SOFTWARE SIZE ESTIMATION OF INDIVIDUAL PROJECTS

Cuauhtémoc López-Martín, Arturo Chavoya, and María E. Meda-Campaña

Information Systems Department, CUCEA, Guadalajara University
Periférico Norte 799, Los Belenes, Zapopan, Jalisco, México

cuauhtemoc@cucea.udg.mx, achavoya@cucea.udg.mx, emeda@cucea.udg.mx

ABSTRACT
Background: Software project size is often used as independent variable for predicting dependent variables such as effort, schedule, costs or risks of software projects. The better the size estimation accuracy, the better the prediction accuracy of dependent variables. The size of each project is usually estimated in terms of the number of lines of code for the programming language in which the project will be coded. This estimation is typically done using expert judgment techniques or applying prediction models.

Hypothesis: There is a statistically significant difference amongst size prediction accuracy of projects by the object-oriented programming language used, when they are estimated by expert judgment from specified requirements in natural language.

Method: A population of 1,414 individual software projects was developed by 202 practitioners. Each project had its own specified requirements in natural language, and each one was developed within a controlled experiment and following a disciplined process. A sample of 676 projects developed in C++ or Java was selected for this study.

Results: There was a statistically significant difference between size estimation accuracy for C++ and Java at a 95% level of confidence.

Conclusions: The size estimation accuracy of software projects coded in Java was better than the estimation accuracy of projects coded in C++.

KEY WORDS
Software development, software size estimation, programming languages, expert judgment.

1. Introduction

The estimation accuracy of factors related to a software project is usually attributable to the accuracy of size prediction: “Software sizing activity is the heart of any software estimation model, if you get that wrong, there is little chance of arriving at a sensible effort or cost estimate” [9]. Those factors can be the effort, schedule, costs, or risks, whereas the size of a software project can be measured in either function points or lines of code [24].

Once the specification of requirements for the software project has been established, the approaches for estimating the project size can be one of the following two [9]:

1) Expert judgment approaches based upon the use of historical sizing data, drawing analogies with previously developed projects to arrive at a size of a target project.

2) Based upon specific standardized techniques, such as IFPUG, NESMA or COSMIC.

In this work, the size of each software project was obtained from estimation by expert judgment, and each was measured in lines of code whose size estimation depended on the programming language used. Programming languages can be classified into imperative and declarative languages [6]. Imperative languages include structured (e.g. C or Pascal), object-oriented (e.g. C++ or Java), and concurrent (e.g. Ada or Modula) languages, whereas declarative languages include logic-based (e.g. Prolog), functional (e.g. Lisp), and fourth generation languages for databases (e.g. SQL). The population of software projects for this study included several programming languages, whereas the final sample included only projects coded in C++ or Java.

In accordance with the abstraction level of software projects (the abstraction describes on which level the measurements and estimations are carried out), there exist the following five levels [22]: organization, process, project, individual and task. This study is related to the individual level once software projects were individually developed by practitioners. Project size is analyzed from two kinds of lines of code: new and changed, and reused.

Individual projects are studied based upon the following assumptions:

1) “The performance of a development organization is determined by the performance of its engineering teams, […] the performance of an engineering team is determined by the performance of the team members, and […] the performance of the engineers is, at least in part, determined by the practices these engineers follow in doing their work” [7].
2) “Unless engineers have the capabilities provided by personal training, they cannot properly support their teams or consistently and reliably produce quality products” [27].

The data of the software projects for this research were gathered following a disciplined process of practices of the Personal Software Process (PSP) [27]. The levels of software engineering training can be classified in the small as well as in the large. A small level training involves the application of software engineering principles to the development of a software product by an individual, whereas a large level involves the application of software engineering principles to the development of a software product by a team [8]. PSP was selected for this work because its practices and methods have consistently proven useful when applied to the creation of small programs by practitioners for delivering quality products on predictable schedules [7].

