SMELLWARE: un juego para la enseñanza de buenas prácticas en el proceso de desarrollo de software

SMELLWARE: a game for teaching best practices within the software development process

Carlos Mario Zapata Jaramillo¹ María Clara Gómez Álvarez^{2*} Juan Carlos Hernández Palencia¹

Recibido 15 de julio de 2020, aceptado 1 de septiembre de 2020

Received: July 15, 2020 Accepted: September 1, 2020

RESUMEN

En el contexto del desarrollo de software, los *smells* son porciones de código fuente que sugieren la presencia de problemas más profundos en el sistema, como lo es el código duplicado. La identificación, corrección y prevención de código incorrecto (*smells*) en los proyectos de *software* requiere conocimiento y habilidades específicas del proceso de desarrollo que se pueden aprender, entrenar y afinar. Algunos autores cuentan con evidencias donde se muestra que el uso de juegos educativos permite motivar, enseñar, entrenar y afinar distintas habilidades. Usando los principios de los juegos educativos, algunos autores proponen estrategias para que los equipos de desarrollo de *software* adopten buenas prácticas en su qué hacer, pero dichas propuestas no tienen en cuenta distintos tipos de *smells* y las estrategias para corregirlos. En este artículo se propone *Smellware* como un juego para la enseñanza de buenas prácticas en el proceso de desarrollo de *software*. Igualmente se describe la estructura del juego y los resultados de su aplicación en estudiantes de ingeniería de *software*. Esta propuesta busca que los desarrolladores de *software* reconozcan distintos tipos de *smells* y adquieran mayor conciencia sobre los efectos de estas malas prácticas en el desarrollo de *software*.

Palabras clave: Juegos educativos, buenas prácticas, desarrollo de *software*, código incorrecto.

ABSTRACT

In software development, smells are parts of source code that suggest a deeper problem in the system such as duplicated code. Specific knowledge and skills are required for identifying, correcting, and preventing bad code (smells) in the software development process. Such knowledge and skills can be learned, trained, and tuned. Some researchers have evidence about the usage of educational games for motivating, teaching, training, and improving several skills. By using principles of educational games, some authors propose strategies for software development teams to adopt good practices, but these proposals suffer to show different types of code smells and strategies to correct them. In this paper we propose Smellware as a game for teaching best practices in the software development process. The paper describes the game structure and the results of its application in students of software engineering. This proposal aims to strengthen the skills of software developers for identifying and addressing several types of bad code smells and raise awareness about the effects of these bad practices in software development.

Keywords: Educational games, best practices, software development, code smells.

¹ Universidad Nacional de Colombia. Departamento de Ciencias de la Computación y de la Decisión. Medellín, Colombia.
E-mail: cmzapata@unal.edu.co; jcherna0@unal.edu.co

² Universidad de Medellín. Facultad de Ingenierías - Ingeniería de Sistemas. Medellín, Colombia.
E-mail: mcgomez@udem.edu.co

* Autor de correspondencia: mcgomez@udem.edu.co

INTRODUCCIÓN

Los *bad code smells*, *code smells* o simplemente *smells* son porciones del código fuente de un programa que suelen sugerir la presencia de problemas más grandes en todo el sistema. La presencia de *smells* en un proyecto de *software* no implica que el programa no cumpla los requisitos funcionales y tampoco señala la presencia de errores en el sistema, pero sí indica debilidades en el diseño del *software* que aumentan el riesgo de fallas [1]. Un ejemplo de un *smell* puede ser el código fuente duplicado o la presencia de métodos demasiado largos. Como se trata de consecuencias que no son inmediatas, el desarrollador puede generar *software* que cumple con los requisitos exigidos, pero en el mediano y largo plazo se evidencian estas dificultades. Por ello, la identificación de estas malas prácticas requiere conocimiento y habilidades específicas para construir código que prevenga estas situaciones.

Por otro lado, los juegos educativos se usan como alternativa para aumentar la efectividad y adherencia en los procesos de aprendizaje [2, 3]. Algunos autores proponen juegos educativos para motivar, enseñar, entrenar y afinar distintas habilidades requeridas [4]. Estas propuestas de juegos educativos enseñan conocimientos y promueven habilidades requeridas en distintas fases del proceso de desarrollo de *software*. Sin embargo, en estos acercamientos aún hace falta hacer explícitas las distintas manifestaciones de los *code smells*, así como las mejores prácticas para solucionarlos.

