COMPUTATION E INFORMATICA

Recibido 13 Dic. 2022 *ReCIBE, Año 12 No. 1, May. 2023*

Aceptado 09 Ene. 2023

Hacia una propuesta de rúbrica para la evaluación de la calidad de comentarios de código en Java

Towards a rubric proposal for the evaluation of the quality of code comments in Java

Juan Carlos García Murillo[1]
garciamjuancarlos14@gmail.com

Universidad Autónoma de Zacatecas[1]

RESUMEN

En un desarrollo de software la documentación juega un papel importante para conocer el origen, detalles, razón de ser, entre otros factores de un proyecto. Su utilidad es muy amplia y se extiende incluso como apoyo para que futuros desarrolladores puedan darle continuidad al sistema con menores dificultades. El código de un software es de los elementos más complejos de entender, ya que cada desarrollador posee una forma propia de programar. La manera por excelencia para hacer un código entendible es acompañarlo de buenos comentarios: completos y consistentes. En este artículo proponemos una rúbrica para evaluar la calidad de comentarios de código en Java. Dicha rúbrica está dividida por categorías según el tipo de comentario a evaluar: comentario de autoría, comentario de clase, comentario de método y comentario de fragmento de código, cada categoría establece un conjunto de reglas que deben cumplirse para declarar que un comentario es de calidad. El aumento en la legibilidad y mantenibilidad del código en proyectos de software que se encuentran actualmente en uso después de haber usado la rúbrica nos ayudará a validar su efectividad. Una vez comprobada esta eficacia ayudará a los equipos a poseer códigos correctamente comentados y a reducir costes en labores de entendimiento y mantenimiento del código pues este será más legible y descriptivo.

Palabras clave: Comentario de código, evaluación, Java, software, mantenimiento.

ABSTRACT

In a software development, the documentation plays an important role for knowing the origin, details and reason to be, among other factors of a project. Its utility is wide and is extended even as support for future developers can continue the development with less difficulty. The code of a software is one of the most complex elements to understand, since each developer has their own style of programming. The best way for making the code understandable is accompany it with good comments all of them complete and consistent. In this article, we propose a rubric to evaluate the quality of code comments in Java. This rubric is divided into categories according to the type of comment to be assessed: copyright comment, class comment, method comment, and code snippet comment. Each category defines a set of rules that must be met to declare a comment has quality. Through the increment of readability and maintainability of code in software projects that are currently in use after using the rubric will help us to validate its effectiveness. Once this effectiveness has been verified, it will help the teams to have correctly commented codes and to reduce costs in the work of understanding and maintaining the code, since it will be more readable and descriptive.

Keywords: Code comment, assessment, Java, software, maintenance.

1. INTRODUCCIÓN

Es común codificar software con miradas al corto plazo, o con un enfoque "egoísta", es decir, no se piensa muchas veces en que los integrantes de una empresa, institución o de un proyecto por más grande o pequeño que sea está sujeto a cambios en su equipo de trabajo.

Programar es un arte, no hay una forma cuadrada que obligue a todas las personas a realizar código siguiendo la misma lógica o forma. Esto complica la labor de entendimiento de un código ajeno, sobre todo al momento de darle mantenimiento a una pieza de instrucciones lógicas que no es de nuestra autoría. Incluso aun habiendo sido nosotros los desarrolladores de un código, después de un tiempo se vuelve complejo de entender.

En los lenguajes de programación existen los comentarios de código: un comentario de código es la interpretación y explicación del código. Estas descripciones utilizan el lenguaje natural para explicar la lógica y la función implementada detrás del código (Steidl et al., 2013) . En este contexto, los comentarios de código representan la principal fuente de documentación de un sistema de software y son, por lo tanto, fundamentales para el entendimiento del código tanto en la fase de desarrollo como en la fase de mantenimiento (Woodfield et al., 1981).

A pesar de que es posible comentar el código, muchos desarrolladores no realizan esta importante tarea, condenando a los futuros programadores a gastar mucho tiempo tratando de entender cómo funciona el software en cuestión; esto en parte es producto de que la ausencia de documentación en el código no impide la ejecución de los programas. Ahora bien, existen personas que sí comentan su código, pero a un nivel poco detallado o 'incompleto'.