Because the specification of requirements of software projects is usually written in natural language, there is always the risk of introducing ambiguities in them and provoking misunderstandings in the developers [12]. Hence, in this research each practitioner received the specified requirements before the project size was estimated.

Considering the importance of estimating the size of software projects whose requirements have been specified, and that this estimation may depend on the programming language used, the research question of this study is the following:

Does the programming language used have any influence on the size estimation accuracy of software projects when their requirements have been specified in natural language?

Given that the sample of this study integrates only those projects coded in C++ or Java, the hypotheses of this research are the following:

H_1: There is a statistically significant difference in the size estimation accuracy of new and changed code between the projects coded in C++ and those coded in Java when the requirements of the projects are specified in natural language.

H_2: There is a statistically significant difference in the size estimation accuracy of reused code between the projects coded in C++ and those coded in Java when the requirements of the projects are specified in natural language.

1.1 Project Size

At present, the number of lines of code represents a common measure for sizing software projects [24]. The lines of code can be classified into the following types [27]:

- Base: Total lines of code of previous projects.
- New: Lines of code written during the current project programming process.
- Changed: Lines of code changed in the base project when modifying a previously developed project.

Reused: Lines of code of previously developed projects (base code), that are used without any modification.

It is a common practice to group the new and changed lines of code (N&C) to be used as one independent variable [27]. Hence, in this study, two kinds of lines of code are analyzed: N&C and Reused.

On the other hand, there are two measures of source code size [23]: physical and logical. In this study, all practitioners selected the physical counting.

1.2 Accuracy Criterion

A common criterion for evaluating the estimation accuracy in the software engineering field is the Magnitude of Relative Error (or MRE) [13], which is used in this research. MRE is defined as follows:

\[
MRE_i = \frac{|Actual\ Size_i - Estimated\ Size_i|}{Estimated\ Size_i}
\]

The MRE value is calculated for each observation \(i\) whose size is estimated. The aggregation of MRE over multiple observations (N) can be achieved through its mean (MMRE) as follows:

\[
MMRE = \frac{1}{N} \sum_{i=1}^{N} MRE_i
\]

The estimation accuracy is inversely proportional to the MMRE value.

2. Related Work

In spite of its importance, there are unfortunately few studies related to size estimation (a systematic review reveals only 3% of research articles involved a case study [9]). The identified studies related to our research are the following:

Verner and Tate [17] proposed an equation for software size estimation which is generated from a bottom-up approach, that is, it determines the size of individual software components or modules first, and then obtains subsystem and system sizes by summing up component sizes (in lines of code). They identified factors affecting software size (independent variables) from which the equation is built using regression methods. Afterwards Dolado [18] corroborated the following two results from Verner and Tate:

1) The partition of a system in components is not technology-independent; and function-point techniques are only the de facto measure of functionality.
2) Estimation of the size of a system by sizing the components (bottom-up) can be advantageous and accurate.

Živković et al. [1] estimated project size using function points as measure. This estimation was from Unified Modeling Language (UML) diagrams built in three abstraction levels.

Hakkarainen et al. [14] proposed a neural network model for estimating software size measured in function points. The multilayer feedforward neural network used was trained with structured analysis descriptions.

Živković et al. [2] analyzed the impact on software size (measured in function points) when it was estimated from class diagrams having diverse detail levels.

Hakuta et al. [20] proposed a model for estimating software size from the software project design phase.

Levesque et al. [10] estimated software size (measured in function points) from UML diagrams.

Demirörs and Gencel [21] compared several techniques for estimating the size of software projects, which was measured in function points.

Kaczmarek and Kucharski [15] presented a methodology for estimating software size. The methodology involved measures such as classes and methods written in Java. The analysis of almost one million lines of code led them to the conclusion that the average class size and the average method size were independent from application size, that is, they could be useful for estimating application size when the number of classes was known.

MacDonnel [25] applied fuzzy logic modeling methods for estimating software size measured in lines of code written in fourth generation programming languages. The independent variables were entity-relationship diagrams and functional decomposition charts depicting the high-level menu, screen and report processing hierarchy of the system. His results suggested that fuzzy predictive models can outperform their statistical regressions.

