An Empirical Study of Factors Impacting the Size of Object-Oriented Component Code Documentation

Parag C. Pendharkar  James A. Rodger
Penn State Harrisburg   Indiana University of Pennsylvania
Middletown, PA 17057   Indiana, PA 15705
(717)-948-6028    (724)-357-5944
pxp19@psu.edu  jrodger@grove.iup.edu

ABSTRACT
In this paper, we identify a set of factors that may be used to forecast source code documentation for object oriented software components. Using field data on 152 object oriented (OO) software components and the multiple regression model, we empirically test the impact of proposed factors on the prediction of OO source code documentation. Our results indicate that the number of methods, the number of sub-classes, the number of GUI elements and the number of events handled all increase the number of lines of code for OO source code documentation. Additionally, we show that scale economies exist in OO source code documentation and use non-linear regression model Classification and Regression Tree (CART) to improve OO source code documentation forecasting accuracy.

Categories and Subject Descriptors
D.3.3 [Programming Languages]: Language Constructs and Features – abstract data types, polymorphism, control structures.

General Terms
Management, Measurement, Documentation, Economics.

Keywords
Object Oriented Programming Languages, Code Documentation, Multiple Regression, Hypothesis testing.

1. INTRODUCTION
The use of component-based software development (CBSD) has surged recently. Among the reasons for popularity of CBSD are amenability of CBSD in distributed software development, ease of decomposition of software project (when using CBSD), and advances in programming languages. Recent advances in programming languages make platform independent components, such as Java Beans and Windows-only Active X control components available to the user.

Software code documentation plays an important role in lowering maintenance costs. However, when a component-based approach is used to develop software, it may be difficult to estimate the overall size of the object-oriented code documentation. Since code documentation is a “necessary evil” managers need to allocate programmer time for such activity. One approach to estimate the overall code documentation size is to develop accurate forecasting models that can forecast the size of code documentation. The documentation size estimates can be combined with the software size estimates to come up with the total project size.

Estimation of software documentation size is important when CBSD approach is used for software development. Correct estimation of software documentation size in terms of document line of code (DLOC) is important because both, the overall software effort estimation and the software cost estimation models use LOC as an independent variable. Thus, it can be argued that increasing forecasting accuracy of the software documentation LOC may result in better software development effort and accurate cost estimates.

In our paper, we identify the factors that may impact the size of object-oriented component code documentation size. Using object-oriented software metrics literature, we develop a forecasting model for estimating the size of object-oriented code documentation. Using real-life data of 152 object-oriented software components, we test our model. Our results indicate that certain software metrics play a significant role in estimation of software code documentation size. We also show that scale economies exist for software code documentation and illustrate the application of non-linear model Classification and Regression Tree (CART) in forecasting object-oriented software code documentation.

2. FACTORS IMPACTING SIZE OF OO CODE DOCUMENTATION
We draw on the literature related to OO software size to develop our propositions related to OO code documentation size. The causal relationship between software size and source code documentation is well established (see [2]).

The size of an object-oriented system depends on the complexity of the system. There are three levels of complexity for an object-oriented system – method, class and system [6]. Chidamber and Kemerer [4] note that complexity at the method
level is determined by the number of the methods used in a class. They state that the number of methods increase the amount of time and effort required to develop and maintain the class. Further, many methods in a class have a great impact on sub-classes, as sub-classes inherit all the methods from the parent class. Given that sub-classes inherit methods from the parent class, many methods may impose restrictions or cause difficulties in the ability to inherit [4]. The number of parameters used in a method increases the cognitive complexity. The impact of number of parameters used in the method on the component size is not well established [5][6].

Class level complexity depends on the complexity of the given class itself, the complexity of any inherited class, and the number of inherited classes [6][4]. Class complexity may be explained by the complexity of methods [6]. The number of sub-classes that inherit methods of the parent class increases their complexity. Chidamber and Kemerer [4] identify the following reasons for increase in complexity with increase in number of sub-classes. With an increase in the number of sub-classes the level of reuse increases. Further, with increase in the number of sub-classes more testing is necessary to ensure proper reliability.

System level complexity includes non-object-oriented parts that may be too relevant to be neglected [6]. System level non-object-oriented parts include a set of global definitions of types, structures, unions and global declarations of the variables. The complexity at system level may measured as: number of system functions/procedures, number of global definitions, number of global variables, number of graphical user interface (GUI) elements in a component, and number of events and state changes handled by window [6][2].