En los proyectos de software el tiempo es una variable que debe manejarse con mucho cuidado ya que los retrasos pueden impactar negativamente al éxito, presupuesto y confianza del equipo de desarrollo. Los comentarios de mala calidad o la ausencia de ellos es un factor que puede orientar al proyecto a sufrir de este problema, por lo que es importante asegurar que en todo momento se cuente con documentación en el código y que la misma sea de calidad.

En este artículo definiremos una rúbrica para la evaluación de la calidad de comentarios de código en el lenguaje de programación Java, a fin de proveer a la comunidad una serie de reglas, que, de ser validadas, les permitirán reducir retrasos por entendimiento del código.

2. ESTADO DEL ARTE

Hirohisa Aman y sus colaboradores proponen evaluar cuantitativamente el valor de los comentarios de código aprovechando un modelo de procesamiento de lenguaje natural (NLP, por sus siglas en inglés) bien diseñado: Doc2Vec. En el artículo se considera un método o función como un documento y el contenido del mismo se expresa como un vector usando el modelo Doc2Vec. Después de esto, crean un nuevo vector con el mismo contenido del método, pero sin los comentarios de código. Luego, comparan los vectores obtenidos antes y después de la eliminación de los comentarios. A medida que los comentarios borrados brindan información más relevante, la diferencia entre los vectores se hace más grande. Sus resultados muestran que un método que tiene comentarios poco informativos es probable que sea propenso a sufrir modificaciones. (Aman et al., 2016).

Bai Yang y sus colaboradores recopilan investigaciones relevantes sobre los comentarios de código, incluyendo principalmente cuatro aspectos: generación automática de comentarios de código, consistencia de comentarios de código, clasificación de comentarios de código y evaluación de calidad de comentarios de código, que es el tema de nuestro interés. Para cada aspecto exponen técnicas que han sido utilizadas para abordar cada problema. En el ámbito de la evaluación de comentarios de código proporcionan una tabla comparativa que ilustra las investigaciones recopiladas sobre este tema exponiendo sus ventajas, desventajas, el método que utiliza cada una para realizar la evaluación, entre otras. Los resultados de esta investigación sirven como una guía para los inspectores de código para elegir las técnicas o herramientas adecuadas en materia de análisis y evaluación de comentarios de código (Bai et al., 2019).

En el estudio de Martina Iammarino y sus compañeros proponen evaluar la calidad de los comentarios de código en términos de coherencia con el código fuente, centrándose exclusivamente en su contenido. Presentan un enfoque basado en la técnica de *Topic Modeling*, y *Natural Language Processing*. El modelado de temas se refiere a modelos estadísticos utilizados para extraer temas que aparecen en una colección de documentos. Un tema es un conjunto de palabras similares, cada una asociada a un peso que expresa la probabilidad de que esté relacionada con el tema. El método utilizado para el modelado de temas en este trabajo tiene por nombre *Asignación Latente de Dirichlet.* Dicho método es una técnica no supervisada de minería de texto que es la encargada de asignar a cada palabra del documento un tema.

El proceso de evaluación propuesto consta de extraer los temas en el código fuente y en los comentarios de dicho código por separado y calcular la *Divergencia Kullback Leibler* que hay entre ellos a fin de conocer la coherencia que hay entre ambos. Los resultados del artículo muestran una similitud en la tendencia de distribución de temas, por lo tanto, casi todas las clases están asociadas a no más de tres temas (Iammarino et al., 2019).

Deze Wang y sus compañeros proponen tratar la evaluación de comentarios de código como un problema de clasificación donde el clasificador decide si la calidad del comentario es confiable o no. Se propone el uso de la red neuronal de entrada múltiple *DComment* para convertir en vectores al código y comentarios respectivamente basado en sus características. Dichos vectores son tratados por separado para finalmente introducirlos al clasificador que nos dirá si el comentario es confiable o no. Los resultados experimentales de la investigación muestran que el enfoque utilizado, en general, supera otras técnicas, tanto en su conjunto de datos etiquetado como en el conjunto de datos público, con una puntuación de 96.91% y 91.90% respectivamente. Usando los datos de entrenamiento y los de prueba de distintas fuentes, su enfoque aún puede lograr un rendimiento razonable, lo que demuestra su capacidad de generalización (Wang et al., 2019).

Pooja Rani en este estudio aborda el tema de la evaluación de comentarios desde tres perspectivas: qué preguntan los desarrolladores sobre prácticas de comentado de código, lo que estos mismos escriben en sus comentarios y cómo los investigadores los apoyan para evaluar la calidad de los comentarios. Para cada aspecto se realizó un análisis distinto a fin de generar hallazgos importantes.

