An Experimental Study on Conceptual Data Model Based Software Code Size Estimation

Oguz Atak¹, Cigdem Gencel²

¹Havelsan Inc., ²Informatics Institute-METU
oatak@sbd.havelsan.com.tr, cgencel@ii.metu.edu.tr

Abstract:
Effort and cost estimation is crucial in software management. Estimation of software size plays a key role in the estimation process. SLOC has been a commonly used software size metric. However, SLOC of a software project is available only after the coding phase. We need to estimate SLOC early in the life cycle in order to make reliable effort and cost estimation, which are crucial at the beginning of a project.

Being an early phase product of the life cycle and being widely used during the requirements elicitation process of OO systems, the use of Conceptual Data Model for estimating SLOC have been explored in a number of studies. In this study, we explore whether the Conceptual Data Model can serve as an early indicator of software size by conducting an empirical study on two sample projects, which have similar characteristics and developed by the same software company.

Keywords
Software size estimation, Conceptual Data Model, Early estimation, Source Lines of Code (SLOC)

1 Introduction

Software size estimation is an important practical problem in software engineering. Estimation errors are essential cause of poor management. An error in size estimation further causes significant errors in effort and cost estimation, since they consider size as the primary predictor.

There are various approaches to software size estimation. Today’s metrics and methods are criticized by the general difficulty of the estimation process and the immaturity of the measurement science for the software engineering [1], [2], [3]. Major difficulties can be listed as; unclear theoretical basis, lack of standards on methods, insufficient validation of metrics, inconsistent development phase and product coverage of the estimation models, and estimation timing.

Estimation timing is one of the most significant difficulties of software size estimation. We need to have enough knowledge about the software project to make a meaningful size estimate. However, most of the software estimates should be performed at the beginning of the life cycle, when we do not know the details
of the problem we are going to solve. Meli et al. [4] pronounces this as a paradox: “Size estimation would be necessary when we do not have enough information and early estimation methods to obtain it. When we can measure with the greatest accuracy, we do not need that information any more”.

Being a product attribute, software size has been measured by measuring the artifacts, deliverables or documents produced during a software process activity.

Software size measurement process has involved a wide range of metrics and methods. The first and the oldest approach to measure the size of the software product is to associate it with the “length” attribute of the source code. It can be measured in terms of Source Lines of Code (SLOC), number of characters, etc. However, we can measure this attribute only after the code is available.

In order to develop a model that can estimate code size very early in the life cycle, process products available in the very early phases need to contain indicators of the length attribute [5]. The estimators in other engineering disciplines use construction standards and architectural drawings to assess the size of the final product and to aid in developing initial project size very early in the development life cycle. However, the software engineering field lacks such architectural form to assist estimators.

Software metrics developed with traditional methods usually do not take into account specific characteristics of object-oriented (OO) design. An OO design process is only partly based on functionality and concentrates more on actions, behaviours and interfaces. So, new size estimation models are expected to support the use of OO elements like Classes, Interfaces and Relations instead of logical inputs, logical outputs and logical files.

The representation of a real world problem with an object model will correspond to a great extent to the implementation of the corresponding system in an OO programming environment. An object model, which is also referred by the term “Conceptual Data Model”, consists of a set of objects that interact with one another. Being an early phase product of the life cycle and being widely used during the requirements elicitation process of OO systems, the use of Conceptual Data Model for estimating SLOC have been studied in a number of studies [6] [7] [8].

In this study, we explore whether the Conceptual Data Model can serve as an early indicator of software size by conducting an empirical study. A case study is conducted on two sample projects, which have similar characteristics and developed by the same software company. Both projects were developed by the same development team, using the same programming language -Java- and framework. Although the clients and the domains differ, both projects are typical Management Information Systems (MIS) involving workflow and content management operations.
This paper is organized as follows; the related research on software size metrics and OO size estimation methods are briefly summarized in the second section. In the third section a case study is presented. Finally, conclusions are given in the last section.

2 Background

Software size can be described in terms of length, functionality, and complexity [3]. Reuse can also be identified as an aspect of size, since it measures how much of a product was copied or modified.

In this study, our scope is size estimation of OO systems. The OO approach centers on modeling the real world in terms of objects. A real world problem is represented by a set of objects that interact with one another in OO design. However traditional approaches mainly emphasize a function-oriented view that separates data and procedures. And also software metrics developed with traditional methods do not readily count the OO notions such as classes, inheritance, encapsulation and message passing.