En este artículo se propone *Smellware* como un juego para la enseñanza de buenas prácticas en el proceso de desarrollo de *software* y se describen sus componentes principales como son: objetivos de aprendizaje, materiales y reglas del juego. Finalmente se presentan los resultados de la aplicación de *Smellware* a dos grupos de estudiantes de ingeniería de sistemas donde se evidencia el potencial del juego para generar mayor conciencia en los desarrolladores sobre la importancia del código limpio.

El desarrollo de este trabajo se estructura de la siguiente manera: en la primera Sección se brinda contexto sobre la importancia de las buenas prácticas en el desarrollo de *software*. En la segunda sección se continúa con el análisis de los juegos educativos que enseñan buenas prácticas en el proceso de

desarrollo de *software*. En la tercera sección se explica la propuesta *Smellware* (un juego para la enseñanza de buenas prácticas en el proceso de desarrollo de *software*) y, finalmente, se discuten los resultados, las conclusiones y el trabajo futuro en las Secciones cuarta y quinta respectivamente.

MARCO TEÓRICO

Los *smells*, en el contexto del desarrollo de *software*, hacen referencia a partes del código fuente de un programa que pueden representar inconvenientes mucho más profundos en toda la solución. Los *smells* se caracterizan por ser síntomas rápidamente identificables en porciones del código fuente [1]. Como ejemplos de estos *smells* se puede considerar el código duplicado o la presencia de clases o métodos demasiados largos.

La presencia de *smells* en un proyecto de *software* no necesariamente implica que el programa no cumpla los requisitos funcionales o incluso que señale la presencia de bugs, pero sí indica debilidades en el diseño del *software* que aumentan el riesgo de fallas. Estos *smells* pueden representar deficiencias en el desempeño del *software* y esfuerzos adicionales innecesarios para adecuaciones futuras y dificultades en el mantenimiento que pueden incurrir en la introducción de errores [5]. Para el caso del *smell* de código fuente duplicado, éste representa un esfuerzo adicional por parte del equipo desarrollador ya que debe soportar la misma funcionalidad en dos o más partes del sistema y adicionalmente incrementa la posibilidad de inyección de errores ya que las actualizaciones pueden no realizarse en todas las partes del sistema donde sea requerido. Para el caso del *smell* de métodos demasiado largos dan indicios de que el método tiene más de una responsabilidad lo que incrementa la complejidad y disminuye la cohesión, sin considerar el esfuerzo adicional por parte de los desarrolladores para entender y modificar la o las funciones de las que dicho método es responsable. Como se trata de consecuencias que no son inmediatas, el desarrollador puede generar *software* que cumple con los requisitos exigidos, pero en el largo y mediano plazo se evidencian estas dificultades. Generando así riesgos y sobreesfuerzos que afectan la calidad del producto generado y aumentan sus costos en la medida que la detección de estos errores se aleja del momento en que fueron introducidos [6]. Por ello, la identificación de estas

malas prácticas (*smells*) requiere conocimiento de heurísticas [7] y habilidades específicas como el *refactoring* [8] para construir código que prevenga estas situaciones. En algunos casos los mismos desarrolladores no son conscientes de los riesgos que conllevan estos *smells* [9].

Algunos autores promueven las siguientes ventajas al momento de usar los juegos como una herramienta de enseñanza. Los juegos educativos desarrollan habilidades de comunicación y promueven el aprendizaje por la interacción entre pares [10]. Estos juegos estimulan el pensamiento crítico y la toma de decisiones de forma práctica [11]. Los juegos educativos incrementan la velocidad de aprendizaje y mejoran la adquisición de conocimientos [12].

ANTECEDENTES

A través de los juegos educativos algunos autores proponen estrategias para que los equipos de desarrollo de *software* adopten distintas buenas prácticas en las diferentes etapas del desarrollo de *software*.