En el primer aspecto el autor realizó un análisis de importantes foros de programadores tales como *StackOverFlow* y *Quora;* como resultado de este proceso se presenta una taxonomía empíricamente validada de preguntas relacionadas con la convención de comentarios de varios foros comunitarios.

El segundo aspecto consta de un análisis de los comentarios de código que poseen proyectos de software realizados en *Java, Python* y *Pharo* donde se obtiene como resultado una taxonomía validada empíricamente que caracteriza los tipos de información que se encuentra en comentarios de clase.

En el tercer y último aspecto, el autor expone la planeación, objetivos, criterios de inclusión y exclusión, etc., de una futura revisión sistemática de literatura, donde se espera obtener como resultado hallazgos importantes para formular una taxonomía de calidad de comentarios que pueda ayudar a los investigadores y desarrolladores a identificar los atributos de calidad adecuados para cada tipo de comentario.

Los hallazgos preliminares de la investigación muestran que los desarrolladores incorporan varios tipos de información en los comentarios de clase en los lenguajes de programación. Aún así, enfrentan problemas para ubicar pautas relevantes en materia de escribir comentarios coherentes e informativos, verificar el cumplimiento de sus comentarios con las pautas y evaluar el estado general de la calidad de los comentarios (Rani, 2021).

3. RÚBRICA PARA LA EVALUACIÓN DE LA CALIDAD DE COMENTARIOS

Como pudimos ver, tres de los cinco artículos presentados en el estado del arte utilizan alguna técnica de análisis de datos o *machine learning* para realizar la evaluación de comentarios de código, dichos artículos son: *A Doc2Vec-Based Assessment of Comments and Its Application to Change-Prone Method Analysis* (Aman et al., 2016)*, A Topic Modeling Approach To Evaluate The Comments Consistency To Source Code* (Iammarino et al., 2019) y *Deep Code-Comment Understanding and Assessment* (Wang et al., 2019). Esto convierte al uso de estas técnicas en el común denominador para abordar la evaluación de comentarios. Ante la necesidad de contar con equipos de cómputo con una alta capacidad de procesamiento para usar algoritmos de análisis de texto o de lenguaje natural, la rúbrica que será propuesta en este artículo, a diferencia de las investigaciones descritas con anterioridad, se compone de una serie de reglas, donde su cumplimiento será definido por un integrante del equipo de desarrollo (aunque esto a futuro se podría automatizar), eliminando así el requisito de contar con una alta y costosa capacidad de procesamiento cuando no se posea. Todo esto con el objetivo de ayudar a mejorar la calidad de los comentarios de código.

Java es uno de los lenguajes de programación más utilizados en la industria, ya sea para proyectos que llevan años existiendo o nuevos desarrollos. Dada la importante presencia de este lenguaje de programación en la tecnología actual, crece entonces la demanda de que el código hecho en Java posea comentarios de calidad que promuevan su legibilidad y mantenibilidad.

3.1 Categorías de comentarios en Java

En la Universidad de la prefectura de Okayama en Japón establecen cuatro categorías para dividir los comentarios que existen en Java, las cuales son: designación de derechos de autor, descripción simple de la clase, manual del programador para el método y explicación del fragmento de código (Aman et al., 2016), la figura 1 muestra un ejemplo de cada categoría. A continuación, se describe de una manera más amplia cada categoría:

- **Comentario de autoría:** Posee información relativa al o los creadores del código y sobre derechos de autor. Este tipo de comentarios en el código suelen estar al principio del archivo.
- **Comentario de clase:** Describe de manera general las funciones que desempeñan los métodos de la clase.
- **Comentario de método:** Describe la funcionalidad que desempeña el método, por ejemplo, el rol que funge cada parámetro, el procedimiento que se realiza dentro, los resultados que produce, así como el significado de cada uno de ellos. En palabras del autor “un manual para el programador sobre el método”.
- **Comentario de fragmento de código:** Describe el funcionamiento de una línea de código o un conjunto de líneas de código dentro de un método. Se suele hacer uso de esta categoría de comentarios cuando dentro de una función hay partes que son aún más complejas de entender a simple vista.

