	3.7	3.3	4.0	11.4
W	4.7	133.0	100.0	62.5
X	10.2	15.0	17.5	26.1
V	4.7	11.9	11.1	31.4
MARE	5.8	40.8	33.2	32.9

It can be seen that NDE, CN and COCOMO81 give a much larger MARE than AVN. Although the AVN method does not always give the best estimation, it is a very consistent estimation method. The AVN method acquires a MARE of 6%, while the other methods fluctuated greatly from project to project. We credit the highly accurate results to the use of the virtual neighbor and the proportional refinement. Three additional factors may contribute to this encouraging result. They are a stable team, a stable development environment, and projects that are very similar in nature.

For project W, the three methods other than AVN perform poorly. This may be due to the small effort of project W (one day). The good result from AVN suggests that it also works well with small projects.

When we compare the result from AVN with that of NDE, NDE gives a slightly better estimation than AVN for project U only. This may be due to the fact that

NDE will not account for the localization effect of projects because it has no reference to its neighbor at all. NDE only uses the normalized value (ND) for estimation.

When we compare the result from AVN with that of CN, AVN outperforms CN for all four projects. Recall that as the CN method uses the concept of neighbor but only uses the closest neighbor, its estimation accuracy greatly depends on this single closest neighbor. If this single closest neighbor is a typical project, then the CN method will give a good estimation. However, if this closest neighbor is an abnormal project, then the CN method will give a poor estimation. On the other hand, the AVN will minimize the effect of choosing a non-typical neighbor because it uses the virtual neighbor. Unless the two closest neighbors are also non-typical points, the AVN will consistently give a better result.

When we compare the result from AVN with the COCOMO81, AVN always give a better estimation. This may be due to the obsolescence of the historical values used for the parameters of the COCOMO81. That is, the historical parameter values no longer reflect the current development environment of the organization. The use of COCOMO81 with outdated parameter values will reduce its accuracy.

It must be noted that the four models are based on slightly different project data. Also, if the COCOMO81 were calibrated to the same project database as the AVN method, its performance would likely improve.

4. Concluding Remarks

We introduced a new approach to estimate project effort. Based on the case study, the AVN method can provide highly accurate estimation for maintenance projects. Compared to other estimation methods, estimation by the AVN method yields a highly accurate result, with no more than 10% relative error.

It must be noted that the cost drivers and the developed model are specific to the environment of the studied organization. Others will have to carry out a similar study to determine the exact model for their particular environment. Also, the number and types of cost drivers will likely be different in another environment. Since the development environments between the mainframe and front-end teams are different in terms of tools and method, and all of the 24 data sets were collected from projects of the mainframe development team, the result reported here may not apply to the front-end development team of the organization. In order to use the result of this project, it is suggested that the front-end team carry out a similar study.

In the case study, historical data from 20 past projects is suggested to be used for the AVN. One may question the physical meaning of this *magic number* of 20. This number may be used as an indicator of the rate of changes in the development environment. If the number is large, it means that old historical project data can also be used for estimation. That is, the development environment of the organization is very stable. This occurs when there are no great technology changes in the organi-

zation and the capability of the developers is very consistent. If the number is small, it may indicate that the development environment is a volatile one and the capability of the developers has changed.

It must be noted that the number of projects used for best estimation depends on the current project requiring estimation, and the distribution of the previous projects relative to the new estimation. In general, we cannot rely on using the same number of projects for all estimation. A better result may be obtained from using several sets of recent projects. In any case, this number of 20 projects is site specific and others will have to determine the best number for their own environment.

In our study, 14 relevant cost-drivers are found to be necessary for the estimation. Some of the proposed dimensions are deleted because of lack of data. It is not suggested that those dimensions are not essential to the process of estimation. If the data collection phase is longer, more dimensions may be collected for experimentation. Future work can be done to enlarge the size of the sample data. Also, we will experiment with other dimensions.

In this case study, a linear relationship between the cost drivers and the expanded actual effort has been identified. For other environments with different sets of cost drivers, the relationship may not be linear. This is another reason why other companies will need to develop their own model.

When applying AVN to another environment or non-maintenance projects, we recommend the following steps:

(1) Identify available cost drivers: It is important to collect as many cost drivers as possible. Cost drivers from the COCOMO can serve as a starting set. Also, staff may be able to identify additional cost drivers that are relevant to the specific organization and applications. For those cost drivers that have a low impact on the effort relative to others, they may be removed. Also, cost drivers with many missing data may be discarded. Collinearity between some drivers can be removed by factor analysis.

(2) Determine the expanded actual effort: If the organization reuses design or code, then the effort data should be adjusted to reflect this. Some investigation may be needed to identify the relationship between expanded effort and reuse%.

