Comparison of software development productivity based on object-oriented programming languages

Cuauhtémoc López-Martín[1], Arturo Chavoya[2], and María Elena Meda-Campaña[3]
[1, 2,3] Information Systems Department, CUCEA, Guadalajara University, Jalisco, Mexico
[1] cuauhtemoc@cucea.udg.mx, [2] achavoya@cucea.udg.mx, [3]emeda@cucea.udg.mx

Abstract - *The reasons for measuring software productivity are to identify how to reduce software development costs, improve software quality, and improve the rate at which software is developed. In this paper, a data set of 572 software individual projects developed from 2005 to 2010 with practices based on a process specifically designed to laboratory learning environments (Personal Software Process) is used to know if there is any statistically significance difference between the productivity of developers whose projects were written using the object oriented programming languages C++ and Java. Results suggest that there is difference between projects developed in these two programming languages when software projects have been developed in a disciplined way in a laboratory learning environment.*

Keywords: Software development productivity, Object oriented programming languages, Empirical software engineering.

1 Introduction

The abstraction describes on which level the measurements software projects are carried out; there exist the following five abstraction levels [8]: organization, process, project, individual and task. This study is related to the individual level once software projects were individually developed by practitioners.

There are at least the following four options to collect as well as to report software production data [16]: developer self report, project or team manager, outside analysts or observers, and automated performance monitors. This study was related to the first option. In order to reduce bias, each developer used the same personal practices based upon Personal Software Process (PSP). The PSP was selected because the levels of software engineering education and training could be classified in the small as well as in the large software projects [1]. In the case of small software projects, the PSP whose practices and methods are used for delivering quality products on predictable schedules can be useful [6]. Moreover, it has been suggested that a higher productivity over the entire system life cycle use to be associated with the use of a disciplined programming methodology [2]; even, productivity increases when extensive use of modern programming practices are applied (as top-down design, modular design, design reviews, code inspections, and quality-assurance programs) [17].

In accordance with the use of PSP for gathering data, some previous researches have approached their efforts to PSP automate [9] [14] and yet others have incorporated PSP concepts into their programming courses [11].

There have been diverse measures of productivity ([5] [16] [18]), in them have been indicated that the measure of productivity most commonly used is that of size over effort productivity = size / effort. That is the one used in this study; the size is measured using number of lines of code developed by unit of effort.

There have been two main directions on the study of productivity in software engineering literature [18]: (1) researches have been focused on the measure or estimation of productivity, and (2) emphasis has been laid on the discovery of methods or significant factors for productivity improvement. The approach of this study is related to second direction.

Because of the type of programming language is one of the two main factors found having significantly influence on the productivity [7], the sample of this study integrates 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 development productivity between the projects coded in C++ and those coded in Java when the projects are developed in a disciplined way in a laboratory learning environment.

H_2: There is not a statistically significant difference in the development productivity between the projects coded in C++ and those coded in Java when the projects are developed in a disciplined way in a laboratory learning environment.

1.1 Software measurement

Measures of source code size can be classified in physical source lines and in logical source lines [13]. The count of physical lines gives the size in terms of the physical length of the code as it appears when printed. Lines of code have been used by previous researches focused on productivity analysis of large projects [3] [4] [10] [16] [17] and even more recently when productivity has been related to individual projects [15].

In this study, the independent variable in the prediction models is New and Changed (N&C) code and it is considered as physical lines of code (LOC). N&C is composed of added and modified code [6]. The added code is the LOC written during the current programming process, while the modified code is the LOC changed in the base software project when modifying a previously developed project. The base project is the total LOC of the previous project while the reused code is the LOC of previously developed projects that are used without any modification.

A coding standard should establish a consistent set of coding practices that is used as a criterion when judging the quality of the produced code [6]. Hence, it is necessary to always use the same coding and counting standards. The software projects developed of this study followed such guidelines.