Figura 1. Categorías de comentarios

3.2 Reglas de la rúbrica

3.2.1 Reglas para la categoría de autoría

En esta categoría deberá situarse todo lo relativo a los derechos de autor. Según el diccionario de la real academia española podemos definir este concepto de la siguiente manera: “derecho que la ley reconoce al autor de una obra intelectual o artística para autorizar su reproducción y participar de los beneficios que esta genere”. (Real Academia Española, n.d.). Dicho así el código deberá estar protegido de acuerdos a las normas o licencia que prefieran el o los autores.

La declaración de derechos de autor debe situarse al inicio del archivo de código y contener los siguientes elementos: Palabra declarativa "*Copyright"*, año de creación, nombre del o los autores (persona física o empresa), oración "todos los derechos reservados", licencia sobre la que se rige el código, enlace para consultar la licencia utilizada o descripción de la licencia dentro del mismo bloque de comentario.

3.2.2 Reglas para la categoría de clase

Esta categoría de comentarios debe componerse de los siguientes elementos: descripción general de las funcionalidades que desempeña la clase o lo que representa, nombre del o los autores y la versión. Para la versión y el autor es necesario utilizar las etiquetas *@author* y *@version* respectivamente (*How to Write Doc Comments for the Javadoc Tool*, n.d.). Es necesario mencionar que la descripción debe ser congruente con el código que contiene.

En Java existen variables a nivel de clase, en caso de existir, todas deberán estar comentadas con una descripción que permita conocer el rol que desempeña cada una de ellas.

Dentro de las clases podemos diferenciar tres tipos donde para cada uno la descripción de la clase debe seguir una cierta línea de especificación:

- **Clase matemática:** cuando la clase contiene operaciones matemáticas, transformaciones etc., su descripción requiere estar orientada a resumir estas funcionalidades, por ejemplo, una clase que realice operaciones estadísticas.
- **Clase de tabla:** cuando la clase representa un tipo de dato o tabla de una base de datos la descripción debe estar encaminada a comentar la información que almacena al momento de estar instanciada, por ejemplo, una clase que almacene la información de un "empleado".
- **Clase combinada:** cuando la clase almacena datos y éstos mismos pueden ser manipulados dentro de la clase, la descripción debe definir qué datos almacena la clase y qué operaciones pueden aplicarse, por ejemplo, una clase "Arreglo".

Las interfaces son un caso especial en Java, ya que no son una clase como tal, pero tienen un gran parecido, por lo tanto, las reglas a seguir para documentarlas son las mismas que fueron descritas con anterioridad.

3.2.3 Reglas para la categoría de método

La composición de un método en Java varía de acuerdo a lo que el programador necesita para codificar su funcionalidad, por lo tanto, la cantidad de reglas no es la misma para todos los casos. Toda función en Java comparte dos reglas: describir de manera detallada lo que realiza la función y que la descripción sea congruente con el código que contiene el método. Sin embargo, hay elementos adicionales que pueden estar presentes en la función y que su existencia requiere ampliar el comentario del método en cuestión. En la tabla 1 se describen las reglas a seguir cuando existe cada elemento.

Número de regla	Nombre del elemento	Descripción de la o las Regla(s)
1	Valor de retorno	Explicar lo que significa cada valor que regresa la función y bajo qué condiciones ocurre esto. El formato a usar es el siguiente: *@return [explicación]*.
2	Parámetro(s)	Explicar el rol que desempeña cada parámetro dentro de la función. El formato a usar es el siguiente: *@param [nombre del parámetro] [explicación]*.
3	Excepción(nes)	Explicar bajo qué condiciones ocurre cada excepción que lanza el método. El formato a usar es el siguiente: *@throws [nombre de la excepción] [explicación]*.
4	Anotación *@Deprecated*	Indicar desde qué versión el método ha sido declarado como obsoleto (*deprecado)* y el o los métodos que lo
5	(*How and When to Deprecate APIs*, n.d.)	reemplazaron. El formato a usar es el siguiente: *@deprecated [versión en que se deprecó]. Reemplazado por [{@link #nombre_del_metodo}]*

Tabla 1. Reglas para los elementos variables en un método.

3.2.4 Reglas para la categoría de fragmento de código

La última categoría está compuesta por el código que se escribe dentro de los métodos. El objetivo aquí es conseguir un efecto en donde “la función nos platique a profundidad lo que realiza”, para conseguir esto es necesario establecer como regla que todo comentario, a excepción de las variables declaradas dentro de los métodos, deberán ser escritos en primera persona. Dicho esto, los fragmentos de código que se requieren comentar son los siguientes:

- Variables: describir el rol que desempeña.
- Condiciones: describir lo que evalúa.
- Ciclos: describir bajo qué condiciones el ciclo termina.
- Operaciones matemáticas con llamadas a funciones: describir lo que se obtiene a través de la llamada a otras funciones y el resultado que se consigue.

