Facultad de Ingeniería
ISSN: 0121-1129
ISSN: 2357-5328
revista.ingenieria@uptc.edu.co
Universidad Pedagógica y Tecnológica de Colombia
Colombia

Echavarría-Flórez, Ingrid-Sofía; Restrepo-Calle, Felipe
Métricas de legibilidad del código fuente: revisión sistemática de literatura
Facultad de Ingeniería, vol. 29, núm. 54, 2020, -Marzo
Universidad Pedagógica y Tecnológica de Colombia
Colombia

DOI: https://doi.org/10.19053/01211129.v29.n54.2020.11756

Disponible en: https://www.redalyc.org/articulo.oa?id=413962511031

Software Readability Metrics: A Systematic Literature Review

Ingrid-Sofía Echavarría-Flórez; Felipe Restrepo-Calle

Citación: I.-S. Echavarría-Flórez, F. Restrepo-Calle, “Software Readability Metrics: A Systematic Literature Review,” *Revista Facultad de Ingeniería*, vol. 29 (54), e11756, 2020.
https://doi.org/10.19053/01211129.v29.n54.2020.11756

Recibido: Junio 31, 2020; **Aceptado:** Septiembre 17, 2020;
Publicado: Septiembre 18, 2020

Conflicto de intereses: Los autores declaran no tener conflicto de intereses.

Software Readability Metrics: A Systematic Literature Review

Ingrid-Sofía Echavarría-Flórez[1]

Felipe Restrepo-Calle[2]

Abstract

Software quality is an aspect directly linked to future maintenance costs, and is generally quantified by means of quality metrics of the software products. One of the main aspects to evaluate software quality is its maintainability, since it has a high impact on the total costs of software projects. In particular, it is estimated that 70% of the maintenance time is dedicated to understand the code, so it is important to be able to measure the readability of a source code fragment properly. Readability is defined as the ease with which a person can read and understand a piece of code written by another person, this feature is crucial to facilitate the understanding of the code during software maintenance tasks. In this sense, it is necessary to be able to measure the readability of the source code. Therefore, over the past few years, researchers in the area have proposed multiple metrics to measure the readability of source code. However, it remains a challenge to be able to accurately assess the readability of source code in a widely accepted way. Therefore, it is essential to give continuity to this type of research and, for this, it is necessary to know the most recent advances. This paper presents a synthesis and analysis of code readability metrics, through a systematic review of literature, showing a compilation of the characteristics and methods used for their measurement. Results of this work will be useful for researchers in the area to propose new source code readability metrics.

Keywords: software readability; source code; software quality; maintainability; metrics; software engineering.

[1] M. Sc. Universidad Nacional de Colombia (Bogotá, Colombia). iechavarria@unal.edu.co. ORCID: 0000-0001-9412-2794

[2] Ph. D. Universidad Nacional de Colombia (Bogotá, Colombia). ferestrepoca@unal.edu.co. ORCID: 0000-0003-4226-1324

Métricas de legibilidad del código fuente: revisión sistemática de literatura

Resumen

La calidad del software es un aspecto ligado directamente a los costos futuros de mantenimiento y, generalmente, se cuantifica mediante métricas de calidad de los productos de software. Uno de los aspectos a tener en cuenta para evaluar la calidad del software es su mantenibilidad, ya que tiene un alto impacto sobre los costos totales de los proyectos de software. En particular, se estima que el 70% del tiempo de mantenimiento se destina a comprender el código, por lo que resulta importante poder medir la legibilidad de un fragmento de código fuente adecuadamente. La legibilidad se define como la facilidad con la que una persona puede leer y comprender un fragmento de código escrito por otra persona. Esta característica es crucial para facilitar la compresión del código durante las tareas de mantenimiento de software, por lo que resulta necesario poder medir la legibilidad del código fuente. Por lo tanto, a lo largo de los últimos años, los investigadores en el área han propuesto múltiples métricas para medir la legibilidad del código fuente. No obstante, sigue siendo un reto poder evaluar con precisión la legibilidad del código fuente de una forma ampliamente aceptada. Por ello, es indispensable dar continuidad a este tipo de investigaciones y, para esto, es necesario conocer los avances recientes. Este artículo presenta una síntesis y análisis de las métricas de legibilidad de código, mediante una revisión sistemática de literatura, mostrando una recopilación de las características y los métodos utilizados para su medición. Los resultados de este trabajo serán de utilidad para que los investigadores en el área puedan proponer nuevas métricas de legibilidad del código fuente.

