Investigación en Progreso: Estudio Comparativo de la Incidencia de los Lenguajes de Programación en la Productividad Informática

Research in Progress: Comparative Study of Programming Languages Incidence in Informatics Productivity

Mauricio R. Dávila [1,2]

1. Programa de Maestría en Ingeniería de Sistemas de Información.
Escuela de Posgrado, Facultad Regional de Buenos Aires. Universidad Tecnológica Nacional. Argentina.
2. Laboratorio de Investigación y Desarrollo en Espacios Virtuales
Departamento de Desarrollo Productivo y Tecnológico. Universidad Nacional de Lanús. Argentina.
davilamr.80@gmail.com

***Resumen*— En la actualidad existe una gran diversidad de lenguajes de programación y a menudo esto dificulta la tarea de seleccionar el lenguaje adecuado para desarrollar una solución determinada. Este estudio se centrara en analizar uno de los factores que afecta la productividad, el referido al lenguaje de programación, siendo el objetivo de la investigación elaborar un procedimiento que permita realizar estudios comparativos sobre la incidencia de los lenguajes de programación en la productividad informática.**

Abstract— There is a great diversity of programming languages nowadays and often this makes difficult the task to select the appropriate language to develop a determined solution. This study will focus on analyzing one of the factors that affect productivity, the one referring to the programing languages, being the research's aim build a procedure that allows making comparative studies on the incidence of programming languages in computer productivity.

***Palabras clave*— productividad, programación.**

***Index Terms*— productivity, programming.**

I. JUSTIFICACIÓN

Tradicionalmente , la productividad ha sido definida como la relación de la salida producida por unidad de entrada [28]; donde los insumos son los recursos necesarios para producir los productos obtenidos. Según el diccionario de la Real Academia Española [40], la definición económica de la productividad refiere a un concepto que describe la relación entre lo producido y los medios empleados para hacerlo. El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) ha publicado en 1992 un estándar para las métricas de productividad de software [24] allí define la productividad como la relación entre una salida primitiva (líneas de código fuente , puntos de función o documentos) y su correspondiente entrada primitiva (esfuerzo , por ejemplo, horas hombre). El concepto de entradas y salidas aparecen tanto en las definiciones de productividad referidas a la Ingeniería del Software como en las definiciones tradicionales.

Es aceptado que factores tales como: las limitaciones de tiempo, los requisitos de fiabilidad, los lenguajes de programación, el tamaño del equipo de desarrollo, la volatilidad de los requisitos, la habilidad del personal con las herramientas, la disponibilidad de personal, la participación del cliente, y la duración del proyecto influyen en la productividad [15-25-33]. A pesar de su aceptación, la influencia, positiva o negativa, de algunos de estos factores en la productividad no es clara.

Este estudio se centrara en analizar uno de los factores que afecta la productividad, el referido al lenguaje de programación empleado, un lenguaje de programación sirve como un marco dentro del cual un programador organiza sus ideas acerca de los procesos y es el lenguaje quien proporciona los medios que permiten articular soluciones simples en pos de resolver problemas complejos [2]. Se considera que existe la necesidad de realizar estudios comparativos sobre la influencia en la productividad de los lenguajes de programación utilizados para desarrollar artefactos de software.

II. FUNDAMENTOS

Tomando como punto de partida la década de 1940 donde surgen las primeras computadoras modernas hasta la actualidad, se han creado una gran cantidad de leguajes de programación de alto nivel. Kinnersley [30] realiza un trabajo de recopilación que reúne información de más de dos mil quinientos lenguajes. En la enciclopedia en línea Wikipedia [53] se encuentra un articulo que reúne información de mas de seiscientos cincuenta lenguajes de programación. Muchos de estos lenguajes de programación han quedado en desuso (ALGOL 60, CPL, FACT, entre otros) [45-31], algunos de los surgidos muchos años atrás han evolucionado e incorporado nuevas características como es el caso de Cobol [9] o Fortran [14], también han surgido nuevos lenguajes que han logrado un lugar en el mercado como ser Erlang [12], JavaScript [26], Java [27] o Python [39] y continúan surgiendo otros que buscan posicionarse como es el caso de Go [18], Hack [20] o Swift [48]. Para determinar que lenguajes se encuentran en uso, se

puede tomar como parámetro que, al día de hoy, en el repositorio GitHub existe código escrito en casi trecientos lenguajes de programación distintos [55] y que poco más de doscientos cuarenta lenguajes son tomados en cuenta para elaborar el índice TIOBE [49] el cual mide el volumen de información disponible en la web sobre cada lenguaje. Michael Scott [46] entiende que la existencia de tantos lenguajes de programación puede responder a tres motivos:

- Evolución: La informática es una disciplina joven; en constante búsqueda de mejores formas de hacer las cosas.
- Propósitos Especiales: Muchos lenguajes fueron diseñados para un dominio del problema específico.
- Preferencia personal: A diferentes personas les gustan cosas diferentes.