3.3 Uso de la rúbrica

Esta rúbrica será aplicable para asegurar la calidad de los comentarios de código cuando se utilice Java como lenguaje de programación. A través del cumplimiento de las reglas descritas se fomentará tener un código con comentarios altamente descriptivos lo que impactará positivamente en la legibilidad y por lo tanto en la mantenibilidad del código.

Deberán cumplirse todas las reglas definidas previamente, para esto se usará la tabla 2 que funge como el *checklist* de apoyo para la persona encargada de realizar la revisión. Los comentarios deberán analizarse por archivo de código, por lo tanto, no podrá dejarse incompleta la inspección de algún archivo y pasarse a otro. La rúbrica será utilizada por un integrante del equipo de desarrollo siempre y cuando no sea el autor del código a revisar y deberá llenar el *checklist* como se explica a continuación.

El *checklist* consta de un encabezado donde se deberá especificar el nombre del revisor, el nombre del archivo de código revisado y la fecha en que se realizó la revisión. posteriormente hay tres columnas:

- **Regla:** describe la regla que se tiene que cumplir,
- **¿Cumple?:** permite al revisor marcar con una “X” si la regla se cumple o dejar el espacio vacío en caso contrario. Existe la posibilidad de marcar la casilla con un N/A cuando la regla no aplique, por ejemplo, que la clase no contenga variables, que el método no posea parámetros, que no existan condiciones dentro del método, etc.
- **Observaciones:** tiene a fin permitir al inspector comentar lo que le falta al comentario para cumplir con la regla cuando sea el caso.

Se recomienda a los desarrolladores conocer la rúbrica previa a la escritura de sus comentarios para agilizar las revisiones de comentarios y reducir el incumplimiento de las reglas establecidas.

Una vez que se complete la revisión de los comentarios y la rúbrica esté debidamente llenada, se le hará llegar al equipo de desarrollo para que realice las correcciones pertinentes. Por ningún motivo deberán hacerse modificaciones al código si aún no se atienden las correcciones señaladas en la rúbrica. El revisor será el encargado de supervisar que los cambios solicitados sean aplicados.

Nombre del revisor: **Nombre del archivo de código:**		**Fecha:**
REGLA	**¿CUMPLE?**	**Observaciones**
COMENTARIOS DE AUTORÍA		
Se encuentra al inicio del archivo de código		
Contiene el formato: *Copyright [año de creación]. [Autor(es)] todos los derechos reservados*		
Nombra la licencia bajo la que se rige el código		
Contiene un enlace para consultar la licencia utilizada o describe la misma en el bloque de comentario		
COMENTARIOS DE CLASE O INTERFAZ		
Nombre de la clase o interfaz:		
Describe de manera general la clase		
La descripción es congruente con el código que contiene		
Cada variable de clase tiene una descripción del rol que desempeña		
Nombra al o los autores de la clase con el formato: *@author [nombre del autor]*		
Indica la versión de la clase con el formato: *@version [numero de versión]*		
Indica la última fecha de actualización del comentario		

COMENTARIO DE MÉTODO		
Firma del método:		
Describe detalladamente lo que realiza la función		
La descripción es congruente con el código que contiene el método		
Explica cada valor que regresa la función y bajo qué condiciones con el formato: *@return [explicación]*		
Explicar el rol que desempeña cada parámetro dentro de la función con el formato: *@param [nombre del parámetro] [explicación]*		
Indicar desde qué versión el método ha sido declarado como obsoleto (*deprecado)* y el o los métodos que lo reemplazaron con el formato:		
@deprecated [versión en que se deprecó]. Reemplazado por [{@link #nombre_del_metodo}]		
COMENTARIO DE FRAGMENTO DE CÓDIGO		
Las variables describen el rol que desempeñan en el método		
Las condiciones describen lo que evalúan		
La descripción de las condiciones está redactada en primera persona		
Los ciclos describen bajo qué condiciones terminan		
La descripción de los ciclos está redactada en primera persona		
Los comentarios de las operaciones aritméticas con llamadas a otras funciones describen lo que se obtiene a través de la llamada a los métodos externos y el resultado que se consigue de toda la operación		