(3) Determine the cost driver–effort relationship: The organization can first apply linear regression analysis to relate cost drivers to effort. Certain cost drivers may be discarded if their influence on effort is small. If the linear regression analysis does not provide a good model, then other non-linear models may be attempted.

(4) Determine the virtual neighbor and apply proportional refinement: The AVN method will perform best when the new project is similar to those in the historical database.

We plan to investigate whether effort for new development projects can be estimated using the above procedure.

Notes

1. SCU, http://sunset.usc.edu/research/COCOMOII/cocomo_main.html

References

Aron, J. D. 1969. *Estimating Resource for Large Programming Systems*. Rome, Italy: NATO Science Committee.

Banker, R. H., Chang, H., and Kemerer, C. 1994. Evidence on economies of scale in software development. *Information and Software Technology* 36(5): 275–282.

Black, R. K. D., Curnow, R. P., Katz, R., and Gray, M. D. 1977. *BCS Software Production Data*, Final Technical Report, RADC-TR-77-116, Boeing Computer Services, Inc.

Boehm, B. 1981. *Software Engineering Economics*. Englewood Cliff, NJ: Prentice-Hall.

Boehm, B., Clark, B., Horowitz, E., and Westland, C. 1995. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering* 1: 57–94.

Briand, L., Langley, T., and Wieczorek, I. 2000. A replicated assessment and comparison of common software cost modeling techniques. In: *Proceedings of the International Conference on Software Engineering*, 377–386.

Conte, E., Dunsmore, H. E., and Shen, V. Y. 1986. *Software Engineering Metrics and Models*. Menlo Park: Benjamin Cummings.

Donelson, W. S. 1976. Project planning and control, *Datamation*, (June): 73–80.

Gaffney, Jr. J. E., and Werling, R. 1993. Estimating software size from counts of externals, a generalization of function points. In: eds Gulledge, T. R. and Hutzler, W. P. *Analytical Methods in Software Engineering Economics*, Berlin: Springer-Verlag.

Gray, A. R., and MacDonell S. G. 1997. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology* 39(6): 425–437.

Kemerer, C. F., 1987. An empirical validation of software cost estimation models. *Communication of the ACM* 30(5): 436–445.

Mukhopadhyay, T., Vicinanza, S. S., and Prietula, M. J. 1992. Examining the feasibility of a case based reasoning model for software effort estimation, *MIS Quarterly* 16: 156–173.

Parkinson, G. N. 1957. *Parkinson's Law and Other Studies in Administration*. Boston: Houghton-Miffin.

Putnam, L.H. 1978. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering* SE-4(4) (July): 345–361.

Shepperd, M., and Schofield, C. 1997. Estimating software project effort using analogies, *IEEE Transactions on Software Engineering* 23(12): 736–743.

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

Thayer, R. H., and McGettrick, A. D. 1993. *Software Engineering, A European Perspective*. Los Alamitos, CA: IEEE Computer Society Press.

Walston, C. E., and Felix, C. P. 1977. A method of programming measurement and estimation, *IBM Systems Journal* 16(1): 54–73.

Wittig, G. E., and Finnie, G. R. 1994, Using artificial neural networks and function points to estimate 4GL software development effort. *Australian Journal of Information Systems* 1(2) 341–358.

Wolverton, R. W. 1974. The cost of developing large-scale software, *IEEE Transactions on Computer* c-23(6): 615–636.

Wong, C. M. 1999. Dimensional analysis for estimating software project effort using analogies. MSc in Information Technology Dissertation, Hong Kong Polytechnic University.

Dr. Hareton Kam Nang Leung is an associate professor of the Department of Computing, The Hong Kong Polytechnic University. He currently serves as the Director of the *Laboratory for Software Development and Management*, which focuses on software process, quality improvement, and other applied software technology research.

Dr. Leung has conducted research in software testing, software maintenance, quality and process improvement, and software metrics. A member of IEEE, IEEE Computer Society, and Hong Kong Computer Society, Dr. Leung currently serves on the Editorial Board of *Software Quality Journal* and the Program Committee of many international and local conferences on software quality, software metrics, and maintenance.

Prior to joining HK PolyU, Dr. Leung held team leader positions at BNR, Nortel, and GeneralSoft Ltd. His consultancy clients include MPFA, VTech, Hong Kong Productivity Council and the Industry Department of Hong Kong.

Dr. Leung strongly believes that process and quality improvement will help a company to compete in the global market. In 1995, he co-founded the *Hong Kong Software Process Improvement Network (HKSPIN)* to promote software process improvement and quality improvement in HK. He was the Program Co-Chair of the *First Hong Kong Conference on Quality Software Development* in 1996, and the Industry Chair of the *Second Hong Kong Conference on Software Development* in 1997. He has been involved in several software process improvement and ISO 9000 projects in Hong Kong. He has played a major role in the design and development of HKSQA – a software quality assurance model for Hong Kong.