A pesar de existir una gran cantidad de lenguajes no todos son ampliamente utilizados. La IEEE elabora un índice denominado "The Top Programming Languages 2015" [8], formado por una lista formada por tan solo cuarenta y ocho lenguajes considerados como los mas populares. En su libro Scott [46] considera que los siguientes factores influyen en la popularidad de un lenguaje de programación:

- Potencia expresiva: tienen un gran impacto en la capacidad del programador para escribir código claro, conciso y fácil de mantener.
- Facilidad de uso para el usuario principiante: en muchos casos el éxito se debe a una muy baja "curva de aprendizaje".
- Facilidad de implementación: Algunos lenguajes son exitosos ya que se pueden implementar con facilidad en diversos dispositivos.
- Normalización: Casi todos los lenguajes ampliamente utilizado tiene una norma internacional oficial o (en el caso de varios lenguajes de scripting) una sola aplicación canónica.
- Código abierto: La mayoría de los lenguajes de programación de hoy tienen al menos un compilador/ intérprete de código abierto.
- Compiladores/Intérpretes: muchos lenguajes tienen éxito, en parte porque tienen compiladores y herramientas de apoyo que hacen un buen trabajo.
- Economía, patrocinio, y la inercia: Por último, hay otros factores además de los méritos técnicos que influyen en gran medida el éxito. El respaldo de un poderoso patrocinador es uno.

En este escenario, con una gran diversidad de lenguajes de programación, algunos mas populares que otros, se dificulta la tarea de seleccionar el lenguaje adecuado para desarrollar una solución determinada. Algunos estudios tratan de determinar cual es el lenguaje adecuado para resolver un problema mediante la comparación de características de desempeño como ser: uso de CPU, uso de Memoria o Tiempos de ejecución [16-35-37-38]. Pero dejan de lado que características como las antes mencionadas son sólo relevantes en entornos en los cuales los recursos disponibles son limitados y es posible llegar conclusiones del estilo: "El lenguaje A demora menos en realizar una tarea X que el lenguaje B", esta afirmación carece de validez en entornos de "Cloud Computing" [3-17], en los cuales los recursos son prácticamente ilimitados. La computación en la nube es un paradigma, a esta altura, muy consolidado para el aprovisionamiento de servicios bajo demanda, a este concepto se lo conoce con el nombre de elasticidad y refiere a la capacidad de agregar y quitar recursos "sobre la marcha" para manejar la variación de la carga [10].

Otras características que suelen ser tenidas en cuenta a la hora de comparar lenguajes son las constructivas [2-22-46-47-52], aquellas que indican como es el lenguaje y como éste realiza las cosas:

- Forma de ejecución, nativa o en una máquina virtual.
- Manejo de la memoria.
- Manejo de errores.
- Plataformas que lo soportan.
- Ecosistema de las bibliotecas.
- Paradigma.

En muchos estudios se hacen referencia a las métricas que se pueden establecer sobre el código producido con un determinado lenguajes de programación [4-21-23-32-34-41-42], en el caso de pretender utilizarlas para determinar si un lenguaje es superior a otro, no se debe perder de vista que dichas métricas solo se enfocan en el código resultante pero no tienen en consideración las características del programador que elaboró el código que se esta evaluando y por lo tanto se corre el riesgo de sobre calificar o sub calificar un lenguaje por un falencia o virtud del programador. Abdel-Hamid [1] identifica dos factores que afectan tanto a la calidad como a la productividad del software; (I) las características de la tarea (es decir, la naturaleza compleja de una tarea) y (II) los recursos del equipo (es decir, las habilidades del programador). El impacto del capital humano es tenido en cuenta en métodos de estimación como ser COCOMO [7] que desde su primera versión identifica cinco factores relacionados con las capacidades del programador en el desarrollo de software, estos factores de ajuste se mantienen en versión actual de método.

III. Objetivo

Los artefactos de software se consideran como partes pequeñas y manejable de un proyecto de software. Tsui [51] los describe como la "unidad material" que puede estar en cualquier forma tal como la documentación o el código fuente. Mediante el código fuente un programador especifica las acciones a ser realizadas por un computador, utilizando para ello la sintaxis de algún lenguaje de programación.

