Efficiency in Software Development Projects

Aneesh Chinubhai

Dharmshines Desai University
aneeshchinubhai@gmail.com

Abstract

A number of different factors are thought to influence the efficiency of the software development process. These include programming languages, use of formal methodologies, CASE tools, etc. Moreover, efficiency in the context of software development has traditionally been measured as the ratio of functionality, either lines of code or function points, and the effort expended. This is a unidimensional measure and ignores factors such as quality and elapsed time. This study utilizes Data Envelopment Analysis (DEA) to develop a multidimensional measure of efficiency in the context of software development and then examines how efficiency varies with other influencing factors such as the use of CASE tools, use of methodologies, team size, application type, etc. using a decision tree based model.

Keywords: Efficiency, DEA, CASE Tools, Formal Methodology, Team Size

1. Introduction

A number of different factors are thought to influence software development productivity. It is well known that various programming languages have evolved over the years and this progression through various generations of languages (2GL, 3GL, 4GL, etc.) have led to an increase in productivity of development. Studies have been carried out examining the lines of code required per function point for various languages. This, however, has been a unidimensional view of productivity.

Other factors such as the use of modern development tools such as those utilized for Computer Aided Software Engineering (CASE), the use of a formal development methodology, development type (new development, re-development, etc.) the number of people on the development team, etc. also influence productivity.

Premraj and others [17] analyzed around 600 projects from Finland over a period from 1978 to 2003. The analysis was based on the traditional definition of productivity. The authors acknowledged that it is easier to be productive if quality is disregarded. Inclusion of a quality metric would lead to a better measure of productivity. While there is a large sample size, all the projects analyzed are from Finland.

Another study [3] examined the relationship between project complexity, team size and productivity. They defined productivity as function points per man month. Their analysis used a log-linear regression model. They found a strong negative relationship between team size and productivity. However, they state that a variable indicating quality is missing from the model.

Another study [4] of 98 European firms found a negative correlation between team size and productivity. Analysis of 79 software development projects from the European Space Agency also found a negative relationship between team size and productivity [15].

In other studies, Case [6] and Kriebel [14] conclude that a high level of productivity requires quality to be sacrificed and vice versa. However, they could not examine in detail
some of the other factors and their possible interplay with the limited amount of data available to them.

All of the studies mentioned above use the traditional function points per man-hour or equivalent definition of productivity. They also do not consider a number of other contributing factors such as the use of methodologies, use of CASE tools, the type of application being developed, etc. Moreover, the data sets are either limited in size or quite dated by today’s standards. The analysis conducted in this paper considers that there are multiple outputs (functionality, product defects and elapsed time) as a result of the software development process which consumes development effort as an input.

The objective of this analysis, therefore, is to measure efficiency using multiple measures of output and then to examine how efficiency varies with the factors such as team size, use of CASE tools, use of formal methodologies, type of project, etc.

The approach used in this study will first involve the measurement of efficiency scores using a Data Envelopment Analysis (DEA) model. The DEA scores are subsequently analyzed using a decision tree model in order to examine the relationship between efficiency and other contributing factors mentioned above.

2. Data Envelopment Analysis

Data Envelopment Analysis (DEA), developed by Charnes, Cooper and Rhodes (CCR) [8], is a non-parametric technique used for efficiency assessment. They refer to each unit that is analyzed as a Decision Making Unit (DMU). DEA uses linear programming to develop an efficient frontier that “envelops” the set of DMUs under consideration.