Palabras clave: legibilidad de código; código fuente; calidad de software; mantenibilidad; métricas; ingeniería de software.

Métricas de legibilidade do código fonte: revisão sistemática de literatura

Resumo

A qualidade do software é um aspecto ligado diretamente aos custos futuros de manutenção e, geralmente, quantifica-se mediante métricas de qualidade dos

produtos de software. Um dos aspectos a ter em conta para avaliar a qualidade do software é sua manutenibilidade, já que tem um alto impacto sobre os custos totais dos projetos de software. Em particular, estima-se que 70% do tempo de manutenção destina-se a compreender o código, pelo que resulta importante poder medir a legibilidade de um fragmento de código fonte adequadamente. A legibilidade define-se como a facilidade com a que uma pessoa pode ler e compreender um fragmento de código escrito por outra pessoa. Esta característica é crucial para facilitar a compreensão do código durante as tarefas de manutenção de software, pelo que resulta necessário poder medir a legibilidade do código fonte. Portanto, ao longo dos últimos anos, os pesquisadores na área têm proposto múltiplas métricas para medir a legibilidade do código fonte. Porém, segue sendo um desafio poder avaliar com precisão a legibilidade do código fonte de uma forma amplamente aceita. Por isso, é indispensável dar continuidade a este tipo de pesquisas e, para isto, é necessário conhecer os avanços recentes. Este artigo apresenta uma síntese e análise das métricas de legibilidade de código, mediante uma revisão sistemática de literatura, mostrando uma recopilação das características e os métodos utilizados para sua medição. Os resultados deste trabalho serão de utilidade para que os pesquisadores na área possam propor novas métricas de legibilidade do código fonte.

Palavras chave: legibilidade de código; código fonte; qualidade de software; manutenibilidade; métricas; engenharia de software.

I. INTRODUCCIÓN

La calidad en los productos de software es de gran importancia, principalmente, por los altos costos que implica modificar un programa si es de baja calidad [1]. Dentro de la labor de mantenimiento, los desarrolladores gastan aproximadamente 70% de su tiempo tratando de comprender el código fuente, lo cual podría disminuir si el código fuera más legible [2]. La legibilidad se refiere a la facilidad con la que una persona puede leer y comprender un fragmento de código [1, 3]. Si un producto de software es poco legible, su calidad se considera baja y, por consiguiente, el costo de mantenimiento será mayor y los desarrolladores gastarán más tiempo [1, 4]. Por ello, es importante poder identificar cuáles son las métricas para medir la legibilidad del código con el fin de automatizar su medición [1].

Aún no se cuenta con un método definitivo que permita evaluar la legibilidad de un fragmento de código fuente con confianza [2, 5]. Por lo anterior, es necesario identificar este tipo de características mediante un análisis sistemático de los trabajos relacionados con la medición de la legibilidad del código fuente, ofreciendo así una base sólida para futuras investigaciones en el área. En este contexto, este artículo presenta los resultados obtenidos mediante una revisión sistemática de literatura, entregando una recopilación de las características del código fuente que más influyen en la legibilidad, las métricas utilizadas para su medición, los métodos automáticos de medición, los retos actuales y las aplicaciones de interés. En particular, en este artículo se propone dar respuesta a las siguientes preguntas de investigación:

RQ1. ¿Cuáles son las características que permiten medir la legibilidad del código fuente de un software?

RQ2. ¿Qué métodos son utilizados para obtener el valor de medida de cada una de esas métricas de forma automática?

RQ3. ¿Cuáles son los principales retos y aplicaciones de interés en las investigaciones relacionadas con la legibilidad del código fuente?

El artículo está organizado de la siguiente manera: la Sección II presenta el proceso realizado para llevar a cabo la revisión sistemática de literatura; la Sección III

presenta los resultados y discute las respuestas a cada una de las preguntas de investigación. Por último, la Sección IV concluye el trabajo.

