SOFTWARE DEVELOPMENT PRODUCTIVITY ATTRIBUTES: AN EXPLORATORY STUDY

Abbas Heiat, Montana State University-Billings, College of Business, 1500 University Drive, Billings, MT, USA, (406) 657-1627, aheiat@msubillings.edu

ABSTRACT

In this paper I used regression analysis and artificial neural network to predict productivity of the software development. The results obtained by ANN were superior to that of regression analysis. Clearly Multilayer Perceptron was an effective network in this case and efficiently predicted the productivity in terms of independent variables. The results of experimentation with ANN clearly indicated the nonlinear relationship between the productivity of software development and our independent variables.

INTRODUCTION

Software development productivity is an important project management concern. One study reports that a 20% improvement in software productivity will be worth $45 billion in the U.S and $90 billion worldwide [1]. As a result, a number of empirical studies of software productivity have appeared in the literature over the past three decades. Scacchi has published a report that examines empirical investigations in relation to software development attributes, tools, techniques, or some combination of these that have a significant impact on productivity of software production [2]. These studies focus on the development of large scale software development. Twelve major software productivity measurement studies are reviewed including those at IBM (Albrecht [3,4]), TRW (Boehm [1,5,6, 7]), NASA (Bailey and Basili [8]), ITT (Vosburg et.al [9]) , and international projects (Lawrence [10], Cusumano and Kemerer [11]). In addition, Scacchi examines a number of other theoretical and empirical studies of programmer productivity, cost-benefit analysis, and estimation of software cost(Thadhani [12] ,Lambert [13], Cerveny and Joseph[14]).

Based on his survey, Scacchi identifies a number of software productivity attributes:

- Computing resources and easy-to-access to support system specialists
- Contemporary software engineering tools and techniques
- System development aids for coordinating software projects
- Programming languages
- Software project Complexity. indicated by size of source code delivered, functional coupling, and functional cohesion
- Reuse software that supports the information processing tasks required by the application
- Stable system requirements and specifications
- Small, well-organized project teams
- Experienced software development staff

However, Scacchi believes that it is not always possible or desirable to improve software productivity by cultivating the entire project characteristics listed above.

Banker et.al, applying regression and data envelopment analysis to the data collected at large regional bank conclude that significant factors improving productivity included project team capability and good system response time. Factors that significantly but negatively affected productivity included lack of

team application experience and high project staff loading. The use of a new structured analysis and design methodology also resulted in productivity in the short term [15].

PRODUCTIVITY MEASURE OF SOFTWARE DEVELOPMENT

In general, productivity is understood as a ratio of outputs produced to inputs used. However, researchers may use different outputs and inputs for measuring productivity. IEEE Standard 1045 calculates productivity in terms of effort as an input and lines of code or function points as output [16]. The two most common methods for measuring complexity or size of a software development project are Function Points and Lines of Code.

The main limitation of the LOC model is that it depends on the accuracy of an early estimate of lines of code. This estimate is usually based on the past experience of the systems analyst. Certainly, most organizations would find it difficult, if not impossible, to locate experienced analysts who could come up with an accurate estimate of the system size using a LOC model [17]. A third problem with LOC model is that it does not take into account the resources available to the systems development team. These include among other things the types of language used in coding, software tools, the skills and experiences of the team itself [18].

An alternative method for estimating systems development effort was developed by Albrecht [3]. Albrecht introduced the concept of Function Points (FP) to measure the functional requirements of a proposed system. In FP modeling the size of the system is determined by first identifying the type of each required function in terms of inputs, outputs, inquiries, internal files, and external interface files. To calculate the value of function points for each category, the number of functions in each category is multiplied by the appropriate complexity weight. The total systems effort is then calculated by multiplying the sum of function points for all categories by the Technical Complexity Factor (TCF). The TCF is determined by assigning values to 14 influencing project factors and totaling them. Readers unfamiliar with the FP model are referred to Albrecht and Gaffney [4]. Albrecht argued that FP model makes intuitive sense to users and it would be easier for project managers to estimate the required systems effort based on either the user requirements specification or logical design specification [3]. Another advantage of the FP model is that it does not depend on a particular language. Therefore, project managers using the FP model would avoid the difficulties involved in adjusting the LOC counts for information systems developed in different languages. In this paper I have used the following equation for calculation of productivity: Productivity = Effort/Function Points

METHODOLOGY

I used regression analysis and artificial neural network to predict productivity of the software development in terms of variables explained in Data Acquisition and Pre-Processing section of this paper.