Killer App. [13] es un método propuesto como juego de mesa, que busca que los participantes entiendan el proceso de aseguramiento de la calidad en el desarrollo de *software*. En esta iniciativa el jugador cuenta con un conjunto de cartas que representan estrategias y habilidades propias del proceso de aseguramiento de calidad de *software*. El jugador debe administrar estas cartas para lograr la mayor rentabilidad para la empresa de desarrollo que representa, pero con los mejores estándares de calidad. Este juego enfatiza en las habilidades necesarias para el aseguramiento de la calidad y estrategias para lograrlo, pero donde el mismo jugador puede obtener una carta que mejora la calidad del *software*. Esta mecánica no incluye un contexto sobre las razones por las cuales el *pair programming* es favorable para el aseguramiento de la calidad o en qué casos se puede aplicar y qué tipo de resultados se puede esperar de aplicar esta práctica.

The Hard Choices [14] es un juego de mesa donde se simula el proceso de desarrollo de *software*. Este juego comunica a los participantes los conceptos de incertidumbre, riesgos y deuda técnica. La mecánica del juego constantemente invita al jugador a decidir si desea avanzar más rápido en el juego para

lograr los objetivos económicos esperados o si el jugador desea prepararse para enfrentar riesgos que posteriormente se puedan materializar y castigar su avance. A pesar de que el juego cuenta con tarjetas que representan las buenas prácticas para evitar que los riesgos por mala calidad se materialicen, no presenta retroalimentación sobre la mejor forma de usar estas herramientas.

Anukarna [15] es un juego que representa la realización de un proyecto de *software*. El jugador se somete a una serie de decisiones durante el proceso de desarrollo donde en cada decisión afecta cuatro variables del proyecto: tiempo, presupuesto, calidad y deuda técnica. La idea del juego es lograr finalizar el proyecto dentro del presupuesto, con la máxima calidad, la menor deuda técnica y en el tiempo estimado. Esta dinámica está enfocada en la estrategia de la revisión por pares (*code review*) donde un par del equipo de desarrollo de *software* valida el código construido por otro integrante del equipo de trabajo. El juego es asistido por computador, donde a través de la simulación se muestra al jugador distintos escenarios propios del proceso de revisión de código y allí se invita al jugador para que tome una serie de decisiones sobre las prácticas que debe adoptar al momento de realizar la revisión de código fuente de sus pares. De esta forma el juego puntúa el mejor desempeño, si el jugador ha tomado la decisión de adoptar la mejor práctica según cada escenario. A pesar de que esta iniciativa muestra buenas prácticas en la revisión de código fuente, no ofrece la posibilidad de reconocer distintos *smells* que también harían parte del proceso de revisión por pares.

Game Competitive Source Control [16] es una propuesta de ludificación del proceso de desarrollo de *software* donde se identifican distintos *smells* a través de análisis estático de código fuente, posterior a su identificación, estos *smells* se puntúan y se les realiza el seguimiento a través del sistema de control de versiones, logrando así identificar y asignar el puntaje correspondiente al integrante del equipo de desarrollo que ha eliminado algún *smell*. Complementándolo con una lista de los desarrolladores con mayor puntaje, se incentiva a través de la competencia que se logre desarrollar aplicaciones de mejor calidad. Teniendo en cuenta que ya se han identificado los *smells* presentes en una aplicación, esta iniciativa carece del factor de

retroalimentación hacia el desarrollador para que logre identificar el por qué ese segmento de código ha sido señalado como un *code smell* y adicionalmente esta estrategia deja de lado la posibilidad de que el desarrollador aprenda sobre las formas como se puede solucionar un determinado tipo de *smell*.

FindBugs [17] es una herramienta que se alimenta del resultado del análisis estático de código fuente. La herramienta identifica bugs y *smells*, asignando puntos al desarrollador cuando soluciona alguno de los bugs identificados y mantiene actualizada una lista con los puntajes de todos los desarrolladores del proyecto. A través del factor competencia estimula la eliminación de bugs y *smells* en la aplicación. También indica cuáles son las malas prácticas y promueve el hecho de disminuir los bugs, pero no retroalimenta sobre la mejor manera de resolverlos o no premia si el bug se resuelve usando buenas prácticas. Se limita a asignar un puntaje a los desarrolladores en la medida en la que eliminan los bugs. A pesar de que indica la presencia de *smells* no muestra al desarrollador por qué eso es un *code smell*, ni cuáles serían las alternativas para resolverlo.

Estos acercamientos de juegos educativos tienen como objetivo la enseñanza de buenas prácticas en el proceso de desarrollo de *software*. Este objetivo lo logran al incorporar estrategias para incentivar la corrección de bug y *smells* por parte de los participantes. Sin embargo, estas propuestas no muestran cuáles son los distintos tipos de *smells* y cuáles son las mejores prácticas para resolverlos. Teniendo presente estas experiencias y con el ánimo de concientizar a los desarrolladores sobre los distintos *smells* y sus implicaciones, se propone Smellware.