Reuse and improvements of productivity are one of the objectives of using OO methodology. Providing higher functionality and higher level of inheritance may increase the DLOC, but the increase may be non-linear. In other words, scale economies may exist in OO source code documentation. Based on our literature review, we propose following propositions.

Proposition 1: More methods per component lead to higher software component DLOC.

Proposition 2: More sub-classes lead to higher software component DLOC.

Proposition 3: More GUI elements in a component lead to higher software component DLOC.

Proposition 4: More events handled by in a component lead to higher software component DLOC.

Proposition 5: Variable returns to scale economies are present in software component DLOC.

3. DATA COLLECTION AND ANALYSIS

Data for 152 components were obtained from a real life X/Motif-based C++ research and data-analysis application. The development time for the application was 24 months and a staff of four to nine full-time developers created the component-based application. Table 1 illustrates the means and standard deviations of the independent and dependent variables. The details of our data are available in Bielak [2].

Table 1: The regression parameters and their significance

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLOC</td>
<td>479.93</td>
<td>473.42</td>
</tr>
<tr>
<td>Methods</td>
<td>19.1</td>
<td>17.6</td>
</tr>
<tr>
<td>Sub-Classes</td>
<td>7.1</td>
<td>10.9</td>
</tr>
<tr>
<td>GUI elements</td>
<td>11.51</td>
<td>17.3</td>
</tr>
<tr>
<td>Events</td>
<td>7.5</td>
<td>11.1</td>
</tr>
</tbody>
</table>

We use multiple-regression to test our first four propositions. Thus, our forecasting model is as follows.

\[
DLOC = f (\text{methods}, \text{sub-classes}, \text{GUI elements}, \text{events})
\]

The dependent variable in the model is DLOC and the independent variables are the number of methods in a component, the number of sub-classes in a component, the number of GUI elements in a component and the number of events in a component. We test linear relationship and the function \( f() \) assumes linear mapping from four independent variables to the dependent variable. Table 2 illustrates the statistical significance results of the overall model. The overall model is significant at 0.01 level. Further, 92% of the variance in the dependent variable is explained by the variances in the independent variables.

Table 2: Multiple Regression Results for the Overall Model

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>SS</th>
<th>MS</th>
<th>F Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>4</td>
<td>28626879</td>
<td>7156719.8</td>
<td>201.7*</td>
</tr>
<tr>
<td>Error</td>
<td>147</td>
<td>5216036</td>
<td>35483.2</td>
<td></td>
</tr>
<tr>
<td>Cor. Total</td>
<td>151</td>
<td>33842915</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*significant at \( \alpha = 0.01 \); R-Square=0.92

Table 3 illustrates the regression coefficients and statistical significances of each of the individual variables. The results indicate that all the four independent variables are statistically significant at 0.01 level. Thus, all the four propositions are significant. Based on the values of the regression coefficients, the number of GUI elements in a component has a large impact on DLOC followed by the number of methods, the number of events and the number of sub-classes respectively.
Table 3: The regression parameters and their significance

| Parameter   | Estimate | Std. Error | t Value | Pr >|t| |
|-------------|----------|------------|---------|-----|---|
| Intercept   | 20.29    | 23.36      | 0.87    | 0.386 |   |
| Methods     | 11.00    | 1.03       | 10.97   | 0.0001* | |
| Sub-Classes | 6.98     | 1.67       | 4.19    | 0.0001* | |
| GUI elements| 11.97    | 1.10       | 10.83   | 0.0001* | |
| Events      | 8.34     | 1.91       | 4.38    | 0.0001* | |

*significant at α = 0.05

The results of statistical tests using the DEA model show that our fifth proposition is significant at 0.05 level. This means that variable returns to scale (VRS) economies are present in software component DLOC. In other words, the mapping f(0), used in the proposed forecasting model, from the independent to dependent variable, is non-linear.

The existence of VRS economy means that DLOC will not linearly increase with increase in independent variables, but a non-linear relationship exists. This indicates that non-linear forecasting models for forecasting DLOC may fare better than the linear multiple regression approach. Linear regression, although useful in testing statistical significance, provides limited information and does not fit the data accurately. Better data fitting and more accurate heuristics can be developed if a non-linear model is used to learn a forecasting function. The selection of non-linear model is important. For example, if non-linear quadratic regression model is chosen then a researcher is making an assumption of quadratic relationship between the independent and the dependent variables. Artificial neural networks (ANNs) are used to learn non-linear forecasting models, but do not provide any information on the type of relationship learned to map a set of independent variables to the corresponding dependent variable. We looked at different non-linear forecasting models, and selected one model called CART that met the following selection criteria:

1. The non-linear model should not have any apriori structural restriction, and
2. The non-linear model should be easy to interpret.