2 Experimental design

Because measuring software productivity presupposes an ability to construct a measurement project comparable to those employed in experimental designs for behavioral studies, it is necessary to insure that the measures employed are reliable, valid, accurate, and repeatable. It means that to measure software production implies understanding of the relationship between measurement and instrumentation employed to collect and measure data [16]. Hence, in this paper data collected were related to the same instruments (logs), phases, and standards suggested by PSP.

The experiment was done inside a controlled environment having the following characteristics:

1. All of the developers were experienced, working on software development inside their enterprises at which they were working; however, no one of them had received a course related to personal practices for developing software at individual level.
2. All developers were studying a postgraduate program related to computer science.
3. Each developer wrote seven project assignments. However, only four of them were selected from each developer. The first three projects were not considered because they had differences in their process phases and logs, whereas in latest four projects phases are the same: plan, design, design review, code, code review, compile, testing and post-mortem, and they are based upon the same logs.
4. Each developer selected his/her own programming language whose code standard had the following characteristics: each compiler directive, variable declaration, constant definition, delimiter, assign sentence, as well as flow control statement was written in a line of code.
5. Developers had already received at least one formal course about the object oriented programming language of their choice and they had good programming experience in that language. The sample of this study reduced the bias because it only involved developers whose projects were coded in C++ or Java. One reason for selecting these kinds of languages is because object-oriented languages facilitate high productivity [16].
6. As this study was an experiment with the aim to reduce bias, we did not inform developers about our experimental goal.
7. Developers filled out an Excel sheet for each task and submitted it electronically for examination.
8. Each PSP course had a group of fifteen developers or less.
9. All of developers coincided with the counting standard depicted in Table 1.
10. Developers were constantly supervised and advised about the process.
11. The code written in each project was designed so to be reused in subsequent projects.
12. The developed projects had complexity similar to those suggested in the original PSP [6]. From a set of 18 individual projects, a subset of seven was randomly assigned to each of all developers. A brief description by project is the following:

- Estimating the mean of a sample of *n* real numbers.
- Estimating the standard deviation of a sample of *n* real numbers.
- Matrix addition integrated by real numbers.
- Summing the diagonal of a real numbers square matrix.
- Translating from a quantity to letters.
- Calculating the correlation (*r*) between two series of real numbers.
- Computing the linear regression equation parameters *a* and *b* ($y=a+bX$).
- Calculating *z-values* from a sample of real numbers.
- Calculating the size of a sample.
- Calculating the *y-values* from a sample of real numbers using the normal distribution equation.
- Calculating the estimation standard error (from $y=a+bX$).
- Calculating the coefficient of determination (r^2) from a linear regression equation.
- Calculating both upper and lower limits from a sample of real numbers based upon its standard deviation and mean
- Calculating the coefficient of variation from a distribution.
- Estimating the values based upon statistical empirical rule.

- Counting the physical lines of code of a software project omitting comments and blank lines.
- Both storing and searching records from a file.
- Both deleting and modifying records from a file.
- Data used in this study belong from those developers, whose data for all seven exercises were correct, complete, and consistent.

Table 1. Counting standard

1) Count type Physical/logical	Type Physical
2) Statement type a) Executable b) No executable Declarations Compiler directives Comments and Blank lines	Included Yes Yes, one by text line Yes, one by text line No
3) Clarifications *{* and *}*	 Yes

3 Data analysis

Data from 572 individual software projects developed by 143 practitioners between the years 2005 to 2010 were used to be compared in this study (Appendix A). Once the sample of 572 software projects was developed with C++ (288 projects) and Java (284 projects), we analyzed if there was any statistical difference in their productivity values. Table 2 shows that since the p-value of the F-test is less than 0.05, there is a statistically significant difference between the productivity of the two languages at the 95.0% confidence level. This difference result can graphically be observed in the means plot of Figure 1, which shows that those projects coded in Java had a better productivity that those coded in C++ with (28 versus 25 N&C lines of code by hour).