Traditionally, efficiency has been measured as a ratio.

\[
Efficiency = \frac{Output}{Input}
\]  

(1)

When multiple inputs and outputs are considered, appropriate weights are assigned to each input and output. However, these weights may not be appropriate across the board. Therefore, DEA attempts to portray each DMU in the best possible light by using different weights for each input and output. The weights are determined by solving a series of linear programs – one for each DMU. The formulation is as follows:

\[
\begin{align*}
\min & \quad \Theta \\
\text{s.t.} & \quad \sum_{i} \lambda_{i} X_{i} \leq \Theta X_{0} \\
& \quad \sum_{i} \lambda_{i} Y_{i} \geq Y_{0} \\
& \quad \lambda_{i} \geq 0
\end{align*}
\]  

(2-5)

In the formulation above, known as the envelopment form, \( \lambda_{i} \) represents the weight that is given to DMU \( i \) in its effort to dominate DMU \( 0 \), \( \Theta \) is the efficiency, \( X_{i} \) are the inputs and \( Y_{i} \) are the outputs. An alternative formulation, known as the multiplier form is as follows:

\[
\begin{align*}
\max & \quad \sum_{r} u_{r} y_{r} \\
\text{s.t.} & \quad \sum_{r} u_{r} y_{rj} - \sum_{i} v_{i} x_{ij} \leq 0
\end{align*}
\]  

(6-7)
In the formulation above, $u_r$ and $v_r$ represent the output and input weights respectively, $x_r$ and $y_r$ represent the inputs and the outputs. The two formulations shown above are dual linear programs.

The CCR model discussed above assumes constant returns to scale (CRS). Banker, Charnes and Cooper [2] modified the CCR models to account for variable returns to scale. This model is known as the Banker-Charnes-Cooper (BCC) model. The CCR model (constant returns to scale) assumes that the DMU under consideration is operating at the optimal scale. The BCC model, on the other hand, allows DMUs to be classified as efficient even if they are not operating at the optimal scale size.

The BCC model addresses variable returns to scale by enveloping the production possibility set with a convex hull as opposed to a conical hull in the CCR model resulting in a piecewise linear efficient frontier. The following convexity constraint is added to the envelopment formulation:

$$\sum_{i=1}^{N} \lambda_i = 1$$  \hspace{1cm} (11)

The formulation is shown using equations 12-16.

$$\min \quad \Theta$$  \hspace{1cm} (12)

$$s.t. \quad \sum \lambda_i X_i \leq \Theta X_0$$  \hspace{1cm} (13)

$$\sum \lambda_i Y_i \geq Y_0$$  \hspace{1cm} (14)

$$\sum \lambda_i = 1$$  \hspace{1cm} (15)

$$\lambda_i \geq 0$$  \hspace{1cm} (16)

The multiplier version of the formulation is shown below.

$$\max \quad \sum u Y_i - u_0$$  \hspace{1cm} (17)

$$s.t. \quad \sum u Y - \sum v X - u_0 \leq 0$$  \hspace{1cm} (18)

$$\sum v X_0 = 1$$  \hspace{1cm} (19)

$$u \geq \varepsilon$$  \hspace{1cm} (20)

$$v \geq \varepsilon$$  \hspace{1cm} (21)

The variable $u_0$ is free in sign and is associated with the constraint $\sum \lambda_i = 1$ in the envelopment formulation. The sign of the variable $u_0$ also indicates the nature of returns to scale for the DMU being evaluated as follows:

- $u_0 < 0$ indicates increasing returns to scale.
- $u_0 > 0$ indicates decreasing returns to scale.
- $u_0 = 0$ indicates constant returns to scale.
DEA scores are restricted to a scale of 0 to 1. This may lead to a situation where there are a number of DMUs which have a score of 1. In case ordinary least squares regression (OLS) models are used for post-DEA analysis of scores, they may predict a value greater than one since one cannot restrict OLS models to a scale of 0 to 1. In order to overcome this problem of DEA scores being restricted to a scale of 0 to 1, a DEA model called the Super-efficiency model has been developed by Andersen and Petersen [1]. The superefficiency models yield DEA scores that are not restricted to being less than 1. This linear programming formulation for the superefficiency model is the same but it excludes data for the DMU under consideration from the reference set. Superefficiencies result from a DEA model where DMUs can obtain efficiencies greater than 1 because each DMU is not allowed to use itself as a peer.

A number of books give a detailed exposition of DEA and its applications ([7], [10], [16], [18] and [19]).