The metrics for OO systems are different due to the different approach in programming paradigm. Chidamber and Kemerer developed and evaluated design metrics for object-oriented systems, to justify the quality of an object model [9]. They also studied on the quality issues for OO programming. Another study related to quality issues of OO programming is made by Kaczmarek and Kucharski [8] in which Java code characteristics are investigated.

Up to now, several size estimation metrics and methods have been developed for OO systems. “Object Points (OP)” is a software sizing method based on an earlier work by Kauffman and Kumar [10]. The underlying concepts for this method are very similar to FPA. In this method, objects are taken as the basis of the counting process. However, such objects are not directly related to “objects” in the OO methodology, but rather refer to screens, reports, or 3GL modules.

Another method is presented by Caldiera et al. [11], “Object Oriented Function Points (OOFP)”. The method is characterized by a mapping of FP concepts (logical files and transactions) to OO concepts (classes and methods), and by a flexible method for handling specific OO concepts like inheritance and aggregation.

“Predictive Object Points (POP)” method [12] takes advantage of the characteristics of object-oriented development. The technique is based on a measure called “weighted methods per class (WMC)” which is suggested by Chidamber and Kemerer (1994) [9]. Other metrics involved in POP count are: number of top-level classes (TLC), average depth of inheritance tree (DIT), and average number of children per base class (NOC).
Kammelar [13] expressed the functional size in “Component Object Points (COP)” obtained from elements, which may be candidates for reusable software components. The method takes use cases as a base in its calculations.


“Use Case Points (UCP)” method was developed by Gustav Karner [15]. UCP method is based on FPA method. The measurement process involves the weighted count of the number of actors of the use case model and the number of use cases.

Whitmire proposed the application of his “3D Function Points” to object-oriented software systems, by considering each class as an internal file and messages sent across the system boundary as transactions [16]. However, 3D Function Points require a greater degree of detail in order to determine size and consequently make early counting more difficult.

“Class Point (CP)” is a method that generalizes the FP method for OO systems [17]. The Class Point method is especially focused on classes, which are the entities the method should count and weigh on the basis of their complexity levels, as was the case for functions in FPA. The CP method is based on information from design documentation. Clearly, much of the information required by the CP method is also available only when the design of the system completes.

The “Shepperd and Cartwright Size Prediction System” [18] was developed in 1997. By using the data from the empirical investigation of an industrial OO real time C++ system, they found that the count of states per class in the state model could be a good predictor of size in SLOC.

Tan et al. [6] [7] proposed a SLOC estimation method based on “Conceptual Data Model” for information systems developed with OO methodology. Conceptual Data Model refers to the class diagram that models the entities and concepts in an information system and the relationships between them. The size of a data-intensive system in SLOC is estimated based on the number of classes, relationships and attributes per class.

Kaczmarek and Kucharski [8] studied the quality issues of OO programming and investigated Java code characteristics. They found out that average class size and average method size does not depend on application size, but rather on process maturity, design quality and application specific factors such as user interface complexity, etc.
3 Case Study

3.1 Research Approach

In this study, our aim is to replicate two previous studies [6] [7] [8] to estimate SLOC. We first explore whether the Conceptual Data Models can serve as early indicator of software size by conducting an empirical study on sample projects.

The characteristic of a data-intensive software system, and therefore the source code of its software, is well characterized by its conceptual data model [6]. Moreover, conceptual data models are widely used during the requirements analysis phase of the software development life cycle. In building conceptual data models, at least Entity, Attribute and Relationship constructs are used.

Based on these assumptions and through much experimentation for a programming language or environment, in [6] and [7], a Multiple Linear Regression analysis was made on the data from a number of software projects with similar characteristics to estimate SLOC from the constructs of their conceptual data models.

The proposed multiple linear regression model, formulated in [6] is as follows:

\[
KLOC = \beta_0 + \beta_C C + \beta_R R + \beta_A \bar{A} \tag{1}
\]

In this model, the dependent variable is the size of its source code in thousand lines of code (KLOC); and \(\beta_0, \beta_C, \beta_R, \beta_A\) are the coefficients to be estimated from the samples. The model has the following three independent variables to characterize the conceptual data model:

- C: the total number of classes in the conceptual data model.
- R: the total number of relationships in the conceptual data model.
- \(\bar{A}\): the average number of attributes per class.

Similar models, each having different coefficient, were built for each programming language or the development environment used.