Table 2. ANOVA for productivity by programming language

Source	Sum of squares	Degrees of freedom	Mean square	F-ratio	P-value
Between groups	1689.47	1	1689.4	8.15	0.0045
Within groups	118136.	570	207.25		
Total	119825.	571			

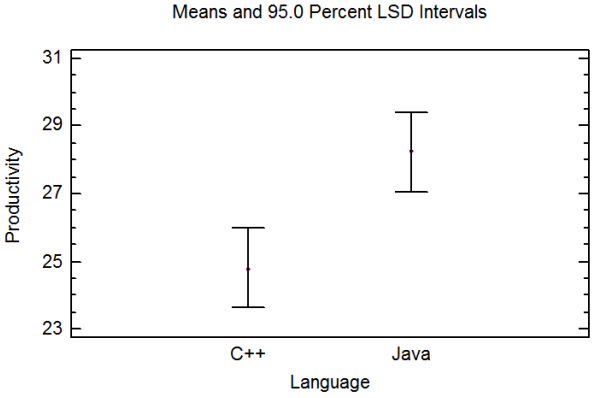

Figure 1. Plot of means for programming languages

The validity of an ANOVA is based on the analysis of the following three assumptions of residuals [12]:

1) Independent samples: Each project was developed independently and by a single practitioner, so 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 Figure 2), and

3) Normal populations: A normal probability plot of the residuals should be roughly linear (as shown in Figure 3).

Hence, the three assumptions for residuals in the productivity data set were considered as met.

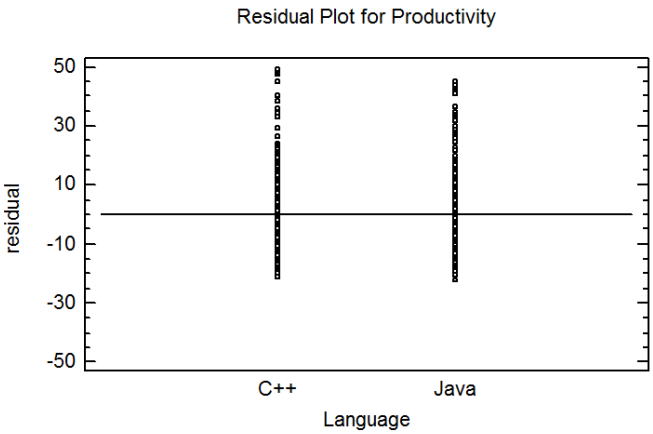

Figure 2. Equal standard deviation plot from programming languages

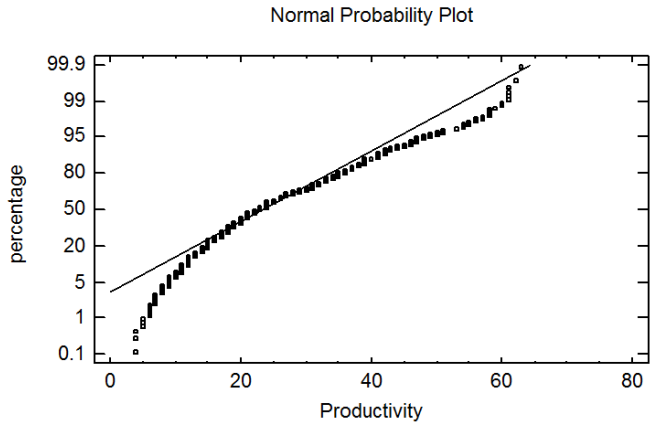

Figure 3. Normality plot from programming languages

4 Conclussions

Owning that there are relevant reasons for measuring software productivity and based upon the assumption that the programming language has influence in the software

development productivity, this study compared software projects code in two object-oriented programming languages. The software projects were developed following a disciplined process in a controlled environment.

