Descripción de *code smells*: Enfoque ontológico

Ivian Laobel Castellano Betancourt[1[0000-0001-6459-0263]], Nemury Silega Martínez[1[0000-0002-8436-5650]], Katya Martha Faggioni Colombo[2[0000-0001-6577-5761]], Gilberto F. Castro[2,3[0000-0001-9050-8550]]

[1]Universidad de las Ciencias Informáticas, Habana, Cuba
{ilcastellano,nsilega}@uci.cu
[2] Universidad de Guayaquil, Guayaquil, Ecuador
{katya.faggionic, gilberto.castro}@ug.edu.ec
[3] Universidad de Católica Santiago de Guayaquil, Guayaquil, Ecuador
gilberto.castro@ucsg.edu.ec

Abstract. Los *code smells* son problemas asociados a fragmentos del código fuente de un software que, aunque no impiden que este funcione correctamente, deben ser reestructurados pues generan un impacto negativo en su calidad. Estas anomalías son comunes durante el desarrollo de software y generalmente suceden por la incorrecta aplicación de buenas prácticas en la programación, por parte de los desarrolladores y arquitectos de software con insuficiente conocimiento. Afectan las características de calidad del software como la mantenibilidad degradando la reusabilidad y la comprensibilidad. La coexistencia de estos errores en el código fuente provoca problemas de arquitectura que dificultan el proceso de mantenimiento y evolución del software aumentando los fallos en el futuro. Para abordar el problema descrito, en este artículo se presenta una ontología que permite describir los *code smells* como base de conocimiento. Para el desarrollo de la misma se utilizó la herramienta Protégé y se siguió una metodología para el desarrollo de ontologías, también se utilizó el razonador Pellet para inferir el conocimiento. Finalmente, la ontología podrá ser utilizada para capacitar a desarrolladores y arquitectos de software mediante una base de conocimiento que permitirá compartir y comunicar el conocimiento generado, así como la descripción de la información relacionada con los *code smells*.

Keywords: Descripción, *Code smells*, Ontología.

1 Introducción

En la actualidad, la industria del software se ha desarrollado a medida que evolucionan las tecnologías de la información. De igual manera, han surgido nuevos modelos y propuestas para obtener productos de software con calidad, cumpliendo las expectativas del cliente y ciertos requisitos de calidad [1].

La norma ISO/IEC 25010 determina las características de calidad de un software que se pueden evaluar, entre ellas se encuentra la mantenibilidad [2]. Esta se define como la capacidad del producto de software para ser modificado efectiva y eficientemente, por necesidades evolutivas, correctivas, perfectivas o adaptativas [2]. La evolución y mantenimiento de un software son tareas costosas en el desarrollo de software. Estos costos crecen mientras que los sistemas sean más grandes y complejos [3].

Una preocupación común que dificulta este proceso es la existencia de problemas de diseño estructural, que no fueron atendidos en etapas tempranas de desarrollo. Estos problemas se describen como *code smell* (CS). Este concepto fue atribuido a Kent Beck en el año 1999 [4] y luego, su marco conceptual se complementó con anomalías o malos "síntomas" a nivel de diseño de software.

Los CS son problemas estructurales en el código fuente de un software. Están asociados a fragmentos de código que deben reestructurarse para mejorar la calidad. Surgen por la incorrecta aplicación de buenas prácticas, lo que demuestra que el código esté pobremente escrito, tenga mala calidad y esté mal organizado o estructurado. En consecuencia, generan un impacto negativo en la calidad, sea difícil de extender, modificar y que dificulte el mantenimiento en etapas maduras del software. Además, ralentizan el desarrollo de software y aumentan el riesgo de errores o fallos en el futuro. Estas anomalías generalmente ocurren por el insuficiente conocimiento de los programadores y arquitectos con poca experiencia en el desarrollo de software [5].

Con el fin de conocer el comportamiento de los CS, la importancia que se le atribuyen y el estado actual de este tema en el desarrollo de software, se realizó un estudio empírico a partir de una encuesta. Esta se aplicó a programadores y arquitectos, en dos empresas de desarrollo de software en Cuba. Para la aplicación del instrumento se tuvo en cuenta los tipos de CS definidos por Fowler [4] y Lanza [6]. El cuestionario consta de dos partes. La primera parte recopila información de los encuestados acerca de antecedentes sobre las respuestas, como años de experiencia en la producción de software, lenguajes de programación conocidos y utilizados, proyectos en los que han participado y roles desempeñados. La segunda parte proporciona información referente a los CS y el seguimiento que se les realiza.