The information required by the proposed method is more readily available in the early stage of software development. In the worst case, all the information required can also be fully available when the requirements analysis completes.

The second aim of this empirical study is to observe whether the average class size and average method size does not depend on application size as proposed by Kaczmarek and Kucharski [8]. They examined the quality issues of OO programming and the Java code characteristics.

According to this study, measured Java application metrics prove that average class size and average method size should not depend on application size, but rather on process maturity, design quality and application specific factors such as user interface complexity etc [8].
3.2 Description of the Case

For the case study we selected two projects from an organization. The development organization is a CMM Level 3 certified company located in Turkey. The projects developed in the organization are usually from information systems domain and the customers were mainly the governmental organizations.

Both projects have similar characteristics and they were developed using the same development team, programming language -which is Java-, and the same framework. According to ISO/IEC TR 14143-5 [19], we determined the projects belonging to “Information System” domain.

Project-1 is development of an Information System for the Justice Ministry of Turkey. The project was completed in 48 months. The effort utilized was 3900 man-months. The project consisted of 8 main modules. During the project life cycle, incremental model was followed and the software was delivered in 2 iterations.

Project-2 is an Information System developed for election system in Turkey. The client is the government, as in Project-1. The project is relatively small and consisted of a single module. The project was developed with waterfall methodology in 12 months. The development effort utilized for this project is 60 man-months. The summary of the projects’ description is given in Table-1.

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Size (SLOC)</th>
<th>Duration (Month)</th>
<th>Effort (Man-Months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-1</td>
<td>8,010.835</td>
<td>48</td>
<td>3,900</td>
</tr>
<tr>
<td>Project-2</td>
<td>203.336</td>
<td>12</td>
<td>60</td>
</tr>
</tbody>
</table>

Both projects were built using the same application framework. The framework was developed concurrently with Project-1 and later some modifications were made during the development of Project-2. Project-1 consisted of 8 modules, each of which was deployed independently. These modules were developed and maintained by different teams. Moreover, these modules have separate conceptual models and different branches on the code repository.

3.3 Data Collection and Analysis

In this study, we first evaluated the model proposed in [7] to estimate the SLOC from the metrics collected from the conceptual data model. And also quality issues of Java mentioned in [8] are inspected. And further we explored the use of Conceptual Data Model based size estimation method.
In [7], a separate software-sizing model is built according to the programming language used and to the source (industry/open-source) of the dataset. Among the proposed models, which namely best suits to our sample projects is “Industry Java-Based System” and it is formulated as follows:

$$KLOC = -10.729 + 1.324 \times C + 1.254 \times R + 0.889 \times \bar{A}$$  \hspace{1cm} (2)

In order to evaluate the above model, we extracted data from the Conceptual Data Models. The total number of classes (C), the total number of relations (R) and the total number of attributes (A) are extracted from the conceptual data models. These metrics reflect the early development phase of the project. Actual SLOC count is taken from the code repository. Physical SLOC excluding the comment and blank lines are counted using ‘Understand for Java [20]’ reverse engineering, code navigation, and metrics tool. The codes created with automatic code generators are also excluded. Class libraries and Components-Off-The-Shelf (COTS) were not counted.

The obtained dataset, the estimated SLOC utilizing the model, the actual SLOC and the differences between these values are given in Table-2.