In the last two decades, Artificial Neural Networks have been used for predictions in diverse applications. In general, results have been superior to conventional methods [19]. In recent years, a number of studies have used neural networks in various stages of software development. Hakkarainen et al, estimated software size by training an ANN. They used structured specification descriptions as input and Demarco Function Bang, Albrecht's Function Points and Symon's mark II Function Points size metrics as output. The results of their study indicated that ANN could be used successfully to estimate software size [20]. Srinivasan and Fisher compared two approaches 1) a back propagation neural network and 2) Regression Trees, using Boehm's historical database. Their experiments indicated that neural network and regression trees are competitive with model-based approaches [21]. Finnie and Wittig applied artificial neural networks (ANN) and case-based reasoning (CBR) to estimate software

development effort [22]. They used a data set from the Australian Software Metrics Association. ANN was able to estimate software development effort within 25% of the actual effort in more than 75% of the cases, and with a MAPE of less than 25%. Carolyn Mair et al, used 67 software projects derived from a Canadian software house to evaluate prediction performances of regression, Rule Induction (RI), CBR and ANN techniques [23]. The results of the study showed considerable variation between the 4 models. MAPE for RI ranged from 86% to140%. MAPE for regression ranged from 38% to 100%. MAPE for CBR ranged from 43% to 80% and for ANN ranged from 21% to 66%. MAPE results suggest that ANN seem to be the most accurate and RI is the least accurate technique [23]. Shukla conducted a large number of simulation experiments using genetically trained neural networks. He used a merged database comprising 63 projects, and Kemerer database comprising 15 projects. The results indicated a significant estimation improvement over Quick Propagation Network and Regression Trees approaches. Shukla concluded that there is still a need to apply neural networks to diverse projects with wide range of attributes because it is "unclear which techniques are most valuable for a given problem. …, experimental comparison using rigorous evaluation methods is necessary" [24].

The MultiLayer Perceptron (MLP) is one of the most widely implemented neural network topologies. In terms of mapping abilities, the MLP is believed to be capable of approximating arbitrary functions. This has been important in the study of nonlinear dynamics, and other function mapping problems. MLPs are normally trained with the back propagation algorithm. Two important characteristics of the MultiLayer Perceptron are:

1. Its smooth nonlinear Processing Elements (PEs). The logistic function and the hyperbolic tangent are the most widely used. Their massive interconnectivity i.e. any element of a given layer feeds all the elements of the next layer.
2. The Multilayer Perceptron is trained with error correction learning, which means that the desired response for the system must be known. Back propagation computes the sensitivity of a cost function with respect to each weight in the network, and updates each weight proportional to the sensitivity.

The Radial Basis Function (RBF) network is a popular alternative to the MLP which can offer advantages over the MLP in some applications. An RBF network can be easier to train than an MLP network. The RBF network has a similar form to the MLP in that it is a multi-layer, feed-forward network. However, unlike the MLP, the hidden units in the RBF are different from the units in the input and output layers. They contain the Radial Basis Function, a statistical transformation based on a Gaussian distribution from which the neural network's name is derived. Like MLP neural networks, RBF networks are suited to applications such as pattern discrimination and classification, pattern recognition, interpolation, prediction and forecasting. In the hidden layer of an RBF, each hidden unit takes as its input all the outputs of the input layer *xi*. The hidden unit contains a basis function which has the parameters centre and width. The centre of the basis function is a vector of numbers, ***ci*,** of the same size as the inputs to the unit and there is normally a different centre for each unit in the neural network. The first computation performed by the unit is to compute the radial distance, *d*, between the input vector xi and the centre of the basis function, typically using Euclidean distance:

$d = SQRT ((x^1 - c^1)^2 + (x^2 - c^2)^2 + ... (x^n - c^n)^2)$

The unit output, ***a*,** is then computed by applying the basis function *B* to this distance divided by the width *w*: $a = B(d/w)$

The basis function is a curve, typically a Gaussian function, which has a peak at zero distance and which falls to smaller values as the distance from the centre increases. As a result, the unit gives an output of one when the input is centered but which reduces as the input becomes more distant from the centre. The output layer of an RBF neural network is essentially the same as for the MLP. Normally it has a linear activation function, making it possible to calculate the weights for those units directly. However, if the output units have non-linear activation functions, then iterative training algorithms must be used [25].

DATA ACQUISITION

The data used in this research project was collected by The International Software Benchmarking Standards Group (ISBSG). The group gathered information from 1238 software projects from around the world. Projects cover a broad cross-section of the software industry. In general, they have a business focus. The projects come from 20 different countries. Major contributors are the United States (27%), Australia (25%), Canada (11%), United Kingdom (10%), Netherlands (7%), and France (7%). Major organization types are insurance (19%), government (12%), banking (12%), business services (10%), manufacturing (10%), communications (7%), and utilities (6%).