II. MÉTODO

La revisión sistemática de literatura, abreviada como RSL (o SLR en inglés), es un método propuesto por Kitchenham, que permite realizar una síntesis de los estudios científicos de calidad sobre un tema específico o pregunta de investigación [6]. El proceso realizado en la RSL se estructuró en cuatro fases: estrategia de búsqueda, recolección de documentos, revisión de resultados de la búsqueda, y análisis y presentación de resultados.

A. Estrategia de búsqueda

Para la búsqueda se utilizaron las palabras clave: *software*, *source code*, *metric*, *model*, *readability* y *classification*. Las bases de datos bibliográficas utilizadas fueron: WoS - Web of Science, Scopus, Google Scholar, IEEExplore y ACM Digital Library. Los criterios de inclusión utilizados para la selección de documentos fueron: idioma del documento (español o inglés) y fecha de publicación (entre el 1 de enero del año 2015 y el 23 de abril del año 2019). Las cadenas de búsqueda definidas y utilizadas en la búsqueda fueron las descritas en la Tabla 1.

Tabla 1. Cadenas de búsqueda utilizadas para la recolección.

#	Cadenas de búsqueda
1	readability AND (software OR code) AND (metric* OR classification OR model*)
2	metric + readability + software + classification + “source code”

Por otra parte, algunos de los factores de calidad evaluados para los documentos fueron los siguientes: tiene objetivo del estudio; descripción clara del contexto del problema; tiene sección de resultados; es reproducible; describe al menos 1 métrica, 1 método o 1 característica; referencias actualizadas; responde las preguntas de investigación. El formato completo de lista de chequeo utilizado y diligenciado en el proceso de extracción de datos está disponible en el repositorio público de GitHub https://github.com/sofi876/MetricasLegibilidadSW, así como los formatos diseñados y utilizados en el proceso de extracción de datos. Para cada estudio se evaluó si

cumplía con los factores de calidad, y fueron rechazados aquellos que no cumplían con el 80%.

B. Recolección de documentos

En esta fase se realizó el proceso de recolección de documentos, utilizando como guía la estrategia de búsqueda definida anteriormente. Todo el proceso de recolección se documentó de forma rigurosa, indicando la razón de exclusión o inclusión de documentos, usando el gestor de referencias Mendeley.

C. Resultados de la RSL

En la Tabla 2 se muestran los resultados obtenidos al realizar la búsqueda. La columna “Selección 1” se refiere a los documentos seleccionados en un primer filtro. La columna “Selección 2” son los seleccionados después de un segundo filtro más riguroso.

Tabla 2. Resultados cuantitativos de la RSL.

Fuente	Cadena	Resultados	Selección 1	Selección 2
WoS	1	66	13	5
Scopus	1	53	49	18
Google Scholar	2	60 (6 páginas)	27	18
IEEE	1	65	19	10
ACM	1	60	18	8

Posteriormente se eliminaron los duplicados y se obtuvieron 33 documentos únicos. Se realizó un tercer filtro más riguroso, obteniendo 24 seleccionados, pero se adicionaron 6 documentos que, pese a no cumplir con el criterio de fecha de publicación, son importantes porque se consideran trabajos seminales en el área, obteniendo así 30 documentos.

La fase final de análisis y presentación de resultados se llevó a cabo con los 30 seleccionados y se discuten en las siguientes secciones, dando respuesta a las preguntas de investigación.

III. Resultados y Discusión

A. Características y métricas de legibilidad del código fuente

Las características hacen referencia a los aspectos del código fuente que tienen mayor influencia con la percepción de legibilidad. Los autores de los estudios seleccionados evaluaron un grupo de características, y propusieron unas métricas para ellas que permitieran medir la legibilidad del código fuente. En la Tabla 3 se presenta el listado de algunas características encontradas que obtuvieron mejores resultados en relación al juicio de legibilidad. El nivel de relación de la característica con la legibilidad está representado por A (alto), M (medio) y B (bajo). Ese nivel de relación proviene de los resultados obtenidos por los autores en sus estudios, y refleja la importancia de la característica con la legibilidad del código. El listado de características se presenta en orden de relevancia.

Tabla 3. Características del código fuente relacionadas con la legibilidad.