Lokan [4] proposed statistical models for predicting the size of software projects measured in lines of code. The independent variables were modules and their variables.

Tan et al. [11] generated a multiple linear regression model for estimating the lines of code from a conceptual data model. The independent variables were classes, number of relationships amongst them, and the average number of attributes by class.

According to previous studies, project size has mainly been estimated from a high level design ([1] [2] [3] [4] [10] [11] [14] [15] [17] [18] [19] [20] [21] [25]) involving UML, structured diagrams, components, or modules. Regarding the size measurement of projects, they have been measured in either lines of code ([3] [4] [11] [15] [17] [18] [19] [25]) or in function points ([1] [2] [10] [14] [21]). As for the estimation technique, studies have used either expert judgment ([1] [2] [10] [17] [18]) or models [20] such as statistical methods [4] [11] [19], neural network [14], fuzzy logic [25], and techniques related to Function Points Analysis such as IFPUG, Mark II, NESMA or COSMIC [21].

The approach for our research is based on measuring the influence of the programming language on software project size when it is measured in lines of code and whose requirements have been specified in natural language.

### 3. Experimental Design

This research supports the necessity of empirical software engineering studies based on reproducibility [16]. Our experimental design is mainly based on the one reported in [5] and it could be regarded as a quasi experiment, as the practitioners were not selected randomly [26], and if there were missing data in a participant’s log, that dataset was excluded from the analysis and its data sample. Our study was based on a modified Personal Software Process with seven assignments, instead of the ten in the original PSP [27], to take into consideration that course duration is a key concern for organizations.

The experiment was performed in a controlled environment with the following characteristics:

- All developers were experienced practitioners working on software development in organizations, and none of them had previously taken a course related to personal practices in software development.
- All the practitioners were registered in a graduate program in computer science. Participation in the study was not mandatory, and the participants did not receive any payment for attending the course.
- Each practitioner selected the programming language to use in the assignments.
- Practitioners had already taken at least one course on the programming language they used in the assignments.
- To reduce bias, practitioners were neither informed about of our experimental objective, nor penalized for their performance.
- Only one assignment was performed by day. At the beginning of each session, a description of the assignment was given to each practitioner. That description included three sections, the first one included a list of documents each practitioner should have for starting the project (project plan summary, time recording log, defect recording log, coding standard, counting standard, test case report, and process improvement proposal); the second section included the requirements specification written in natural language (Appendix A shows an example), and a third section included the documents each practitioner should submit to review, once the project had been finished (including filled logs, source code, code review list, and design review list).
- Practitioners were supervised and mentored about the process during the assignments: after each assignment, their documentation was reviewed and they received feedback when requested about any issue before starting their next assignment.
- All the practitioners adopted a coding and a counting standard established by each of them.
The code written in each assignment was designed by the developer to be reused in subsequent assignments. The assignments had a complexity similar to that suggested in [27]. From a set of 18 software projects, a subset of 7 was randomly assigned to each of the 202 practitioners.

3.1 Data Population

The software projects were developed between the year 2005 and the first months of 2012. They totaled 1,414 and were developed by 202 practitioners. The development process followed by practitioner is shown in Table I [27]. The programming languages used by practitioners were Java, C++, Visual Basic, Delphi, PHP, ABAP, Visual FoxPro and Perl. The projects coded by programming language are shown in Table 2.

Table 1

<table>
<thead>
<tr>
<th>Phase</th>
<th>First</th>
<th>Second</th>
<th>Third</th>
<th>Fourth</th>
<th>Fifth</th>
<th>Sixth</th>
<th>Seventh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Design</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Design review</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Code</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Code review</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Compilation</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Testing</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Postmortem</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

3.2 Data Sample

Given that the 1,414 projects from Table II did not follow all the development process phases described in Table I, a sample of projects that implemented all the phases was selected. The fourth, fifth, sixth and seventh project by practitioner integrated this sample, once the projects were planned, designed (either in flow diagram or pseudocode), reviewed its design, coded, reviewed its code, compiled, tested and documented their final measures (postmortem).