3. Factors Influencing Efficiency in Software Projects

3.1 Sources of Data

Empirical data on software development projects are obtained from a repository [11] maintained by the International Software Benchmarking Systems Group (ISBSG). The repository contains data on more than 4,000 projects with over 100 attributes on each project. However, a significant number of projects included in the database have missing attribute data. This limits the number of usable data in the repository.

3.2 Estimating Efficiencies: The DEA Model

Since software development is still largely a labor driven process, the DEA model was developed using Work Effort measured in man hours as the sole input. Three parameters of product development performance [9] were chosen as outputs: product functionality measured as the number of function points developed, quality measured as the number of defects per function point and lead time measured as the elapsed time per function point from project start to finish. This is shown in figure 1.

**Figure 1: DEA Model: Inputs and Outputs**

Quality per function point and elapsed time per function point are undesirable outputs and need to be minimized while maximizing the number of function points delivered. The undesirable outputs are transformed using the additive inverse approach suggested by Koopmans [13] where an undesirable output $Q$ is transformed using the additive inverse $f(Q)$.
\[ Q = -Q \] and included directly in the model as an output. The BCC Superefficiency model is used to determine DEA efficiency scores of the software development projects in the data set.

### 3.3 Analysis of DEA Results

Superefficiency scores were calculated for 616 software development projects. Descriptive statistics of these scores are shown in table 1.

**Table 1: Descriptive Statistics of Superefficiency Scores**

<table>
<thead>
<tr>
<th>Superefficiency (( \Theta_{\text{SUPER}} ))</th>
<th>N</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>616</td>
<td>0.00576</td>
<td>3.41176</td>
<td>0.17541</td>
<td>0.09035</td>
<td>0.28524</td>
<td></td>
</tr>
</tbody>
</table>

The DEA scores obtained using the Superefficiency model are analyzed further using tree based models.

#### 3.3.1 Analysis Using a Decision Tree:

Decision trees are a commonly used tool that can be used in prediction (numeric dependent variable) as well as classification (nominal dependent variable) applications when the independent variables are either nominal or numeric or a combination of both. Two algorithms are popularly used for generating decision trees where the response variable is numeric: Classification and Regression Trees (CART) and Chi-Square Automatic Interaction Detector (CHAID). The CART algorithm was originally developed by Breiman [5] works by recursively partitioning the data into nodes using binary splits.

The CHAID algorithm was developed by Kass [12] and differs from CART by partitioning the data using multi-way splits. This leads to a wider tree but with a smaller depth which makes it easier to interpret. For problems with a numeric response variable, the CHAID algorithm computes F-tests to determine the difference in mean values between split groups and selects the split variable with the smallest p-value. If the smallest p-value is larger than some predefined \( \alpha \) to split, no further splits are performed and the algorithm stops.

In order to avoid overfitting, some of the nodes of the trees are removed through a process known as pruning. The decrease in predictive accuracy due to pruning of each node is compared to the cost having a more complex tree. A more detailed description can be found in [20].

Decision trees are a particularly attractive tool as they can be easily interpreted. The effect of each decision is easily seen at each split in the tree. A set of if-then type of rules can also be developed by traversing a tree from top to bottom. Using tree based models is similar to piecewise linear regression. A non-linear relationship can, therefore, be approximated more accurately using a tree based model instead of a linear regression model.

The data are split into two approximately equal parts with observations being randomly assigned to each part. This resulted in a training set with 282 observations and a test set with 287 observations. The tree induction algorithm was applied to the training set to develop a model. The model was subsequently tested by applying it to the test set.

A tree based model was developed using the CHAID algorithm keeping same dependent and independent variables as the statistical model. i.e. Superefficiency calculated using a DEA model was the dependent variable and the independent variables are as follows:

- **Used Methodology**
  A binary variable indicating whether a formal development methodology has been used or not.
• **CASE Tool Used**
  A binary variable indicating whether a Computer Aided Software Engineering (CASE) tool has been used.

• **AppType**
  A categorical variable with multiple categories indicating the type of application that has been developed (a proxy variable for the complexity of the application being developed, see next section on Project Complexity)

• **DevType**
  A categorical variable indicating whether the project represents a new development, an enhancement or a re-development.