Característica	Nivel de relación
# promedio de espacios en blanco (identación)	M [1], A [7], B [25, 30], A [10], A [12]
# promedio de Comentarios	M [1], A [7], M [25, 30], A [12]
# máximo de espacios en blanco (identación)	B [1], A [7], B [25, 30], A [10]
Líneas de código (LOC)	A [3, 25], A [12]
Longitud promedio de línea (# de caracteres)	A [1], M [25, 30], M [22]
Legibilidad de comentarios CR	M [2, 25], A [18]
# promedio de paréntesis	A [1], M [25, 30]
# promedio de operadores aritméticos	B [1], B [25, 30], A [12]
Volumen del programa	A [12]
# promedio de identificadores	A [1]
# líneas de comentarios por método (LCM)	A [9]
Nombres significativos	A [7]
Control anidado máximo	A [12]
Longitud de los comentarios	A [10]

Por otra parte, las métricas de legibilidad del software se clasificaron en dos tipos: las simples, que consisten en tomar la medida por un conteo de alguna de las características presentadas; y las compuestas, que requieren de una ecuación más elaborada. Las métricas simples encontradas son: número promedio y máximo de

espacios en blanco (identación), número promedio de comentarios, longitud promedio y máxima de línea, número promedio de paréntesis, número promedio de operadores aritméticos, volumen del programa, número promedio de identificadores, número de líneas de comentarios dentro de un método (LCM), número máximo de ciclos anidados, longitud de los comentarios y número promedio de palabras clave. Las métricas compuestas más relevantes se presentan en la Tabla 4.

Tabla 4. Métricas de legibilidad compuestas.

Métrica	Ecuación
SRES: fácil puntaje de legibilidad de software	SRES = ASL - 0.1 AWL; dónde, ASL es la métrica de longitud promedio de sentencia, y AWL es la métrica de longitud promedio de palabra. A mayor SRES, más difícil para leer [13].
PHD: puntuación de legibilidad de Posnett	PHD = 1 / (1 + e^{-z}); donde, Z = 8.87 – 0.033 V + 0.40 LOC -1.5 H, donde, V es el volumen de Halstead, LOC las líneas de código y H es la información Shannon de tokens [5, 13].
Legibilidad cuantitativa optimizada de Choi	Legibilidad = (-0.020) * (LOC) + (0.040) * (# de comentarios) + (0.037) * (# de líneas en blanco) + (-0.755) * (# de operadores bit a bit) + (-0.153) * (control anidado máximo) + (-0.001) * (Volumen de programa) + (-0.611) * (Entropía).
WSCR: legibilidad centrada en servicio web	WSCR(d) = Alcance(d) + Cohesión(d) + DaCw(d). Utilizada con la herramienta WordNet [14].
ITID: términos id en el diccionario. [25]	ITID(I) = |Terminos(I) ∩ Diccionario| / |Terminos(I) |
CR: legibilidad de los comentarios [25]	FK(S) = 206.835 - 1.015*(palabras(S)/frases(S)) - 84.600*(sílabas(S)/Palabras(S)); donde S es el fragmento de código fuente.
Legibilidad de la clase	Legibilidad (c, P) = $\sum_{i \in 1}^{n} \text{Legibilidad}(mi, c) \sum_{i \in 1}^{n}$ (Legibilidad(mi, c)/|M|) [28]
Legibilidad del programa	Legibilidad(P) =$\sum_{i \in 1}^{n}$ (R(ci, p) / |C|) $\sum_{i \in 1}^{n} R(ci, P)$ [28]

De acuerdo con la literatura, las métricas simples que mejores resultados han reportado para medir la legibilidad del código fuente han sido: la cantidad de espacios en blanco, la cantidad de comentarios, la longitud de la línea de código y la cantidad de paréntesis. Además, las métricas de tipo compuestas que mejores resultados han obtenido son: la métrica de Posnett, abreviada como PHD, la métrica WSCR para el código fuente en lenguaje WSDL (servicios web), y la legibilidad de los comentarios (CR) propuesta por Scalabrino et al [25].

B. Métodos utilizados para medir la legibilidad del código fuente

En la Tabla 5 se presentan los métodos utilizados en las investigaciones para diferentes actividades relacionadas, directa o indirectamente, con los procesos que permiten evaluar la legibilidad en el código fuente. En la primera columna se describen los usos de estos métodos identificados en los estudios, en la segunda columna se encuentran los métodos que fueron hallados en los estudios seleccionados en la RSL.