We use CART to develop non-parametric, non-linear regression tree for our data. CART provides a regression tree that is sometimes easy to interpret than a simple linear regression model [3]. Figure 1 (shown in appendix) illustrates the regression tree, which was obtained using CART algorithm. The regression tree has 11 nodes, and has several decision support advantages. For example, several heuristic decision rules can be derived from the regression tree. A sample of such a decision rule is as follows.

IF Number of GUI Elements >21.5 AND Number of Sub-Classes ≥16.0 THEN DLOC = 1646.2 (391.3).

The mean and standard deviation figures in CART can be used to estimate confidence limits for DLOC. For example, assuming normal distribution and 3-sigma confidence (99.7%), if GUI elements are greater than 21.5 and the number of sub-classes is greater than or equal to 34 then there is 99.7% chance that DLOC is between closed interval [473.2, 5167.9]. Several other rules and confidence factors can be derived and perhaps an expert system can be developed to guide future DLOC size estimates.

CART algorithm, in addition to a regression tree, provides information on the contribution of the independent variable in the prediction of the size of the dependent variable. This information helps a decision maker to rank independent variables based on their contribution to the prediction of SLOC. Table 3 illustrates the results of the variable relative importance for the prediction of DLOC.

Table 5: The relative variable importance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI element</td>
<td>100.00</td>
</tr>
<tr>
<td>Event</td>
<td>94.83</td>
</tr>
<tr>
<td>Sub-Class</td>
<td>70.40</td>
</tr>
<tr>
<td>Method</td>
<td>45.74</td>
</tr>
</tbody>
</table>

When comparing the results of CART and linear regression, from Table 3 and Table 5, it is clear that the number of GUI elements is a major factor that determines the DLOC. However, there is a disagreement between the result obtained from CART and that obtained from linear regression. CART identifies the numbers of events, sub-classes and methods as the other major factors that impact DLOC. The multiple linear regression analysis identifies methods, events and sub-classes as the other major factors that impact DLOC. Thus, the relative variable importance, other than GUI element, for the two techniques is...
different. The results of CART can be considered more reliable as the model is non-linear and non-parametric.

4. SUMMARY AND CONCLUSIONS

Using a few software complexity measures from the literature, we have developed and tested linear forecasting model for estimating OO software component DLOC. The contributions of our study are two fold. First, we identify some relevant variables that may be used to estimate OO component DLOC. Second, we illustrate that scale economies exist in OO component DLOC, and use a non-linear forecasting model CART to learn the non-linear relationship between the independent and dependent variables.

All the components used in our data set were developed using C++ programming language. Thus, the results of our analysis are not confounded by the effects of multiple languages. The existence of scale economies allows managers to compute the potential for reducing maintenance costs. In our case, it allows managers to evaluate the potential percentage cost reduction when non-linear forecasting model, instead of linear forecasting model, is used. Since we don’t have the cost information in our data set, we only present the formula which can be used when the cost information is available. Banker and Slaughter [1] provide the basis for such calculations. According to Banker and Slaughter [1], if \( e^{\text{SCALE}} = \frac{\Theta}{\Theta} \) then the potential percentage cost reduction can be calculated by \( 1 - \sum \left( e^{\text{SCALE}} \right) \frac{\text{actual component development cost}}{\text{total actual component development cost}} \). Under the assumption that all the component development costs were equal, using non-linear forecasting model would reduce cost by approximately 5% on our data set.

Our study is an exploratory study and we do recognize some issues in generalizing our findings for estimating OO software component DLOC for other projects. We believe that generalization of our findings to development teams with different problem domains, languages, or systems that are significantly larger or smaller than the one in this study may be limited at best. We believe that future research is necessary for improving the external validity of the study.

5. ACKNOWLEDGEMENTS

We thank James Bielak of Greenstone Software Architecture & Consulting for providing the data for current research. Thanks are also due to two anonymous reviewers and the program chair for their valuable comments.

6. REFERENCES


Appendix

Figure 1: Regression Tree Using CART Algorithm

**Abbreviations**
- M - Method
- GUI - GUI akmaste
- SC - Sub-Cluster
- B - Bente

The bold numbers at the end of the arrow are
Mean ± 95% CI (SE) for the elements that belong
to a given split.