La encuesta demostró que solo el 47 % de los encuestados refieren poseer conocimientos básicos sobre los CS, aunque reconocen que en los proyectos que han participado se le atribuye poca importancia y se aborda poco sobre el tema. Esto demuestra que no se realiza un seguimiento adecuado a los CS por falta de preparación de los roles involucrados. Así mismo, se listan los tipos de CS más comunes en los proyectos mencionados. Además, el 100 % de los encuestados afirma que no se aplican buenas prácticas o alternativas para disminuir la ocurrencia de los CS en los proyectos que han participado. Se demostró la baja tasa de explotación de herramientas que apoyen la detección de CS, solo el 44 % afirman que se emplea la herramienta *SonarQube*, además se pudo comprobar que no se cuenta con especialistas capacitados para la correcta gestión de los CS. El resultado de la encuesta demuestra la necesidad de alternativas para capacitar a especialistas involucrados en la gestión de los CS.

Por otro lado, las ontologías sirven como instrumento útil para compartir y comunicar el conocimiento [7]. En la literatura se han documentado sus beneficios en diferentes áreas de la ingeniería del software [8]. Para explotar sus potencialidades, el objetivo de este trabajo es desarrollar una ontología que permita describir los CS como base de conocimiento. La ontología describirá los distintos tipos de CS, sus clasificaciones, además de las características de calidad y elementos o principios de diseño de software que afectan. Describirá los problemas de arquitectura que provocan ante la presencia de aglomeraciones de código. También, posibilitará compartir y comunicar el conocimiento explícito acerca del tema representado en la bibliografía científi-

ca. Tiene como propósito de brindar apoyo a los profesionales del desarrollo de software permitiendo capacitar a programadores y arquitectos.

Este artículo se enfoca en la creación de una ontología que permite describir los CS teniendo en cuenta las metodologías para el proceso de creación de ontologías. Se ha estructurado la información de la siguiente forma: la Sección 2 se presentan los conceptos básicos de la investigación y se describen los trabajos relacionados y en la Sección 3 se desarrolla la ontología para la descripción de CS. Finalmente, se presentan las conclusiones y trabajos futuros.

2 Trabajos relacionados

2.1 *Code smell*

Un CS es una estructura o "anomalía" en el código fuente que puede degradar diferentes aspectos de la calidad del código fuente de un sistema informático, como la reusabilidad, comprensibilidad y mantenibilidad. A pesar de que no impide el funcionamiento del software podría ralentizar su desarrollo, aumentar el riesgo de errores o llevar a la introducción de fallas en etapas maduras del producto informático [9-11].

Mäntylä [12] describe diferentes problemas comunes durante el desarrollo de software que pudieran ser causantes de que existan problemas de diseño y arquitectura [13]; se plantea que existen clases con una considerable cantidad de líneas de código debido a diferentes razones las cuales se destacan: innecesario código repetido, variables que no se utilizan y temporales, lo que lleva consigo bajos niveles de abstracción, diseño y reutilización. Se aborda la existencia de métodos con una lista excesiva de parámetros siendo difíciles de comprender e incrementan el acoplamiento.

En ocasiones es necesario realizar cambios en una clase, con lo cual se deben realizar varias modificaciones adicionales en diversos lugares para compatibilizar el cambio realizado, debido a esto, se pierde en tiempo de desarrollo. A su vez, se encuentran diferentes funcionalidades dentro de un sistema que dan solución a un mismo problema, teniendo como resultado que no se cumpla la reutilización de código, lo que conduce a que surjan problemas de rendimiento.

Teniendo en cuenta lo anterior y la importancia que se le atribuye al conocimiento de los CS por parte de los programadores y arquitectos de software se realizó una revisión de la literatura [4-6, 11, 14, 15], la cual permitió identificar un conjunto de catálogos de CS y clasificaciones de estos en cuanto a granularidad y similitud. Como resultado de la búsqueda se evidenció que entre los CS más abordados se encuentran los presentados por Fowler [4] y Lanza [6].

Varias investigaciones han demostrado el impacto de la identificación de CS en etapas tempranas del desarrollo de software y el análisis de estos para poder adoptar decisiones. Por ejemplo, Bastias [16] y López [2] hacen referencia a la herramienta *SonarQube*, siendo útil para el análisis de diferentes elementos de calidad y determina la deuda técnica de un software en cuanto a CS. Tiene como desventaja que los CS identificados contienen un alto número de falsos positivos y además esta herramienta tiene dificultades para analizar código que no se encuentra compilado [17].

Masson [15] realiza un análisis acerca de las herramientas *inCode* e *iPlasma*; donde se deduce que una de las mayores limitantes de estas es que usualmente identifican una cantidad considerable de CS; lo que podría volverse un problema en el desarrollo de software por las siguientes razones.