Este trabajo se centrara en los artefactos conocidos como código fuente siendo el objetivo de la investigación elaborar un procedimiento que permita realizar estudios comparativos sobre la incidencia de los lenguajes de programación en la producción de artefactos software.

Surgen como objetivos específicos de la investigación por un lado el desarrollo de instrumentos que permitan evaluar las aptitudes de los programadores (sección A) y por el otro la definición de métricas que permitan determinar la incidencia en la productividad de los distintos lenguajes (sección B).

A. Desarrollo de instrumentos

En la literatura, las formas de medir o controlar el nivel o la experiencia que tiene un programador varían y a menudo, los investigadores descuidan o no especifican cómo lo han controlado [13]. Para poder llevar adelante una comparación de productividad de manera objetiva, se deberá contar con instrumentos que permitan evaluar las aptitudes de los programadores, para lo cual se elaborará un protocolo que permita desarrollar instrumentos de categorización de los programadores. El método de calificación estará basado en el trabajo de Raphael Poss de la Universidad de Amsterdam quien propone una tabla para caracterizar el nivel de competencia de los programadores para un lenguaje en particular [36].

B. Definición de Métricas

La medición de la productividad debe expresar cuán bien son aplicadas las competencias (utilizando materiales y equipos) para producir productos y servicios dentro de un período de tiempo especificado [29]. Teniendo en cuenta las diferentes actividades que realiza un programador en el proceso de desarrollo de artefactos de software, se deberá establecer el conjunto de métricas y procedimientos a ser utilizadas a efectos de determinar la incidencia de un lenguaje de programación en la productividad.

IV. Metodología de Desarrollo

Para construir el conocimiento de la presente investigación, se seguirá un enfoque de investigación clásico [43-11] con énfasis en la producción de tecnologías [44], a continuación se identifican los métodos (Sección A) y materiales necesarios para desarrollar el proyecto (Sección B).

A. Métodos

Los métodos que se llevarán a cabo en el presente trabajo se enumeran a continuación:

1) Revisiones sistemáticas

Las revisiones sistemáticas de artículos científicos siguen un método explícito para resumir la información sobre determinado tema o problema; se diferencia de las revisiones narrativas en que provienen de una pregunta estructurada y de un protocolo previamente realizado [5].

2) Prototipado evolutivo experimental

El prototipado evolutivo experimental, método de la ingeniería [6], consiste en desarrollar una solución inicial para un determinado problema, generando su refinamiento de manera evolutiva por prueba de aplicación de dicha solución a casos de estudio (problemáticas) de complejidad creciente. El proceso de refinamiento concluye al estabilizarse el prototipo en evolución.

3) Método de Wilcoxon

El Método de Wilcoxon permite analizar la similitud entre un conjunto de datos muestrales apareados, donde cada elemento de la población posee un valor experimental que se desea comprobar y un valor de referencia o de control [19]. Como resultado de la prueba, es posible aceptar o refutar una hipótesis nula (H0) la cual indica que la población de diferencias entre los valores experimentales y los valores de control presentan una diferencia significativa [50].

Se prevé utilizar el Método de Wilcoxon [54] en esta trabajo por cuanto se busca contrastar la distribución de un conjunto de variables que caracterizan un grupo tipo de programadores, a efecto de determinar si las mismas presentan diferencias significativas entre el grupo que utiliza el lenguaje A y el que utiliza el lenguaje B.

B. Materiales

Conjunto de ejercicios de programación de complejidad creciente susceptibles de ser resueltos utilizando los lenguajes de programación considerados.

C. Metodología

Para alcanzar los Objetivos trazados se propone:

i. Realizar una investigación documental exploratoria sobre las métricas utilizadas para determinar productividad en la ingeniería de software, identificar caso de estudio.
ii. Realizar una investigación documental exploratoria sobre los aspectos relevantes a la hora de categorizar a un programador, identificar caso de estudio.
iii. Desarrollar mediante el método de prototipado evolutivo un protocolo que permita valorar elementos a ser utilizados para categorizar a los programadores
iv. Desarrollar mediante el método de prototipado evolutivo un procedimiento que permita elaborar los problemas a ser resueltos por los participantes del estudio a efectos de poder evaluar la productividad.
v. A partir de los problemas desarrollados y los grupos de trabajo identificados, desarrollar la experiencia de resolución de los problemas tomando las mediciones correspondientes.
vi. Evaluar e interpretar resultados.

Referencias

[1] Abdel-Hamid, T. K., & Madnick, S. E. (1989). Lessons learned from modeling the dynamics of software development. Communications of the ACM , 32 (12), 1426–1438

[2] Abelson, H., Sussman, G. J., y Sussman, J. (1996). Structure and interpretation of computer programs. Justin Kelly.