Projects types include enhancement projects (50%), new developments (46%), and 4% are re-developments. Application types consist of Management Information Systems (38%), transaction/production systems (36%), and Office Information Systems (5%). Nearly 3% are real-time systems.

Over 70 programming languages are represented. 3GLs represent 57% of projects, 4GLs 37%, and application generators 6%. Major languages are COBOL (18%), C/C++ (10%), Visual Basic (8%), Cobol II (8%), SQL (8%), Natural (7%), Oracle (7%), PL/I (6%), Access (3%), and Telon (3%). Platform for projects include mainframe projects (54%), midrange (24%), and microcomputers (22%).

Sixty-two (62%) of projects use a standard methodology that was developed in-house), 21% use a purchased methodology. Only 12% do not follow a methodology. The use of CASE tools ranges from 21% of projects using upper CASE, down to 10% for integrated CASE tools. CASE tools of some type are used in 51% of projects. Traditional system modelling techniques (data modelling, process modelling, event modelling, business area modelling) are used in 66% of projects. They are the only techniques listed in 27% of projects; 39% use a combination of traditional modelling and other techniques. The most common single technique is data modelling, used in 59% of projects. RAD/JAD techniques are used in 28% of projects. Object oriented techniques are used in 14% of projects. Prototyping is used in 29% of projects.

Data in the ISBSG database had to be cleaned and pre-processed in order to get, relevant and complete data for analysis. Records with missing value of attributes were excluded and the character values of text attributes or variables were transformed to numeric values. Function points count, total work effort in hours, team size, development platform (mainframe, mid-size, PC), language type (3GL, 4GL, Application Generator etc.), whether a software development methodology was used, programming language and development type (new, enhancement, etc.) attributes were considered for analysis. Productivity attribute was calculated by dividing total work effort in hours by count of function points. Once the data was pre-processed, 468 usable projects were available for analysis. The following Figure represents a 3-D graph of data set used in this research.

DATA MODELING

As mentioned in section 2, Productivity which is calculated by dividing Effort by Function Points is used as dependent variable in regression and artificial neural network model. Function Points, Team Size, Development Platform, Language Type, Methodology, Development Type, Application Type, and Programming Language are used as independent variables.

Regression analysis was performed by this author using Statistica, a powerful statistical and data mining software. Equation is estimated by ordinary least square estimators using 468 observations. Regression estimation results are summarized in Table 1 and 2. Overall, the regression results are poor, producing relatively very low value of coefficient of determination (R^{2}). The critical values for Durbin-Watson

(D.W.) test at 5% level of significance are only significant for Development Platform(**5.57311),** Language Type**(-2.98052),** and Methodology(**2.40586).**

The regression results show an inverse relationship between productivity and function points, language type and programming language. This indicates the smaller or less complex projects are more productive and using fourth generational programming languages (4GLs) contribute more to productivity as compared with 3GLs and 2GLs. Furthermore, using a systematic methodology and PC-based and mid-range servers is positively correlated to productivity. In addition, the results demonstrate that a linear model is not a good approach for analyzing the relationship between dependent and the independent variables and we need to use a nonlinear model. However, in the absence of a specified nonlinear model and a theoretical foundation for it, neural network may be an appropriate tool to study this relationship.

Table 1

	Summary Statistics; DV: Productivity (Stan
Statistic	Value
Multiple R	0.356360
Multiple R²	0.126992
Adjusted R²	0.111777
F(8,459)	8.346083
p	0.000000
Std.Err. of Estimate	0.942456

Table 2

	Beta	**Std Err**	**B**	**Std Err**	**t**	**p**
Intercept			0.000000	0.043565	0.00000	1.000000
Function Points	-0.053353	0.044621	-0.053353	0.044621	-1.19568	0.232438
Team Size	0.032032	0.044327	0.032032	0.044327	0.72264	0.470268
Development Platform	0.255579	0.045859	0.255579	0.045859	**5.57311**	0.000000
Language Type	-0.137447	0.046115	-0.137447	0.046115	**-2.98052**	0.003030
Methodology	0.107331	0.044612	0.107331	0.044612	**2.40586**	0.016529
Development Type	0.055846	0.044948	0.055846	0.044948	1.24244	0.214708
Application Type	0.042246	0.045364	0.042246	0.045364	0.93128	0.352197
Programming Language	-0.070772	0.046206	-0.070772	0.046206	-1.53165	0.126299

ARTIFICIAL NEURAL NETWORK

In Statistica , Automated Neural Networks(ANNs) is a comprehensive and extremely fast neural network data analysis tool. It covers integrated pre- and post-processing, including data selection,

nominal-value encoding, scaling, normalization and missing value substitution, with interpretation for classification, regression and time series problems. a unique wizard-style Intelligent Problem Solver can walk you step-by-step through the procedure of creating a variety of different networks and choosing the network with the best performance. ANNs supports and automatically selects the most important classes of neural networks for the used data. These include

- Multilayer Perceptrons (Feedforward networks)
- Radial Basis Function networks
- Kohonen Self-Organizing Feature Maps
- Probabilistic (Bayesian) Neural Networks
- Generalized Regression Neural Networks
- Linear modeling.