- El desarrollador pudiera verse abrumado por la cantidad de información a analizar; este sería quien decide en cada caso si se le debe dar tratamiento a los CS señalados, pues no todos indican un problema de diseño.
- Cada síntoma encontrado requiere de un análisis particular y teniendo en cuenta su entendimiento del sistema poder identificar aquellos que necesitan ser reparados.

Una vez analizados los resultados del estudio de estas herramientas, se constata que son aplicadas para la identificación de anomalías en el código fuente. Aunque, se abordan un conjunto reducido de CS. También, la precisión de detección varía entre cada una de las herramientas. En los ejemplos anteriores se evidencia que existe inestabilidad para priorizar y realizar análisis de detección de CS. Esto depende en gran medida de la importancia que se le atribuye al seguimiento de los CS durante el desarrollo de software, además de la experiencia y el conocimiento que tengan los programadores y arquitectos.

2.2 Ontología

La ontología es una disciplina de la filosofía que trata con lo que es, con los tipos y estructuras de objetos, propiedades y otros aspectos de la realidad [18]. En la rama de la informática se define una ontología como: *"Una descripción formal y explícita de los conceptos en un dominio de discurso, las propiedades de cada concepto que describen los rasgos y atributos del concepto y las restricciones de los slots"*[7]. Una ontología con un conjunto de instancias individuales de clases constituye una base de conocimiento [7]. En la literatura científica existen estudios de soluciones basadas en ontologías, adecuados para compartir y comunicar el conocimiento en diferentes dominios [7]. Motivado por estos resultados se determinó que el desarrollo de una ontología constituye una solución interesante para abordar los problemas analizados referente a la descripción y análisis de los CS.

Partiendo de la exploración efectuada en la literatura científica acerca de la conceptualización de los CS y sus diferentes comportamientos durante el desarrollo de software, se elabora una ontología basada en la gestión de conocimiento. Tiene como principal utilidad la descripción de los CS en sus diferentes escenarios. La solución propuesta permite describir y compartir su base de conocimiento, la cual es útil para los programadores y arquitectos de software, dado que ayuda a conocer y comprender a mayor escala los CS, facilitando su identificación y análisis en etapas tempranas de desarrollo, así como determinar que tratamientos aplicarles.

3 Ontología para la descripción de *code smells*

Entre los lenguajes más populares para especificar ontologías se distingue OWL

por las facilidades que brinda. Está basado en un modelo lógico que le permite definir los conceptos tal y como son descritos. Utiliza razonadores que permite chequear automáticamente la consistencia de los modelos representados. Para elaborar la ontología se utiliza la herramienta Protégé 5.0 para representar el conocimiento, la cual posee una arquitectura flexible y extensible [19]. Se utiliza el razonador Pellet pues permite realizar las inferencias en el Protégé sobre las instancias y comprobar las propiedades lógico-formales validando la consistencia de la ontología.

Para alcanzar resultados satisfactorios en el desarrollo de la ontología, se debe utilizar una metodología que guíe el proceso de su construcción. En este trabajo se adopta la metodología propuesta por Alvarado [20], creada a partir de los principios y buenas prácticas de las metodologías: *Methontology* [21] y *Desarrollo de ontologías-101* [7]. La metodología propone cinco actividades: 1) determinación de los requerimientos de la ontología, 2) reutilización de ontologías, 3) elaboración del modelo conceptual, 4) implementación y 5) evaluación de las ontologías.

A partir de los conceptos identificados y explicados en los requerimientos de la ontología se elaboró el modelo conceptual haciendo uso de estos conceptos, así como las relaciones entre ellos. En la Tabla 1 se muestran estos conceptos y describen los principales conceptos utilizados. En la Tabla 2 sus relaciones con otros conceptos.