[3] Amazon. (2016). Amazon Cloud Computing. (Amazon, Producer) Consultado el 9 de julio de 2016, disponible en: http://aws.amazon.com

[4] ANSI/IEEE, (2007). Draft IEEE Standard for software and system test documentation. ANSI/IEEE Std P829-2007.

[5] Argimón, J. (2004). Métodos de Investigación Clínica y Epidemiológica. España: Elsevier.

[6] Basili. (1993). The Experimental Paradigm in Software Engineering. En Experimental Software Engineering Issues: Critical Assessment and Future Directions (Ed. Rombach, H., Basili, V., Selby, R.). Lecture Notes in Computer Science , 706.

[7] Boehm, B. W. (1981). Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.

[8] Cass, N. (2015). The Top Programming Languages, 2015. Consultado el 8 de julio de 2016, disponible en: http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015

[9] Cobol. (2016). (IBM) Consultado el 8 de julio de 2016, disponible en: http://www-03.ibm.com/software/products/en/cobocompfami

[10] Coutinho, E. F., de Carvalho Sousa, F. R., Rego, P. A. L., Gomes, D. G., & de Souza, J. N. (2015). Elasticity in cloud computing: a survey. annals of telecommunications-annales des télécommunications, 70(7-8), 289-309.

[11] Creswell, J. (2002). Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research. Prentice Hall. ISBN 10: 01-3613-550-1.

[12] Erlang. (2016). Erlang programming language. (Ericson) Consultado el 10 de julio de 2016, disponible en: http://www.erlang.org/

[13] Feigenspan, J., Kästner, C., Liebig, J., Apel, S., & Hanenberg, S. (2012, June). Measuring programming experience. In Program Comprehension (ICPC), 2012 IEEE 20th International Conference on (pp. 73-82). IEEE.

[14] Fortran. (2016). (The Fortran Company) Consultado el 8 de julio de 2016, disponible en: http://www.fortran.com/

[15] Foulds, L. R., Quaddus, M., & West, M. (2007). Structural equation modelling of large-scale information system application development productivity: the Hong Kong experience. 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007) (pp. 724-731). IEEE.

[16] Fulgham, B., y Gouy, I. (2009). The computer language benchmarks game. Consultado el 9 de julio de 2016, disponible en: http://benchmarksgame.alioth.debian.org

[17] Google Cloud. (2016). Google Cloud Platform. Consultado el 9 de julio de 2016, disponible en: http://cloud.google.com

[18] Go (2016). Go Programming Language Consultado el 9 de diciembre de 2016, disponible en: https://golang.org

[19] Guardía-Olmos, J., Freixa-Blanxart, M., Peró-Cebollero, M. y Turbany Oset, J. (2007). Análisis de Datos en Psicología. Delta Publicaciones.

[20] Hack. (2016). (Facebook Inc.) Consultado el 8 de julio de 2016, disponible en: http://hacklang.org/

[21] Halstead, M. (1977). Elements of Software Science. Elsevier Science Inc., New York, NY, USA.

[22] Harrison, L., Smaraweera, M., Lewis, D., y Lewis, P. (1996). Comparing Programming Paradigms: An Evaluation of Functional and Object-Oriented Programs. Softw. Eng. Journal , 11 (4), 247-254.

[23] Hernández-López, A., Colomo-Palacios, R., García-Crespo, Á., & Cabezas-Isla, F. (2013). Software engineering productivity: Concepts, issues and challenges. . Perspectives and Techniques for Improving Information Technology Project Management.

[24] IEEE, Std 1045. (1992). IEEE Standard for Software Productivity Metrics. Institute of Electrical and Electronics Engineers .

[25] IEEE, (1997). IEEE Standard for Developing Software Life Cycle Processes. IEEE Std 1074-1997 (Revision of IEEE Std 1074-1995; Replaces IEEE Std 1074.1-1995)

[26] JavaScript. (2016). (Mozilla Developer Network) Consultado el 20 de diciembre de 2016, disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[27] Java. (2016). (Oracle) Consultado el 8 de julio de 2016, disponible en: https://www.oracle.com/java

[28] Jefferys, J., Hausberger, S., & Lindblad, G. (1954). Productivity in the distributive trade in Europe: wholesale and retail aspects. Organisation for European Economic Co-operation .

[29] Jun, Y. (2007). Towards Adaptive Project Tracking Using Individual Productivity Metrics. Proceedings of the 6th International Conference on Machine Learning and Cybernetics, (pp. 19-22). Hong Kong.