Tabla 2. *Checklist* de la rúbrica para evaluar comentarios de código

4. DISCUSIÓN

Con el uso de esta rúbrica se espera que la industria de la ingeniería de software tenga una opción más a elegir para asegurar la calidad sus comentarios de código cuando se utilice Java como lenguaje de programación. Se pretende que los comentarios escritos bajo el conocimiento y uso de la rúbrica permitan a los programadores entender de una manera más rápida y sencilla el código, aumentar su legibilidad y así reducir gastos en mantenimiento. También se busca que los comentarios sean completamente congruentes con el código evitando así que al darle mantenimiento se olvide actualizar la documentación.

5. CONCLUSIONES

La investigación sobre la calidad de los comentarios en el código es un área que tiene mucho potencial de mejora. En la parte de la implementación dentro del ciclo de vida del desarrollo de software muchos de los esfuerzos se concentran en la elaboración de nuevos algoritmos, lenguajes de programación o tecnologías que permitan agilizar la tarea de codificar. Sin embargo, estos archivos con grandes cantidades de líneas de código además de cumplir exclusivamente con la funcionalidad para la que fueron creados deben aportar también valor para los desarrolladores cuya tarea sea darles mantenimiento. De aquí aparece la necesidad de contar con opciones que permitan realizar una evaluación a los comentarios escritos por los programadores a fin de asegurar su calidad. En esta investigación favorecemos a los proyectos que no cuentan con equipos de cómputo con altas capacidades de procesamiento ya que la rúbrica no requiere de estas facultades; de esta manera se habilita la opción de contar con un método de aseguramiento de calidad de comentarios de código sin costo computacional, que a futuro puede automatizarse generando así un costo que será mínimo comparado con otras técnicas dada su naturaleza.

Comentar el código es una tarea que erróneamente se deja a la posteridad o en el olvido generalmente por falta de tiempo. Miremos ahora esta labor como una inversión que será retornada al corto o largo plazo según la evolución del sistema en cuestión. Es necesario programar siempre bajo la premisa de que un buen código va acompañado de una buena documentación y si dicha documentación sigue un conjunto de reglas como las que fueron propuestas en este artículo, los futuros desarrolladores encontrarán comentarios con un alto grado de legibilidad, mantenibilidad, estandarizados y un mayor nivel de calidad.

6. REFERENCIAS

Aman, H., Amasaki, S., Yokogawa, T., & Kawahara, M. (2016). *A Doc2Vec-Based Assessment of Comments and Its Application to Change-Prone Method Analysis*. https://bit.ly/2Pb89Au

Bai, Y., Zhang, L., & Zhao, F. (2019). A survey on research of code comment. *ACM International Conference Proceeding Series*, 45–51. https://doi.org/10.1145/3312662.3312710

How and When to Deprecate APIs. (n.d.). Retrieved November 22, 2022, from https://docs.oracle.com/javase/7/docs/technotes/guides/javadoc/deprecation/deprecation.html

How to Write Doc Comments for the Javadoc Tool. (n.d.). Retrieved November 22, 2022, from https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html#tag

Iammarino, M., Aversano, L., Bernardi, M. L., & Cimitile, M. (2019). A Topic Modeling Approach To Evaluate The Comments Consistency To Source Code. In *IEEE Xplore*.

Rani, P. (2021). Speculative Analysis for Quality Assessment of Code Comments. *Proceedings - International Conference on Software Engineering*, 299–303. https://doi.org/10.1109/ICSE-Companion52605.2021.00132

Real Academia Española. (n.d.). *Derecho de autor*. Retrieved November 7, 2022, from https://dle.rae.es/derecho#MYrPxAp

Steidl, D., Hummel, B., & Juergens, E. (2013). Quality Analysis of Source Code Comments. In *2013 IEEE 21st International Conference on Program Comprehension* (pp. 83–92). IEEE.

Wang, D., Guo, Y., Dong, W., Wang, Z., Liu, H., & Li, S. (2019). Deep Code-Comment Understanding and Assessment. *IEEE Access*, *7*, 174200–174209. https://doi.org/10.1109/ACCESS.2019.2957424

Woodfield, S. N., Dunsrnore, H. E., & Shen, V. Y. (1981). *The Effect of Modularization and Comments on Program Comprehension*.