After an statistical analysis based upon ANOVA, the accepted hypothesis is the following:

H_1: There is a statistically significant difference in the development productivity between the projects coded in C++ and those coded in Java when the projects are individually developed in a disciplined way in a laboratory learning environment.

After ANOVA, a plot of means showed that projects coded in Java showed a better productivity than those coded in C++. This result was validated based on the three assumptions of residuals.

Future research is related to comparison between these two programming languages when they are used in industrial software projects.

5 References

[1] Bagert D. J., Hilburn T. B., Hislop G., Lutz M., McCracken M., Mengel S. "Guidelines for Software Engineering Education". CMU/SEI-99-TR-032, ESC-TR-99-002, Software Engineering Institute, Carnegie Mellon University. 1999

[2] Bailey, J., Basili, V. "A Meta-Model for Software Development Resource Expenditures". 5th. Intern. Conf. Soft. Engr., IEEE Computer Society, pp. 107-116. 1981

[3] Boehm B., Abts Ch., Brown A.W., Chulani S., Clarck B. K., Horowitz E., Madachy R., Reifer D. & Steece B. "COCOMO II", Prentice Hall. 2000

[4] Cusumano M., Kemerer, C.F. "A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development", Management Science. Pages 1384-1406. 1990

[5] Fenton N.E. & Pfleeger S. L. “Software Metrics: A Rigorous and Practical Approach". PWS Publishing Company. 1997

[6] Humphrey W. "A Discipline for Software Engineering". Addison Wesley. 1995

[7] Jiang, Z., Comstock C. "The Factors Significant to Software Development Productivity". World Academy of Science, Engineering and Technology. Pages 160 – 164. 2007.

[8] Kai, P., 2011. Measuring and predicting software productivity: A systematic map and review. Information and Software Technology, Elsevier, Volume 53, Issue 4, pages 317-343

[9] Johnson, P.M, Kou, H., Agustin, J., Chan, Ch., Moore, C., Miglani, J., Zhen, S., & Doane, E.J. "Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined". Conference on **S**oftware engineering education and training: process and methodology. Pages 641 – 646. 2003.

[10] Lawrence, M.J. "Programming Methodology, Organizational Environment, and Programming Productivity", Journal of Systems and Software, Elsevier. Vol. 2, Pages 257-269. 1981

[11] Maletic J.I., Howald, A., Marcus A. "Incorporating PSP into a traditional software engineering course: an experience report". International Conference of Software Engineering Education and Training. Pp. 89 – 97. 2001.

[12] Montgomery D. C.. "Design and Analysis of Experiments". John Wiley, 2009.

[13] Park, R.E. "Software Size Measurement: A Framework for Counting Source Statements". Software Engineering Institute, Carnegie Mellon University. CMU/SEI-92-TR-020. 1992.

[14] Postema, M. Dick, M., Miller, J., Cuce, S. “Tool Support for Teaching the Personal Software Process". Computer Science Education, Vol. 10, No. 2, Pages 179-193. 2000.

[15] Rombach D., Münch J., Ocampo A., Humphrey W.S., Burton D. "Teaching disciplined software development". Journal Systems and Software, Elsevier, pp. 747- 763.. 2008.

[16] Scacchi, W. "Understanding Software Productivity". International Journal of Software Engineering and Knowledge Engineering. Revised and reprinted in Advances in Software Engineering and Knowledge Engineering, D. Hurley (ed.), Pages 37-70. 1995.

[17] Vosburg, J., Curtis, B., Wolverton, R., Albert, B., Malec, H., Hoben S., Liu, Y. "Productivity Factors and Programming Environments", Proc. 7th. Intern. Conf. Soft. Engr., IEEE Computer Society, Pages 143-152. 1984

[18] Zhizhong, J., Naudé, P., Comstock, C. An Investigation on the Variation of Software Development Productivity, International Journal of Computer and Information Science and Engineering, Pages 72-81. 2007.