Tabla 5. Métodos utilizados para medir la legibilidad del código fuente.

Uso	Métodos
Clasificador binario con aprendizaje de máquina supervisado [1, 20, 25, 30]	Red neuronal; Red de Bayes; RandomForest; ML perceptron; SMO; Enfoque del intervalo de voto por característica; Aprendizaje profundo; Red neuronal convolucional; Arquitectura de inicio
Identificar características y medir métricas usando aprendizaje de máquina no supervisado [25]	*Clustering* basado en densidad - DBSCAN
Identificar características mediante análisis de regresión / correlación [25]	Análisis de Regresión lineal; Correlación Spearman; Análisis de Regresión no lineal; Selección basada en correlación, con herramienta WEKA; Regresión logística.
Sugerencias de nombres mediante análisis de texto [25]	Lenguaje neuronal log-bilineal; Modelo de subtoken
Medir métricas mediante análisis de texto [25]	AST: árbol de sintaxis abstracta; Análisis de lenguaje natural (léxico).
Impacto en la carga cognitiva [16]	Espectroscopía de infrarrojo cercano funcional (fNIRS)

El aprendizaje de máquina supervisado ha permitido obtener métodos para evaluar automáticamente la legibilidad del código fuente, que, a pesar de no ser métodos definitivos aún, han logrado una precisión por encima del 80%. Los investigadores interesados en continuar con los trabajos podrían usar las mismas técnicas utilizadas en la literatura, tales como *random forest*, redes neuronales o el aprendizaje profundo, aplicándolas de forma individual o combinadas. Se recomienda la exploración de modelos basados en redes neuronales profundas, teniendo en cuenta que el aprendizaje profundo permite el entrenamiento y clasificación sin que sea necesario seleccionar las características del código fuente de antemano. Los resultados recientes más prometedores apuntan en esta dirección. Por otro lado, si lo que se desea es continuar con la búsqueda de las características del código fuente que permiten medir su legibilidad, se pueden

utilizar las técnicas de aprendizaje de máquina no supervisado, como el *clustering* (agrupación), o alguna de las técnicas de regresión para el análisis de la correlación entre la característica y la predicción de legibilidad.

Por otra parte, como resultado adicional de la RSL, a continuación, se listan los repositorios de software identificados que contienen fragmentos de código fuente útiles para nuevos trabajos:

1) https://github.com/zibranm/dataset [7]
2) http://groups.inf.ed.ac.uk/cup/naturalize/#evaldata [8]
3) https://github.com/roncoleman125/Pretty [13]
4) https://github.com/CityUQingMi/DeepCRM/tree/master/Code%20Snippet%20Repository [20]
5) https://dibt.unimol.it/report/readability/ [25].

Asimismo, se presentan repositorios de software que contienen herramientas computacionales utilizadas en los estudios:

1) Medir métricas de legibilidad ARI, SMOG, FKI, GFI, CLI, BRS (https://github.com/ipeirotis/ReadabilityMetrics)
2) Contar o extraer elementos del código fuente (http://se.cite.ehime-u.ac.jp/tool/)
3) Analizador de código estático (https://pmd.github.io/)
4) Evalúa el estilo y diseño del código fuente, de acuerdo con las convenciones (https://checkstyle.sourceforge.io/)
5) Algoritmos usados en el estudio de la carga cognitiva (https://github.com/Smfakhoury/fNIRS-and-Cognitive-Load)
6) Plugin para el IDE IntelliJ que permite evaluar la calidad en tiempo real (https://reticulaplugin.github.io/);
7) Base de datos léxica del idioma inglés (https://wordnet.princeton.edu/)
8) Código Python para contar las violaciones de reglas y otro para transformar el código (https://github.com/CityU-QingMi/DeepCRM)
9) Permite extraer métricas de estilo y generar un archivo CSV (https://github.com/robertyuyang/StylisticFingerprinting)

10) Modelo de legibilidad de Scalabrino et al. [25] (https://dibt.unimol.it/report/readability/)
11) Algoritmo para la segmentación de palabras en inglés (https://pypi.org/project/wordsegment/)
12) Embellecedor de código fuente online (http://prettyprinter.de/)
13) Embellecedor de código fuente (http://astyle.sourceforge.net/)
14) Calcular la legibilidad del código (https://github.com/Bowie-State-University/).