From all the projects, those coded in Java in C++ were selected for this study because there was a larger number of subsamples of them and they consisted of a similar number of projects.

The sample size consisted of 676 projects developed by 169 practitioners in either Java or C++. This sample was divided according to their kind of lines of code. In this case, all of them (676) had new and changed code, and only 640 of them contained reused code (Table III).

Table 3

<table>
<thead>
<tr>
<th>Type of lines of code</th>
<th>Language of Software projects</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java</td>
<td>C++</td>
</tr>
<tr>
<td>New and changed (N&amp;C)</td>
<td>364</td>
<td>312</td>
</tr>
<tr>
<td>Reused</td>
<td>352</td>
<td>288</td>
</tr>
</tbody>
</table>

The MRE by project was calculated finding that the values ranged from 0.00 to 5.36 for the N&C sample, and from 0.00 to 12.0 for the Reused sample. When a high inaccuracy in the MRE was identified in a project (more than 100%), the corresponding practitioner was summoned in order to analyze the discrepancy. Some of the projects were excluded as they were considered outliers once practitioners admitted an error in their interpretation of the project requirements. Of the N&C sample, 654 projects with an MRE from 0.00 to 1.10 were kept, and 22 projects were excluded, as they were considered outliers. On the other hand, 600 projects from the reused code sample had values from 0.00 to 1.06 and were kept, whereas 40 projects considered outliers were excluded. Hence, the final sample sizes were 654 for projects using N&C as independent variable and 600 projects when the lines of the code measured were of the reused type (Table IV).

Table 4

<table>
<thead>
<tr>
<th>Type of lines of code recorded</th>
<th>Original sample sizes</th>
<th>Number of Outliers</th>
<th>Final sample sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>New and changed (N&amp;C)</td>
<td>676</td>
<td>22</td>
<td>654</td>
</tr>
<tr>
<td>Reused</td>
<td>640</td>
<td>40</td>
<td>600</td>
</tr>
</tbody>
</table>

4. Results

An ANOVA of MRE for the N&C and for the Reused code samples are shown in Table V and table VI, respectively. These two tables show that there is a statistically significant difference between the accuracy of size estimation for the two programming languages (p-values from Tables V and VI are less than 0.05). Figures 1 and 2 graphically show the difference in means for the two languages for N&C and Reused code, respectively. These two figures show that projects coded in Java had a better size estimation accuracy than the projects coded in C++, both for N&C and for Reused lines of code.

Table 5

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum of squares</th>
<th>Degrees of freedom</th>
<th>Mean square</th>
<th>F-ratio</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between groups</td>
<td>0.1922</td>
<td>1</td>
<td>0.19229</td>
<td>3.88</td>
<td>0.0489</td>
</tr>
<tr>
<td>Within groups</td>
<td>32.3272</td>
<td>652</td>
<td>0.04958</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>32.5195</td>
<td>653</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6
ANOVA for MRE by programming language (Reused code)

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum of squares</th>
<th>Degrees of freedom</th>
<th>Mean square</th>
<th>F-ratio</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between groups</td>
<td>0.6663</td>
<td>1</td>
<td>0.66639</td>
<td>14.03</td>
<td>0.0002</td>
</tr>
<tr>
<td>Within groups</td>
<td>28.4109</td>
<td>598</td>
<td>0.04750</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>29.0773</td>
<td>599</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Means plot for MRE (N&C code)

Figure 2. Means plot for MRE (Reused code)

The validity of an ANOVA depends on the observance of its three assumptions of residuals, and so they were analyzed for each sample as follows:

1) Independent samples: In this study, groups of practitioners were made up of separate developers and each of them developed their own projects, hence the data are independent.