PROPUESTA

Los juegos educativos son una de las estrategias que se proponen para que los equipos de desarrollo de *software* adopten distintas buenas prácticas en las distintas etapas del desarrollo de *software*.

Como se ha identificado en las distintas propuestas descritas en la Sección de Antecedentes, todas ellas enseñan el uso de buenas prácticas en varios niveles del proceso de desarrollo de *software*, pero si bien enseñan y estimulan el uso de estas buenas prácticas, dejan de lado retroalimentar al desarrollador para

indicarle cuáles son las incidencias en el código fuente a las que le debe prestar atención y cuáles son las mejores alternativas para solucionar estas incidencias.

A partir de lo anterior nace la propuesta de construir *Smellware* como un juego para mostrar distintos tipos de *smells* y las mejores prácticas para solucionarlos. *Smellware* está basado en un juego educativo llamado *Riskware* [18]. *Riskware* es un juego de mesa donde el tablero representa la ejecución de un proyecto en el que se pueden materializar distintos riesgos que afectan el avance del equipo de trabajo. En dicho juego, los integrantes del equipo cuentan con un presupuesto inicial para comprar recursos y controles que les permitan sortear los riesgos que se puedan materializar durante el desarrollo del juego. No conforme con esto, cada equipo de trabajo debe definir previo al inicio del juego, la mejor ruta a seguir dentro del tablero de riesgos del proyecto para que en función de los recursos y controles adquiridos, puedan avanzar en el proyecto con la menor cantidad de imprevistos generados por estos riesgos.

Los imprevistos en la ejecución de un proyecto y los *code smells* en el código fuente se logran representar como riesgos que impactan de forma negativa el avance de un proyecto. En este caso ambos corresponden a riesgos que se mitigan mediante una previa preparación. Haciendo uso de esta analogía, se diseña el juego de *Smellware* con los siguientes componentes:

Objetivos de aprendizaje

Al finalizar este juego, los participantes deben estar en capacidad de:

- Identificar *smells*, así como algunas de las mejores prácticas para resolverlos.
- Reconocer los riesgos que representan los *code smells* en una solución informática.

Materiales

El juego consta de los siguientes materiales:

- **Tablero principal:** representa la ejecución del proyecto de desarrollo donde se podrán encontrar los distintos *smells* (Ver Figura 1). La selección inicial de *smells* se realizó a partir de la revisión de un conjunto de artículos sobre el tema. Sin embargo, luego de varias iteraciones del juego identificamos que lo más conveniente era usar

Fuente: Elaboración propia.

Figura 1. Tablero principal *Smellware*.

smells cuyos nombres fueran autodescriptivos, es decir que con tan sólo leer el nombre del *smell*, el participante tuviera una idea de qué se trataba, y así sucesivamente se filtraron hasta los que hoy se encuentran presentes en el juego propuesto. Todos los equipos participantes deben tener visibilidad del tablero. Los *smells* se distribuyen aleatoriamente en el tablero. A cada equipo de participantes le corresponde una ficha de color que indicará su avance en el proyecto.

- **Hoja de Planeación:** Cada equipo de participantes cuenta con una hoja de planeación donde podrá trazar la ruta que seguirá para finalizar el proyecto. Esta hoja de planeación tiene la misma forma del tablero principal (Ver Figura 2).
- **Tabla de precios de capacitaciones:** Corresponde a la lista de las mejores prácticas para resolver los *smells* presentes en el tablero principal. Cada equipo participante cuenta con esta lista de capacitaciones. Cada una de estas capacitaciones tiene un valor que representa el precio para poder adquirir dicha capacitación (Ver Figura 3).

Reglas

El juego se presenta en dos etapas. La primera (etapa de planeación) corresponde a la organización de equipos de trabajo y planeación del proyecto. La segunda (etapa de ejecución) corresponde a la ejecución del proyecto:

Fuente: Elaboración propia.

Figura 2. Hoja de planeación *Smellware*.

Etapas de planeación

La etapa de planeación se compone de las siguientes actividades:

1. Distribuir los participantes en equipos de trabajo.
2. El tablero principal representa el proyecto de construcción de un programa. El lado izquierdo

Fuente: Elaboración propia.