C. Retos y aplicaciones de la legibilidad del código fuente

Los autores de los estudios seleccionados en la RSL presentaron en sus investigaciones algunas aplicaciones de interés para la legibilidad del código fuente, las cuales se encuentran sintetizadas a continuación:

1) Modelos de legibilidad del código fuente [1, 3, 20, 21, 25, 30] .

2) Apoyar el aprendizaje de las buenas prácticas de legibilidad y los estilos de embellecimiento en los cursos de programación de las instituciones de educación superior [11, 15, 31].

3) Evaluar el impacto de las características de legibilidad del código fuente en la carga cognitiva del lector [16].

4) Gamificación para el aprendizaje de la legibilidad del código fuente en estudiantes universitarios [19].

5) Identificar el perfil de las personas en pruebas de reclutamiento de empresas de software para el cargo de programador, analizando el código fuente de sus programas, y evaluando habilidad en lenguaje y la legibilidad [23].

6) Asignación automática de nombres para los identificadores de clase y métodos [29].

7) Predecir los métodos propensos a fallas de acuerdo a la cantidad de comentarios que tienen.

Las más relevantes son el desarrollo de modelos computacionales para medir la legibilidad del código fuente de forma automática y el apoyo a las instituciones de

educación superior con la enseñanza de buenas prácticas de legibilidad de software [17, 19, 27].

Por otra parte, algunos autores mencionaron deficiencias en sus propuestas, así como posibles mejoras que pueden realizar ellos u otros investigadores que deseen continuar con los estudios, los cuales serían los retos. A continuación, se presenta una síntesis de los retos.

1) Desarrollar una herramienta, algoritmo, o un plugin de apoyo para que un programador pueda medir la legibilidad de su código fuente en tiempo de desarrollo.
2) Continuar las investigaciones y proponer mejores métricas para evaluar la legibilidad del código fuente; por ejemplo, tener una métrica basada en la calidad de los comentarios, analizando el contenido del mismo usando técnicas de procesamiento de lenguaje natural, no por la cantidad de líneas [5].
3) Diversificar los lenguajes de programación de los conjuntos de datos utilizados en los estudios, teniendo en cuenta que casi todos los fragmentos corresponden al lenguaje Java.
4) Probar el nuevo conjunto de datos en los modelos ya propuestos, así como en la replicación de otros estudios.
5) Utilizar el algoritmo de asignación automática de nombres a otros tipos de identificadores en el código fuente, aparte de la clase y el método, mejorando así la legibilidad.
6) Permitir la parametrización en los modelos de legibilidad para que, de acuerdo con tipo de lenguaje de programación, se ajusten las características y las métricas a evaluar.
7) Para los experimentos que usaron WordNet, reemplazarla por otras técnicas como DISCO y NER.
8) Diversificar el conjunto de participantes que ayudaron con la anotación de los datos en los estudios relacionados con los modelos de legibilidad, en cuanto a la experiencia y el origen.

9) Mejorar el rendimiento del modelo de legibilidad IncepCRM, ajustando parámetros.

IV. CONCLUSIONES

La revisión sistemática de literatura realizada permitió obtener y seleccionar 30 documentos finales con información relevante de las características del código fuente relacionadas con su legibilidad, las métricas que permiten medirla, y los métodos para su evaluación automática. Adicionalmente, se identificaron los retos y aplicaciones propuestos por investigadores en el área. Las características del software identificadas como más relevantes para medir la legibilidad del código fuente son los espacios en blanco (identación), comentarios, LOC, número de paréntesis, número de operadores aritméticos, entropía, volumen del programa, número y nombre de los identificadores, estructuras de control, y número de palabras clave [1, 7, 10, 12]. Las métricas más relevantes fueron el conteo de espacios en blanco, la métrica de Posnett (PHD) [3], WSCR para el código fuente en lenguaje WSDL (servicios web) y la legibilidad de los comentarios (CR) [25] .