<table>
<thead>
<tr>
<th>Project No</th>
<th>Module No</th>
<th>C</th>
<th>R</th>
<th>A</th>
<th>$\bar{A}$</th>
<th>Estimated (KLOC)</th>
<th>Actual (KLOC)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-1</td>
<td>Module 1</td>
<td>124</td>
<td>87</td>
<td>739</td>
<td>5.96</td>
<td>267.82</td>
<td>754.38</td>
<td>- 64.5</td>
</tr>
<tr>
<td></td>
<td>Module 2</td>
<td>297</td>
<td>222</td>
<td>772</td>
<td>2.6</td>
<td>663.18</td>
<td>1115.49</td>
<td>- 40.5</td>
</tr>
<tr>
<td></td>
<td>Module 3</td>
<td>201</td>
<td>87</td>
<td>681</td>
<td>3.39</td>
<td>367.48</td>
<td>1116.85</td>
<td>- 67.1</td>
</tr>
<tr>
<td></td>
<td>Module 4</td>
<td>173</td>
<td>109</td>
<td>757</td>
<td>4.38</td>
<td>358.88</td>
<td>1129.19</td>
<td>- 68.2</td>
</tr>
<tr>
<td></td>
<td>Module 5</td>
<td>227</td>
<td>266</td>
<td>517</td>
<td>2.28</td>
<td>625.39</td>
<td>513.32</td>
<td>+ 21.8</td>
</tr>
<tr>
<td></td>
<td>Module 6</td>
<td>155</td>
<td>95</td>
<td>604</td>
<td>3.9</td>
<td>317.07</td>
<td>1124.76</td>
<td>- 71.8</td>
</tr>
<tr>
<td></td>
<td>Module 7</td>
<td>118</td>
<td>108</td>
<td>584</td>
<td>4.95</td>
<td>285.31</td>
<td>1202.1</td>
<td>- 76.3</td>
</tr>
<tr>
<td></td>
<td>Module 8</td>
<td>66</td>
<td>67</td>
<td>355</td>
<td>5.38</td>
<td>165.43</td>
<td>1054.74</td>
<td>- 84.3</td>
</tr>
<tr>
<td>Project-2</td>
<td></td>
<td>514</td>
<td>382</td>
<td>804</td>
<td>1.56</td>
<td>1150.21</td>
<td>212.83</td>
<td>+ 440.4</td>
</tr>
</tbody>
</table>

The results show that there is a significant difference between the estimated and actual SLOC; in a range between -84.3% + 440.4%. And also there is no such consistent pattern reflected.

This might have been due to a number of reasons; the proposed method is not applicable to estimate the size of the case projects or the design does not reflecting the design, i.e. the code involves major modifications after the design phase,
which were not reflected to the design document. The proof of this idea needed further analysis on the code and it is presented in detail in [21].

Therefore, we extracted the data directly from the code instead of the design document through reverse engineering in order to make further analyses.

Existing Java classes in the repository are classified according to their types, which are external classes, inner classes, anonymous classes and interface classes. The dataset obtained after this analysis is given in Table-3.

<table>
<thead>
<tr>
<th>Module</th>
<th>Num. of External Classes</th>
<th>Average Size of External Classes</th>
<th>Num. of Interfaces</th>
<th>Average Size of Interfaces</th>
<th>Num. of Inner Classes</th>
<th>Average Size of Inner Classes</th>
<th>Num. of Anonymous Classes</th>
<th>Average Size of Anonymous Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td>3680</td>
<td>184.96</td>
<td>431</td>
<td>27.29</td>
<td>1242</td>
<td>21.59</td>
<td>3315</td>
<td>8.99</td>
</tr>
<tr>
<td>Module 2</td>
<td>5137</td>
<td>197.51</td>
<td>548</td>
<td>26.91</td>
<td>1418</td>
<td>22.19</td>
<td>4339</td>
<td>8.43</td>
</tr>
<tr>
<td>Module 3</td>
<td>5087</td>
<td>200.31</td>
<td>527</td>
<td>24.24</td>
<td>1164</td>
<td>23.51</td>
<td>4867</td>
<td>9.27</td>
</tr>
<tr>
<td>Module 4</td>
<td>5082</td>
<td>200.59</td>
<td>624</td>
<td>25.93</td>
<td>2461</td>
<td>16.84</td>
<td>5007</td>
<td>7.4</td>
</tr>
<tr>
<td>Module 5</td>
<td>2587</td>
<td>179.21</td>
<td>355</td>
<td>22.26</td>
<td>741</td>
<td>28.76</td>
<td>2680</td>
<td>7.75</td>
</tr>
<tr>
<td>Module 6</td>
<td>4946</td>
<td>207.6</td>
<td>562</td>
<td>26.1</td>
<td>1043</td>
<td>23.8</td>
<td>4749</td>
<td>8.08</td>
</tr>
<tr>
<td>Module 7</td>
<td>5288</td>
<td>206.33</td>
<td>618</td>
<td>29.05</td>
<td>1926</td>
<td>17.33</td>
<td>5108</td>
<td>8.68</td>
</tr>
<tr>
<td>Module 8</td>
<td>4464</td>
<td>215.98</td>
<td>482</td>
<td>26.85</td>
<td>1049</td>
<td>23.74</td>
<td>4447</td>
<td>8.94</td>
</tr>
</tbody>
</table>

In the above dataset, it is observed that average sizes of the anonymous and inner classes of the modules are very close for both projects. And if we only consider the 8 modules of Project-1, the average sizes of external classes, anonymous classes and interfaces show a low variation. Table-4 shows the minimum, maximum, median and average class size values, the standard deviation and coefficient of variation (in percentage).
Table 4. The Variation between the Average Class Sizes