[30] Kinnersley, B. (2016). The Language List. Consultado el 24 de abril de 2016, disponible en: The Language List: http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

[31] Lévénez, É. (2015). Computer Languages Timeline. Consultado el 8 de julio de 2016, disponible en: https://www.levenez.com

[32] Lorenz, M., y Kidd, J. (1994). Object-Oriented Software Metrics. Englewood Cliffs, NJ, USA: Prentice Hall.

[33] Maxwell, K. D., & & Forselius, P. (2000). Benchmarking software development productivity. 17 (1), 80-88.

[34] McCabe, T. J. (1976). A complexity measure. IEEE Transactions on software Engineering , 4, 308-320.

[35] Nanz, S., y Furia, C. A. (2015). A comparative study of programming languages in Rosetta Code. Proceedings of the 37th International Conference on Software Engineering. 1, pp. 778-788. IEEE Press.

[36] Poss, R. (2014). How good are you at programming?. Consultado el 7 de julio de 2016, disponible en: http://science.raphael.poss.name/programming-levels.pdf

[37] Prechelt, L. (2010). Two Comparisons of Programming Languages. Making Software: What Really Works, and Why We Believe It, 239-258.

[38] Prechelt, L. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. IEEE Computer , 33 (10), 23-29.

[39] Python. (2016). (Python Software Foundation) Consultado el 8 de julio de 2016, disponible en: https://www.python.org/

[40] RAE. (2016). Diccionario de la lengua española. Fuente electrónica. Consultado el 9 de julio de 2016, disponible en: http://dle.rae.es

[41] Riaz, M., Mendes, E., & Tempero, E. (2009,). A systematic review of software maintainability prediction and metrics. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (pp. 367-377). IEEE Computer Society.

[42] Rilling, J., y Klemola, T. (2003). Identifying Comprehension Bottlenecks Using Program Slicing and Cognitive Complexity Metrics. 10th IEEE Working Conference on Reverse Engineering. 10, pp. 115-125. Oregon, USA: IEEE.

[43] Riveros, H., y Rosas, L. (1985). El Método Científico Aplicado a las Ciencias Experimentales. México: Editorial Trillas.

[44] Sábato, J., y Mackenzie, M. (1982). La Producción de Tecnología. México: Editorial Nueva Imagen.

[45] Sammet, J. E. (1972). Programming languages: history and future. (Vol. 15). Communications of the ACM.

[46] Scott, M. (2009). Programming language pragmatics. Amsterdam Boston: Elsevier/Morgan Kaufmann Pub.

[47] Sebesta, R. (2012). Concepts of programming languages. Boston: Pearson.

[48] Swift. (2016). (Apple Inc.) Consultado el 8 de julio de 2016, disponible en: https://developer.apple.com/swift/

[49] TIOBE. (2016). (TIOBE Software BV.) Consultado el 8 de julio de 2016, disponible en: http://www.tiobe.com/tiobe_index

[50] Triola, M. 2013. Estadística. Pearson Educación, 11 edición.

[51] F. Tsui, Managing software projects. Jones & Bartlett Learning, 2004.

[52] Watt, D. A. (2004). Programming language design concepts. John Wiley & Sons.

[53] Wikipedia (2016) List of programming languages. Consultado el 24 de abril de 2016, disponible en: https://en.wikipedia.org/wiki/List_of_programming_languages

[54] Wilcoxon, F. 1945. Individual Comparisons by Ranking Methods, Biometrics 1, pp. 80-83.

[55] Zapponi, C. (2016). Languages at GitHub. Consultado el 8 de julio de 2016, disponible en: http://githut.info/

Mauricio R. Dávila Es Licenciado en Informática por la Universidad Atlántida Argentina. Es Candidato del Programa de Magister en Ingeniería de Sistemas de Información de la Escuela de Postgrado de la Facultad Regional Buenos Aires de la Universidad Tecnológica Nacional. Es Coordinador Académico de la Tecnicatura Superior en Programación y de la Tecnicatura Superior en Sistemas Informáticos de la Facultad Regional Avellaneda de la Universidad Tecnológica Nacional. Es Docente de las asignaturas Programación I y Laboratorio de Computación I de la Tecnicatura Superior en Programación de la Facultad Regional Avellaneda de la Universidad Tecnológica Nacional. Es Investigador Tesista del Laboratorio de Investigación y Desarrollo en Espacios Virtuales de Trabajo del Departamento de Desarrollo Productivo y Tecnológico de la Universidad Nacional de Lanús.