La mayoría de los métodos utilizados para determinar la legibilidad de software utilizaron técnicas de aprendizaje de máquina, entrenando clasificadores de forma supervisada [1, 19, 25]. Otros métodos usaron técnicas de análisis léxico, tomando al código fuente como un texto [18, 28]. No obstante, existen evidencias recientes que señalan el camino a seguir usando métodos automáticos basados en redes neuronales profundas [20], donde no es necesario proponer características. Las aplicaciones más relevantes encontradas fueron los modelos automáticos de legibilidad [20, 25] y el apoyo del aprendizaje de buenas prácticas [17, 27]. Los retos más relevantes fueron: 1) desarrollar un algoritmo que pueda ser incorporado a algún IDE para evaluar la legibilidad del código fuente en tiempo de desarrollo; 2) diversificar los lenguajes de programación usados, aparte de Java, y 3) continuar con las investigaciones, ya que aún no se cuenta con el método, las características y las métricas definitivas. Puede consultar información adicional en el repositorio: https://github.com/sofi876/MetricasLegibilidadSW/tree/master/Tablas

CONTRIBUCIÓN DE LOS AUTORES

Ingrid-Sofía Echavarría-Flórez: Análisis formal, Investigación, Curación de datos, Software, Redacción – borrador original.

Felipe Restrepo-Calle: Conceptualización, Metodología, Supervisión, Validación, Redacción – revisión y edición.

REFERENCIAS

[1] R. P. L. Buse, and W. R. Weimer, "Learning a Metric for Code Readability," *IEEE Transactions on Software Engineering*, vol. 36 (4), pp. 546-558, Jul. 2010. https://doi.org/10.1109/TSE.2009.70

[2] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Automatically assessing code understandability: How far are we?," in *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 417-427.

[3] D. Posnett, A. Hindle, and P. Devanbu, "A simpler model of software readability," in *Proceeding of the 8th working conference on Mining software repositories*, 2011, pp. 73-82. https://doi.org/10.1145/1985441.1985454

[4] M. Akour, and B. Falah, "Application domain and programming language readability yardsticks," in *7th International Conference on Computer Science and Information Technology (CSIT)*, 2016, pp. 1-6. https://doi.org/10.1109/CSIT.2016.7549476

[5] U. A. Mannan, I. Ahmed, and A. Sarma, "Towards understanding code readability and its impact on design quality," in *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*, 2018, pp. 18-21. https://doi.org/10.1145/3283812.3283820

[6] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Information and Software Technology*, vol. 51 (1), pp. 7-15, 2009. https://doi.org/10.1016/j.infsof.2008.09.009

[7] D. Alawad, M. Panta, M. Zibran, and R. Islam, "An Empirical Study of the Relationships between Code Readability and Software Complexity," in *27th Int Conf on Software Engineering and Data Engineering*, 2018, pp. 1-7.

[8] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Suggesting accurate method and class names," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 38-49. https://doi.org/10.1145/2786805.2786849

[9] H. Aman, S. Amasaki, T. Sasaki, and M. Kawahara, "Lines of Comments as a Noteworthy Metric for Analyzing Fault-Proneness in Methods," *IEICE Transactions on Information and Systems*, vol. E98.D (12), pp. 2218-2228, 2015. https://doi.org/10.1587/transinf.2015EDP7107

[10] J. Borstler, and B. Paech, "The Role of Method Chains and Comments in Software Readability and Comprehension-An Experiment," *IEEE Transactions on Software Engineering*, vol. 42 (9), pp. 886-898, Sep. 2016. https://doi.org/10.1109/TSE.2016.2527791

[11] H. M. Chen, W. H. Chen, and C. C. Lee, "An automated assessment system for analysis of coding convention violations in Java programming assignments," *Journal of Information Science and Engineering*, vol. 34 (5), pp. 1203-1221, 2018. https://doi.org/10.6688/JISE.201809_34(5).0006

[12] S. Choi, S. Kim, J.-H. Lee, J. Kim, and J.-Y. Choi, "Measuring the Extent of Source Code Readability Using Regression Analysis," in *International Conference on Computational Science and Its Applications*, 2018, pp. 410-421. https://doi.org/10.1007/978-3-319-95171-3_32

[13] R. Coleman, "Aesthetics Versus Readability of Source Code," *International Journal of Advanced Computer Science and Applications*, vol. 9 (9), pp. 12-18, 2018. https://doi.org/10.14569/IJACSA.2018.090902

[14] A. De Renzis, M. Garriga, A. Flores, A. Cechich, C. Mateos, and A. Zunino, "A domain independent readability metric for web service descriptions," *Computer Standards & Interfaces*, vol. 50, pp. 124-141, Feb. 2017. https://doi.org/10.1016/j.csi.2016.09.005

[15] R. M. dos Santos, and M. A. Gerosa, "Impacts of coding practices on readability," in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 277-285. https://doi.org/10.1145/3196321.3196342

[16] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *IEEE/ACM 26th International Conference on Program Comprehension*, 2018, pp. 286-296.