<table>
<thead>
<tr>
<th>Project No</th>
<th>Class Type</th>
<th>Min. Size (SLOC)</th>
<th>Max. Size (SLOC)</th>
<th>Median Size (SLOC)</th>
<th>Average Size (SLOC)</th>
<th>Standart Deviation</th>
<th>Coefficient of Variation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>External Classes</td>
<td>179.21</td>
<td>215.98</td>
<td>200.59</td>
<td>199.06</td>
<td>12.02</td>
<td>6.04</td>
</tr>
<tr>
<td></td>
<td>Interface Classes</td>
<td>22.26</td>
<td>29.05</td>
<td>26.1</td>
<td>26.08</td>
<td>2.05</td>
<td>7.88</td>
</tr>
<tr>
<td></td>
<td>Inner Classes</td>
<td>16.84</td>
<td>28.76</td>
<td>23.51</td>
<td>22.22</td>
<td>3.83</td>
<td>17.22</td>
</tr>
<tr>
<td></td>
<td>Anonymous Classes</td>
<td>7.4</td>
<td>9.27</td>
<td>8.43</td>
<td>8.44</td>
<td>0.65</td>
<td>7.73</td>
</tr>
<tr>
<td>Project-2</td>
<td>External Classes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>150.99</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Interface Classes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Inner Classes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>26.31</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Anonymous Classes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.2</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

From this table, we concluded that:

- “Coefficient of Variation” values are less than 10% for external classes, anonymous classes and interfaces in Project-1. The lower value of ‘Coefficient of Variation’ shows the closeness between the sizes of related class types in the project. The average sizes of these class types can be said to be stable within Project-1. This may be because of the similarity in the environment and the development methodology of these 8 modules. As a result, we can get more precise estimations for similar software systems, which are to be developed in the same environment and with the same development technology [8].

- The average sizes of anonymous classes are similar for Project-1 and Project-2, which is between 8 - 9 SLOC. This result is very close to the average value, which was calculated 8.2 by Kaczmarek and Kucharski [8].

By using the data extracted from the code, we also made multiple linear regression analysis to estimate SLOC. The correlation between the number of classes, attributes, relations and SLOC was also examined.
The multiple linear regression model proposed in [7] is calibrated by using new coefficient values, fitting better to our sample data. The dataset used in building the SLOC estimation model is presented in Table-5. The dataset was analyzed using a statistical analysis tool, called Minitab [22].

<table>
<thead>
<tr>
<th>Y</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>X5</th>
<th>X6</th>
<th>Transformed Y (Y1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>754376</td>
<td>3680</td>
<td>66889</td>
<td>6.01</td>
<td>124</td>
<td>87</td>
<td>5.96</td>
<td>5.877587863</td>
</tr>
<tr>
<td>1115491</td>
<td>5137</td>
<td>89143</td>
<td>5.75</td>
<td>297</td>
<td>222</td>
<td>2.6</td>
<td>6.047466071</td>
</tr>
<tr>
<td>1116850</td>
<td>5087</td>
<td>94906</td>
<td>6.12</td>
<td>201</td>
<td>87</td>
<td>3.39</td>
<td>6.047994849</td>
</tr>
<tr>
<td>1129191</td>
<td>5082</td>
<td>98600</td>
<td>6.15</td>
<td>173</td>
<td>109</td>
<td>4.38</td>
<td>6.052767408</td>
</tr>
<tr>
<td>513322</td>
<td>2587</td>
<td>44068</td>
<td>6.14</td>
<td>227</td>
<td>266</td>
<td>2.28</td>
<td>5.710389878</td>
</tr>
<tr>
<td>1124764</td>
<td>4946</td>
<td>88780</td>
<td>6.17</td>
<td>155</td>
<td>95</td>
<td>3.9</td>
<td>6.051061408</td>
</tr>
<tr>
<td>1202099</td>
<td>5288</td>
<td>96222</td>
<td>6.09</td>
<td>118</td>
<td>108</td>
<td>4.95</td>
<td>6.079940236</td>
</tr>
<tr>
<td>1054742</td>
<td>4464</td>
<td>82549</td>
<td>6.43</td>
<td>66</td>
<td>67</td>
<td>5.38</td>
<td>6.02314624</td>
</tr>
<tr>
<td>203336</td>
<td>1249</td>
<td>15900</td>
<td>5.53</td>
<td>514</td>
<td>382</td>
<td>1.56</td>
<td>5.308214276</td>
</tr>
</tbody>
</table>