Training & value	Smells
\$10 : Decompose Conditional	Long Method
\$10 : Duplicate Observed Data	Large Class
\$30 : Extract Class	Duplicated Code, Large Class
\$10 : Extract Interface	Large Class
\$30 : Extract Method	Duplicated Code, Long Method
\$10 : Extract Subclass	Large Class
\$20 : Form Template Method	Duplicated Code
\$10 : Introduce Assertion	Comments
\$20 : Introduce Null Object	Switch Statements
\$30 : Introduce Parameter Object	Long Method, Long Parameter List
\$20 : Move Method	Switch Statements
\$20 : Pull Up Field	Duplicated Code
\$10 : Rename Method	Comments
\$10 : Replace Array with Object	Primitive Obsession
\$20 : Replace Conditional with Polymorphism	Switch Statements
\$10 : Replace Data Value with Object	Primitive Obsession
\$10 : Replace Method with Method Object	Long Method
\$10 : Replace Parameter with Method	Long Parameter List
\$10 : Replace Type Code with Class	Primitive Obsession
\$30 : Replace Type Code with Subclasses	Primitive Obsession, Switch Statements
\$20 : Substitute Algorithm	Duplicated Code

Figura 3. Tabla de precios de capacitaciones.

del tablero representa el inicio del proyecto, su lado opuesto el final del proyecto. El equipo de jugadores que primero logre pasar a través del tablero desde el inicio hasta el final del proyecto será quien gane el juego. Es importante mencionar que a los equipos participantes no se les definen requisitos u órdenes de trabajo. A cada uno de estos equipos se les indica que su objetivo es finalizar su proyecto de desarrollo de software que es representado en el tablero de juego con una fecha de inicio y fecha de fin.

3. Cada equipo cuenta con: una hoja de planeación, una tabla de precios y un presupuesto inicial de \$40. Este presupuesto lo deben invertir en las capacitaciones para

solucionar los *smells* que aparecerán en el desarrollo del proyecto.

4. El avance en el proyecto se realiza recorriendo los vértices internos del tablero. Estos vértices son los puntos de intersección de 3 *smells*. Al momento de que un equipo de desarrollo se ubica en uno de estos vértices, debe solucionar uno de estos tres *smells*.
5. Antes de iniciar el juego, los equipos de trabajo contarán con 7 minutos donde deben seleccionar las capacitaciones que desean adquirir bajo el presupuesto asignado y deben señalar la ruta a seguir en su hoja de ruta.
6. Una vez todos los equipos han seleccionado sus capacitaciones y diseñado el plan de trabajo, inicia la ejecución del proyecto.

Etapas de ejecución

La etapa de ejecución del juego consta de las siguientes actividades:

7. En cada iteración cada equipo de desarrollo tiene la oportunidad de avanzar en el desarrollo del proyecto implementando un requisito.
8. Al momento de implementar un requisito (avanzar un vértice en el tablero principal), podrá materializarse un *smell*.
9. Cada vértice posee tres *smells* vecinos, cuando un equipo de desarrollo se ubica en un vértice, se define aleatoriamente (por ejemplo, con un dado) cuál de los tres *smells* vecinos será el que se deba resolver.
10. Si el equipo cuenta con la capacitación adecuada para solucionar un *smell*, el equipo podrá continuar en la próxima iteración. En caso contrario, el equipo perderá la oportunidad de seguir implementando requisitos en la próxima iteración y así le dará la posibilidad a que los equipos contrincantes se adelanten.
11. Algunos *smells* están marcados como de alto impacto (HIGH), en caso de que un equipo no esté preparado para solucionar uno de

estos *smells*, perderá la oportunidad de seguir avanzando por 2 iteraciones consecutivas.

12. En la medida en la que se materialicen los distintos *smells*, el moderador del juego explica un ejemplo de en qué consiste este tipo de *smells* y cuál la estrategia para solucionarlo.

RESULTADOS

El juego *Smellware* fue aplicado a dos grupos de estudiantes de ingeniería de sistemas en el semestre 02-2016 en la Universidad Nacional de Colombia y el Instituto Tecnológico de Antioquia, para un total de 35 estudiantes. Una vez finalizado el juego se aplicó una encuesta como instrumento de evaluación para medir el nivel de satisfacción y aprendizaje percibido. La encuesta consta de 4 preguntas: ¿Qué tan cercano a la realidad le pareció el juego? ¿Qué factor de diversión le asignaría al juego? ¿Qué tan simple de jugar le pareció el juego? ¿Qué aprendió del juego?