[17] L. Frunzio, B. Lin, M. Lanza, and G. Bavota, "RETICULA: Real-time code quality assessment," in *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*, 2018, pp. 542-546. https://doi.org/10.1109/SANER.2018.8330256

[18] Y. Liu, X. Sun, and Y. Duan, "Analyzing program readability based on WordNet," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, 2015, pp. 1-2. https://doi.org/10.1145/2745802.2745837

[19] Q. Mi, J. Keung, X. Mei, Y. Xiao, and W. Chan, "A Gamification Technique for Motivating Students to Learn Code Readability in Software Engineering," in *International Symposium on Educational Technology*, 2018, pp. 250-254. https://doi.org/10.1109/ISET.2018.00062

[20] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and Y. Gao, "Improving code readability classification using convolutional neural networks," *Information and Software Technology*, vol. 104, pp. 60-71, Dec. 2018. https://doi.org/10.1016/j.infsof.2018.07.006

[21] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and X. Mei, "An Inception Architecture-Based Model for Improving Code Readability Classification," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering*, 2018, pp. 139-144. https://doi.org/10.1145/3210459.3210473

[22] Q. Mi, J. Keung, and Y. Yu, "Measuring the Stylistic Inconsistency in Software Projects using Hierarchical Agglomerative Clustering," in *The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2016, pp. 1-10. https://doi.org/10.1145/2972958.2972963

[23] D. J. Ferreira Novais, M. J. V. Pereira, and P. R. Henriques, "Program analysis for Clustering Programmers' Profile," in *Federated Conf on Computer Science and Information Systems*, Prague, 2017, pp. 701-705. https://doi.org/10.15439/2017F147

[24] A. Pahal, and R. S. Chillar, "Code Readability: A Review of Metrics for Software Quality," *International Journal of Computer Trends and Technology*, vol. 46 (1), pp. 1-4, 2017. https://doi.org/10.14445/22312803/IJCTT-V46P101

[25] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk, "A comprehensive model for code readability," *Journal of Software: Evolution and Process*, vol. 30 (6), e1958, Jun. 2018. https://doi.org/10.1002/smr.1958

[26] T. Sedano, "Code Readability Testing, an Empirical Study," in *IEEE 29th International Conference on*

Software Engineering Education and Training, 2016, pp. 111-117. https://doi.org/10.1109/CSEET.2016.36

[27] A. Wulff-Jensen, K. Ruder, E. Triantafyllou, and L. E. Bruni, "Gaze Strategies Can Reveal the Impact of Source Code Features on the Cognitive Load of Novice Programmers," in *International Conference on Applied Human Factors and Ergonomics*, 2019. https://doi.org/10.1007/978-3-319-94866-9_9

[28] W. Xu, D. Xu, and L. Deng, "Measurement of Source Code Readability Using Word Concreteness and Memory Retention of Variable Names," in *IEEE 41st Annual Computer Software and Applications Conf*, 2017, pp. 33-38. https://doi.org/10.1109/COMPSAC.2017.166

[29] S. Butler, M. Wermelinger, Yijun Yu, and H. Sharp, "Exploring the Influence of Identifier Names on Code Quality: An Empirical Study," in *Proceedings of the 14th European Conference on Software Maintenance and Reengineering*, 2010, pp. 156-165. https://doi.org/10.1109/CSMR.2010.27

[30] J. Dorn, *A General Software Readability Model*, 2012. https://www.cs.virginia.edu/~weimer/students/dorn-mcs-paper.pdf

[31] I. B. Sampaio, and L. Barbosa, "Software readability practices and the importance of their teaching," in *7th International Conference on Information and Communication Systems*, 2016, pp. 304-309. https://doi.org/10.1109/IACS.2016.7476069