2) Equal standard deviations: In a plot of this kind the residuals should fall roughly in a horizontal band centered and symmetric about the horizontal axis (as shown in Figures 3 and 4), and

3) Normal populations: A normal probability plot of the residuals should be roughly linear (as shown in Figures 5 and 6).
Considering that the normal probability plots shown in Figures 5 and 6 suggested a slight abnormality, a Kruskal-Wallis test was done for MRE. Since the p-value of Tables VII and VIII are less than 0.05, there is a statistically significant difference amongst the medians at the 95.0% confidence level.

Table 7
Kruskal-Wallis test for MRE by language (N&C code)

<table>
<thead>
<tr>
<th>Language</th>
<th>Sample Size</th>
<th>Average Rank</th>
<th>Test statistic</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>294</td>
<td>345.905</td>
<td>5.07001</td>
<td>0.02434</td>
</tr>
<tr>
<td>Java</td>
<td>360</td>
<td>312.471</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8
Kruskal-Wallis test for MRE by language (Reused code)

<table>
<thead>
<tr>
<th>Language</th>
<th>Sample Size</th>
<th>Average Rank</th>
<th>Test statistic</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>267</td>
<td>332.818</td>
<td>16.7579</td>
<td>0.000042</td>
</tr>
<tr>
<td>Java</td>
<td>333</td>
<td>274.387</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Conclusion and Future Research

This research measured the influence of two object-oriented programming languages on software project size when it is measured in lines of code and whose requirements have been specified in natural language. We present two analyses: one related to new and changed code, and the second one involving reused code. All software projects were individually developed based upon requirements that have been specified in natural language. We present two analyses: one related to new and changed code, and the second one involving reused code. All software projects were individually developed based upon an experimental design in a controlled environment.

After a statistical analysis based on parametric and nonparametric tests, the hypotheses accepted were the following:

H1: There is statistically significant difference in the size estimation accuracy of new and changed code between the projects coded in C++ and those coded in Java when the requirements of the projects are specified in natural language.

H2: There is statistically significant difference in the size estimation accuracy of reused code between the projects coded in C++ and those coded in Java when the requirements of the projects are specified in natural language.

Once the means plots were analyzed, we can conclude the following:

When requirements have been specified in natural language, the size of the projects coded in Java were better estimated by practitioners than those coded in C++, either measured in new and changed or reused lines of code.

These results could be useful in order to select Java over C++ with the goal of improving the accuracy of effort, schedule, cost, or risk estimation when small project size is used as an independent variable in an estimation model. However, this preference of Java over C++ should only be done when the software project application could equally be coded in either of these two languages.

Future research will be approached with the aim of answering the following research questions:

- Is there any statistically significant difference in the size estimation accuracy amongst other programming languages when the software project requirements are specified in natural language?
- Is there any statistically significant difference in the size estimation accuracy amongst C++ and Java programming languages when the software project requirements are specified in semiformal languages (e.g. UML)?
- Is there any statistically significant difference in the size estimation accuracy amongst other programming languages when the software project requirements are specified either in semiformal languages or formal languages (e.g. Z)?

Acknowledgements

Authors of this paper would like to thank CUCEA of Guadalajara University, Jalisco, México, Consejo Nacional de Ciencia y Tecnología (Conacyt), Programa de Mejoramiento del Profesorado (PROMEP), as well as to Programa Integral de Fortalecimiento Institucional (PIFI).

References

Appendix A

The description of each of the projects developed by practitioners included its requirements specification in natural language. The requirements specification for one of the eighteen projects was the following (the other 17 are not included due to space restriction; however they can be consulted in [27] [5]):

1) Develop a software project to calculate the standard deviation of a set of $n$ real numbers.
2) The software project must read the $n$ real numbers from the keyboard.
3) A variable or static array(s), database, or other data structure(s) may be used to hold the data.

The formula for standard deviation, $\sigma$, is the following:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - x_{avg})^2}{n-1}}$$

The formula for calculating the mean is the following:

$$x_{avg} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Where:
- $\Sigma$ is the symbol for summation
- $i$ is an index to the $n$ numbers
- $x$ is the data in the set
- $n$ is the number of items in the set