Tabla 1. Términos representativos del dominio. Fuente: elaboración propia

Nombre	**Descripción**
Tipo de *code smell*	Problemas estructurales en el código fuente de un software. Están asociados a fragmentos de código que deben reestructurarse para mejorar la calidad. Surgen por la incorrecta aplicación de buenas práctica [6, 7].
Clasificación de *code smell*	Términos que agrupan a los CS en cuanto a la similitud que podrían tener entre ellos.
Aglomeración de código	Grupo de CS que están relacionadas entre sí por alguna razón. La relación entre dos CS puede establecerse, por ejemplo, a través de llamadas a métodos o herencia [22].
Anomalía de arquitectura	Síntoma de degradación de la arquitectura de un software en su implementación a través de la introducción progresiva de CS [22].
Norma de calidad	Normas internacionales de calidad [2].
Caracteríctica de calidad	Características de calidad que establecen las normas de calidad para un producto de software [2].
Subcaracterís-tica de calidad	Subcaracterísticas de calidad contenidas dentro de las características de calidad [2].
Elemento de diseño	Principios, buenas prácticas o patrones de diseño.
Herramienta	Herramientas que son utilizadas para la identificación de CS.
Métrica	Métricas utilizadas en las herramientas para la identificación de los CS.
Sistema	Producto de software que pudiera tener problemas de arquitectura debido a la ocurrencia de CS.
Componente	Pieza de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar en un sistema.
Clase	Plantilla para la creación de objetos de datos según un modelo predefinido.
Método	Subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como a un objeto, como es el caso de los métodos de instancia.
Lenguaje de programación	Lenguaje formal que proporciona instrucciones que permiten al programador escribir secuencias a modo de controlar el comportamiento físico y lógico de una computadora con el objetivo de que produzca diversas clases de datos.

Tabla 2. Conceptos de la ontología y sus relaciones. Fuente: elaboración propia

Concepto	Relación	Concepto
Tipo de *code smell*	Afecta	Elemento de diseño
		Caracteríctica de calidad
Clasificación de *code smell*	Clasifica a	Tipo de *code smell*
	Compuesta por	Tipo de *code smell*
Aglomeración de código	Está presente en	Componente
	Provoca	Anomalía de arquitectura
Norma de calidad	Contiene	Caracteríctica de calidad
Caracteríctica de calidad	Contiene	Subcaracteríctica de calidad
Herramienta	Identifica	Tipo de *code smell*
		Anomalía de arquitectura
	Soporta	Lenguaje de programación
	Utiliza	Métrica
Métrica	Puede identificar	Tipo de *code smell*
Componente	Tiene	Método
		Clase
Sistema	Tiene	Componente
	Utiliza como tegnología	Lenguaje de programación
Clase	Tiene	Método

Las clases, propiedades, restricciones e instancias son los componentes más importantes de una ontología. Una vez detallados los conceptos relacionados con los CS se realizó un análisis para identificar cada uno de los elementos de la ontología a desarrollar. La Fig. 1 muestra un fragmento de las 22 clases identificadas.

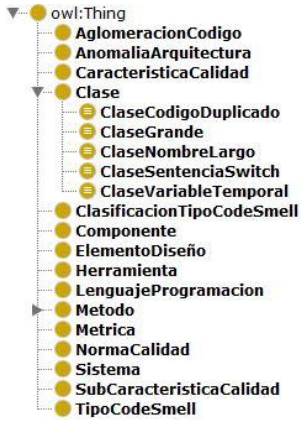

Fig. 1. Clases de la ontología para describir los CS. Fuente: elaboración propia

Una vez identificadas cada una de las clases de la ontología y modeladas en la herramienta, se establecieron las propiedades de cada una de ellas. Las propiedades

pueden ser de dos tipos: propiedad de objeto (*object property*) o de dato (*data property*). Las propiedades de objetos permiten restringir las relaciones de un individuo.

En la Fig. 2 se muestra un fragmento de las 36 propiedades de objetos representadas en la ontología desarrollada mediante una representación de relaciones binarias entre estas propiedades.

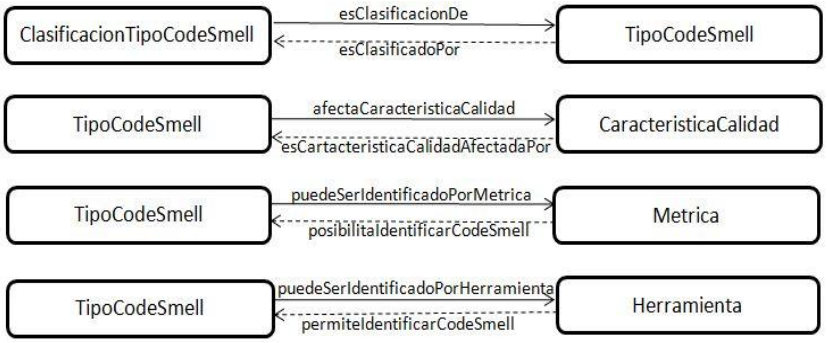

Fig. 2. Propiedades de objeto de la ontología. Fuente: elaboración propia

Con el objetivo de describir los conceptos se definieron en la ontología los atributos de las clases. En la Fig. 3 se muestran las propiedades de dato de la ontología.