Below, the data types in the above table are briefly described:

- **Y**: Actual Physical SLOC
- **Y1**: Lognormal transformed Y
- **X1**: Total Number of Classes from Code
- **X2**: Total Number of Relations from Code
- **X3**: Average Number of Attributes per Class from Code
- **X4**: Total Number of Classes from Conceptual Data Model
- **X5**: Total Number of Relations from Conceptual Data Model
- **X6**: Average Number of Attributes per Class from Conceptual Data Model

Y is the dependent variable in the model; independent variables are divided into two groups:

- Independent variables from code: X1, X2, X3
- Independent variables from conceptual data model: X4, X5, X6

The regression equation for X1, X2, X3 and Y1 variables is obtained as:
\[ Y_1 = 8.88 + 0.000572 \times X_1 - 0.000011 \times X_2 + 0.521 \times X_3 \quad (3) \]

According to the table, regression is statistically significant since \( P < 0.01 \). \( R^2 \) is 0.98 and Adjusted \( R^2 \) is 0.97. So it is concluded that there is a strong relationship between the independent variables and the dependent variable. Independent variables explain nearly all of the variation in the dependent variable. MMRE is calculated 0.98. The value of MMRE tells us that the model isn’t appropriate for prediction. Although the regression is statistically significant, the model is rejected.

The regression equation for \( X_4, X_5, X_6 \) and \( Y_1 \) variables is obtained as:
\[ Y_1 = 15.7 - 0.00149 \times X_4 - 0.00581 \times X_5 - 0.237 \times X_6 \quad (4) \]

Regression is statistically is not significant since \( P > 0.01 \). \( R^2 \) is 0.82. Adjusted \( R^2 \) is 0.72. For our sample data, MMRE is calculated as 0.98. So the model is rejected.

### 3.4 Discussion of the Case Study Results

In this study, we explored whether the Conceptual Data Model constructs can be used for SLOC estimation. We observed that there is a strong relation between the actual number of classes, attributes, relations and the actual SLOC in an OO system.

Conceptual Data Model can serve as an early indicator of software size. However, the estimation relies on the quality of the conceptual data models constructed during the requirements analysis phase. The conceptual data model should be well formed, which would be the one to be reflected in the code. Only then we can use the estimation models reliably.

While building a regression model, the coefficients of independent variables depend on the sample project data. If a model is built based on the historical data for similar projects in an organization, then the estimation results would be much reliable.

Every distribution based on experimental data contains disturbances. The smaller the population for distribution is, the larger the fluctuations are. Therefore, more observations are needed for building reliable regression models. Having few cases relative to number of predictors may lower the reliability of the estimation models.

Any estimation model needs to be tuned to the specific environment where it will be applied. Changes in the technology and the development environment affect the size of the code and accuracy of the estimation. More precise estimation should be expected in a more homogeneous project environment and a limited project type range [8].
4 Conclusion

This study explores the usability of Conceptual Data Model in software code size estimation. In [7], an estimation model for SLOC, based on the basic constructs of a conceptual data model -classes, attributes and relations- is presented. The method is built on the total number of classes, the number of attributes per class and the total number of relations using multiple linear regression analysis.

This study aimed to replicate previous conceptual data model based estimation methods by means of an empirical case study. The sample data is collected from past projects of a software company, which are both in information system domain.

After the first evaluation of the method, the model seemed not to be usable in this organization for size estimation. The reason the model is not working well in our case was that, our data we extracted from the conceptual data model was not reflecting the code. Another point was that the model was built on a limited number of samples and validated with incomplete data. Therefore, we worked with the actual data extracted from the code. By using the data extracted from the code, we also made multiple linear regression analysis to estimate SLOC. The correlation between the number of classes, attributes, relations and SLOC was also examined. Although there is a strong correlation between, the built model is not working on those data as well.

The results of the study showed that we can develop SLOC estimation models based on Conceptual Data Model constructs for similar projects, which are to be developed in similar environment and by using similar methodology.

More empirical research is required on homogeneous and large datasets to explore the relationship between SLOC and the conceptual data model constructs. For example, if the International Software Benchmarking Standards Group (ISBSG) collects data about these attributes in their repository, this will contribute to improve this size estimation approach, which is a promising one.

References