• **LangType**
  A categorical variable indicating the type of language used for development.

• **Max. Team Size**
  A metric variable indicating the maximum size of the development team.

### 3.4 Result and Interpretation

The tree model developed is shown in figure 2. Each node in the tree shows the number of observations \((n)\) falling in that particular group, the number of observations as a percentage of the total sample size and the mean of the predicted efficiency of all the observations falling in that group.

Summary statistics of the model errors are shown in table 2.

<table>
<thead>
<tr>
<th></th>
<th>Training Data</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Error</td>
<td>-0.637</td>
<td>-0.790</td>
</tr>
<tr>
<td>Maximum Error</td>
<td>2.325</td>
<td>1.397</td>
</tr>
<tr>
<td>Mean Error</td>
<td>0.000</td>
<td>-0.010</td>
</tr>
<tr>
<td>Mean Absolute Error</td>
<td>0.119</td>
<td>0.145</td>
</tr>
<tr>
<td>Standard Deviation of Error</td>
<td>0.240</td>
<td>0.255</td>
</tr>
</tbody>
</table>

From figure 2 it is seen that the first split in the tree is made using the Application Type variable with web development projects showing the highest efficiency (0.719) followed by MIS/Other (0.223) and DSS/TPS/WORKFLOW, etc. showing the lowest efficiency (0.140).

Within the WEB category, no further splits are made perhaps due to the small number of observations (7) falling in that category. For both the other categories, further splits are made using the Team Size variable.

For the DSS/TPS/WORKFLOW category (node 1), development projects are divided into two groups: team size of five or less and another group with a team size of more than 5 people. It is seen that projects with smaller teams have higher efficiency (0.217) than projects with larger teams (0.076). For projects where the team size is not mentioned (missing value), a further split is made using the Language Type variable. It is seen here that utilization of fourth generation languages yields a higher efficiency (0.189) than the use of third generation languages (0.126).

The MIS/Other category (node 2) is split further using the Team Size variable. Three categories are created: team size of 3 or less, team size of more than 3 and a category where
the team size is not reported. Smaller teams of three people or less show a higher efficiency (0.446) than teams of more than 3 people (0.121).

Projects with teams of 3 people or less (node 7) are further split based on their usage of CASE tools. Projects where CASE tools have been used show a lower efficiency (0.284) than those where CASE tools have not been used (0.849).

There is no indication that the development type (new, enhancement, etc.) and the use of a formal methodology have any influence on development efficiency.

Figure 2: Decision Tree Model
4. Conclusion

It can be seen from the decision tree analysis that the primary factor that differentiates development projects in terms of efficiency is the type of application that is developed. It is conceivable that the Application Type variable is merely a proxy variable for the complexity of work that is to be done. Web development projects, due to the simplicity of the work to be carried out, are seen to have the highest efficiency.

One can also conclude that the use of smaller development teams lead to higher efficiency. In the tree model in figure 2, each time the Team Size variable has been used as a splitting variable, the node with lower team size has the higher mean efficiency. One reason for this is that the effort required for coordination and communication between team members starts increasing in exponentially with team size leading to decreased efficiencies.

One can also conclude the use of CASE tools actually leads to decreased efficiency especially when used by small teams. Considering that small teams are usually associated with smaller projects, and smaller projects have lower complexity, the use of CASE tools adds more complexity to a project that would otherwise have been simple to execute. The consequence of this complexity is lower efficiency.

It is also seen that the use of fourth generation languages leads to higher efficiency. This is due to the higher level of abstraction that fourth generation languages offer above third generation languages. One does not have to bother with some of the nitty-gritty details of writing code when compared to some of the older languages.

It is also seen that the use of formal methodologies and the type of development are not considered significant based on the tree model that has been developed.

While the use of a methodology, which is normally associated with a more disciplined and systematic approach to software development could lead to fewer number of defects, it could also result in a longer development process. Since the DEA scores are a composite measure of efficiency, the benefits of better quality could be offset by the longer development times thereby indicating that the use of a methodology has no effect on efficiency.

References