Fig. 3. Propiedades de dato de la ontología. Fuente: elaboración propia

4 Conclusiones y trabajo futuro

La revisión de la literatura permitió analizar los tipos de CS y su relación con otros conceptos. La aplicación de una encuesta en proyectos de desarrollo de software teniendo en cuenta la experiencia de los encuestados, permitió conocer los tipos de CS que a menudo inciden en el código fuente de los proyectos, la baja tasa de explotación de herramientas que apoyen la detección de CS, así como la necesidad de aplicar alternativas como soporte para la capacitación de programadores y arquitectos en aras de realizar una adecuada gestión de los CS. Con el uso de la herramienta Protégé y el lenguaje OWL se elaboró una ontología que permite representar el conocimiento. La utilización de una metodología para el desarrollo de ontologías permitió guiar el pro-

ceso de construcción teniendo en cuenta las actividades propuestas.

Como trabajo futuro se pretende elaborar un método que incluya secuencias de actividades para el proceso de utilización, mantenimiento y evolución de la ontología. También, se validará la solución mediante la evaluación de la ontología en proyectos de desarrollo de software y la aplicación de un estudio de caso como método de investigación científica. Se pretende que la ontología creada se utilice en proyectos de desarrollo de software y pueda ser empleada para capacitar a programadores y arquitectos.

Referencias

1. González, L.A.E., N.J. Acosta, and J.L.G. Tovar, *Estándares para la calidad de software.* Tecnología Investigación y Academia, 2017. **5**(1): p. 75-84.
2. López, J.V., *AUDITORÍA MANTENIBILIDAD APLICACIONES SEGÚN LA ISO/IEC 25000*. 2015.
3. April, A. and A. Abran, *Software maintenance management: evaluation and continuous improvement*. Vol. 67. 2012: John Wiley & Sons.
4. Fowler, M., et al., *Refactoring: improving the design of existing code*. 1999: Addison-Wesley Professional.
5. Malavolta, A., *Análisis de detección de Code Smells para el lenguaje JavaScript.* 2018.
6. Lanza, M. and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. 2007: Springer Science & Business Media.
7. Noy, N.F. and D.L. McGuinness, *Ontology development 101: A guide to creating your first ontology*. 2001, Stanford knowledge systems laboratory technical report KSL-01-05 and ….
8. Garzás, J. and M. Piattini, *An ontology for microarchitectural design knowledge.* IEEE software, 2005. **22**(2): p. 28-33.
9. Yamashita, A. and L. Moonen. *Do code smells reflect important maintainability aspects?* in *2012 28th IEEE international conference on software maintenance (ICSM)*. 2012. IEEE.
10. Di Nucci, D., et al. *Detecting code smells using machine learning techniques: are we there yet?* in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2018. IEEE.
11. Alonso, M. and F.H. Klaver, *Aplicación de un Proceso de Refactoring guiado por Escenarios de Modificabilidad y Code Smells.* 2016.
12. Mäntylä, M.V. and C. Lassenius, *Subjective evaluation of software evolvability using code smells: An empirical study.* Empirical Software Engineering, 2006. **11**(3): p. 395-431.
13. Aniche, M., et al., *Code smells for model-view-controller architectures.* Empirical Software Engineering, 2018. **23**(4): p. 2121-2157.
14. AlKharabsheh, K., et al. *Comparación de herramientas de Detección de Design Smells*. 2016. JISBD.
15. Masson, I. and R. Pastore, *Análisis y mejoras de usabilidad y performance sobre una herramienta de asistencia en el refactoring de aplicaciones.* 2016.
16. Bastias, O.A., *Código con"mal olor": un mapeo sistemático.* Revista Cubana de Ciencias Informáticas, 2018. **12**(4): p. 156-176.
17. Ospina Delgado, J.P., *Análisis de seguridad y calidad de aplicaciones (Sonarqube).*
18. Welty, C. and N. Guarino, *Supporting ontological analysis of taxonomic relationships.* Data & Knowledge Engineering, 2001. **39**(1): p. 51-74.
19. Fernández Hernández, A., *Modelo ontológico de recuperación de información para la toma de decisiones en gestión de proyectos.* 2016.
20. Alvarado, R. *Metodología para el desarrollo de ontologías. 2010.* 28 noviembre 2018]; Available from: https://es.slideshare.net/Iceman1976/metodologia-para-ontologias.
21. Fernández-López, M., A. Gómez-Pérez, and N. Juristo, *Methontology: from ontological art towards ontological engineering.* 1997.
22. Oizumi, W.N., et al., *On the relationship of code-anomaly agglomerations and architectural problems.* Journal of Software Engineering Research and Development, 2015. **3**(1): p. 11.