Automated Neural Networks is used for this study. The size of error can be used to determine how well the network output fits the desired output. The simulation runs terminates when the error drops below the specified threshold. The specified threshold for experiments were set to 0.01 which means that the simulations for finding the best network configuration would stop when average error or the difference between actual effort and estimated effort for the training set drops below 1%.

The size of the training set is of fundamental importance to the practical usefulness of the network. If the training patterns do not convey all the characteristics of the problem class, the mapping discovered during training only applies to the training set. Thus the performance in the test set will be much worse than the training set performance. Another aspect of proper training is related to the relation between training set size and number of weights in the ANN. If the number of training examples is smaller than the number of weights, one can expect that the network may "hard code" the solution, i.e. it may allocate one weight to each training example. This will obviously produce poor generalization. Statistica recommends that the number of training examples be at least double the number of network weights. When there is a big discrepancy between the performance in the training set and test set, we can suspect deficient learning. Note that one can always expect a drop in performance from the training set to the test set. At our present stage of knowledge, establishing the size of a network is more efficiently done through experimentation. The issue is the following: The number of PEs in the hidden layer is associated with the mapping ability of the network. The larger the number, the more powerful the network is. However, if one continues to increase the network size, there is a point where the generalization gets worse. This is due to the fact that we may be over-fitting the training set, so when the network works with patterns that it has never seen before the response is unpredictable. The problem is to find the smallest number of degrees of freedom that achieves the required performance in the test set. One school of thought recommends starting with small networks and increasing their size until the performance in the test set is appropriate. Some researchers propose a method of growing neural topologies that ensures a minimal number of weights, but the training can be fairly long. An alternate approach is to start with a larger network, and remove some of the weights. There are a few techniques, such as weight decay, that partially automate this idea. The weights are allowed to decay from iteration to iteration, so if a weight is small; its value will tend to zero and can be eliminated by the development life cycle. In this study I started with one hidden layer and then increased the size of the network to two and three hidden layers. The weight decay approach was used for all networks.

The results of using ANNs to our data are summarized in Table 3. ANNs displays the best networks with the least training error. In our case, the Radial Basis Function network has a test error of 2.06 while Multilayer Perceptrons network has a test error of 0.001. Clearly Multilayer Perceptrons is a more effective network in this case and predicts the productivity in terms of independent variables efficiently. The results of experimentation with ANN clearly indicate the nonlinear relationship between the productivity of software development and our independent variables.

Table 3

Summary of active networks (Standardized data.sta)									
Index	**Net. name**	**Training perf.**	**Test perf.**	**Training error**	**Test error**	**Training algorithm**	**Error function**	**Hidden activation**	**Output activation**
1	RBF 8-45-1	0.012277	0.499071	4.639897E+37		RBFT	SOS	Gaussian	Identity
2	RBF 8-55-1	-0.031253	0.475194	6.215769E+36	1.032425E+23	RBFT	SOS	Gaussian	Identity
3	RBF 8-55-1	-0.002569	0.483851	8.893235E+29	5.681891E+15	RBFT	SOS	Gaussian	Identity
4	RBF 8-56-1	0.215150	0.473080	1.322352E+26	3.680124E+24	RBFT	SOS	Gaussian	Identity
5	RBF 8-60-1	-0.074636	0.475570	2.339567E+35	4.969803E+17	RBFT	SOS	Gaussian	Identity
Summary of active networks (Standardized data.sta)									
Index	**Net. name**	**Training perf.**	**Test perf.**	**Training error**	**Test error**	**Training algorithm**	**Error function**	**Hidden activation**	**Output activation**
6	RBF 8-47-1	0.113908	0.448754	1.195883E+16	2.555914E+14	RBFT	SOS	Gaussian	Identity
7	MLP 8-9-1	0.416223	0.264404	8.567703E-03	9.972790E-03	BFGS 14	SOS	Exponential	Logistic
8	MLP 8-7-1	0.391854	0.251015	8.769141E-03	9.907189E-03	BFGS 22	SOS	Logistic	Identity
9	MLP 8-5-1	0.434150	0.268484	8.396658E-03	9.985879E-03	BFGS 22	SOS	Exponential	Tanh
10	MLP 8-5-1	0.381583	0.251352	8.840906E-03	9.987633E-03	BFGS 8	SOS	Exponential	Tanh

A full set of references are available upon request.