En la Figura 4 se presentan los resultados obtenidos. De estos resultados se puede concluir que respecto

Fuente: Elaboración propia.

Figura 4. Resultados instrumento de evaluación.

a la variable *nivel de realismo* el 94% de los participantes consideran a *Smellware* un juego cercano a la realidad, lo que se debe al hecho que se simula un proyecto de desarrollo de software que se ejecuta por iteraciones, tal como ocurre en la actualidad con la implementación de metodologías como Scrum o AUP.

Frente al factor *de diversión* del juego, el 88% de los participantes lo califica como bueno, muy bueno o excelente, lo que da cuenta de que el juego cumple con su propósito de combinar aprendizaje con entretenimiento.

La tercera variable por la que se consultó a los participantes fue la facilidad de jugar *Smellware*; en este caso el 97% de los participantes consideran que el juego es fácil de jugar, lo que demuestra el potencial del juego para presentar la importancia de identificar y solucionar los *code smells* en los productos de desarrollo de software que se construyen en equipos de trabajo.

Finalmente, frente a la pregunta ¿qué aprendió del juego?, de carácter abierto, se realiza una asociación de las respuestas de los participantes por similitud. En este caso 29 participantes manifiestan haber mejorado sus habilidades para identificar y solucionar *code smells*, contribuyendo a incrementar la conciencia en los futuros profesionales sobre la importancia de desarrollar código limpio y no sólo programas que cumplan con los requisitos funcionales definidos con los interesados. Algunas respuestas de los participantes a esta pregunta fueron: “En ocasiones es inevitable evitar los *smells*, pero la mayoría de las veces podemos hacerlo aplicando las prácticas adecuadas”, “cómo identificar y dar solución a errores en el código fuente” y “que no se debe abusar de los primitivos, que la lógica que desarrollemos debe estar tan entendible que no debamos dejar comentarios”.

CONCLUSIONES Y TRABAJO FUTURO

Los juegos educativos representan una alternativa para promover el trabajo en equipo, el pensamiento, la comunicación y la creatividad. En este artículo se propone el juego *Smellware* que busca que los participantes conozcan sobre *code smells*, las buenas prácticas para solucionarlos e identificar los efectos de estas prácticas durante el desarrollo de un proyecto de *software*.

Esta actividad se puede replicar imprimiendo el tablero principal (Ver Figura 1), la hoja de planeación (Ver Figura 2) y la tabla de precios (Ver Figura 3). Adicionalmente con un generador de números aleatorios entre el uno y el tres y fichas que representen cada equipo participante, se puede reproducir la experiencia en cualquier espacio y con el mismo efecto.

Este tipo de ejercicios concientiza a los desarrolladores para suplir esas brechas de conocimiento sobre las buenas prácticas en la construcción de *software*. Adicionalmente, con este tipo de prácticas los desarrolladores novatos adquieren conciencia sobre lo que son las malas prácticas y sus consecuencias en el desarrollo de *software*.

A partir de las aplicaciones del juego realizadas en los años 2016 y 2017 identificamos que no es suficiente que los participantes tengan la noción de qué son los *smells* y cómo resolverlos. Sino que también es necesario desarrollar juegos donde los participantes experimenten lo que representa resolver alguna de estas dificultades. Es por ello por lo que en la actualidad venimos trabajando en el diseño de un nuevo juego más elaborado que no requiera la interacción de un facilitador para que a través de la metáfora de la deuda técnica y la ejecución de un proyecto los participantes se vean confrontados en la toma de decisiones sobre la calidad del software. De manera general en este juego todos los equipos deberán construir partes o módulos de un mismo producto de software y se verán enfrentados a la solución de *bugs* y *smells* generados por otros equipos.

Como trabajos futuros se identifican:

1. Aplicar el juego en entornos empresariales para entrenar a desarrolladores de proyectos de software reales.
2. Modificar el juego para que al momento de materializarse un *smell*, se muestre el código fuente con dicho síntoma y sean los participantes quienes deban identificar cuál *code smell* se trata y cuál sería la mejora práctica para solucionarlo.
3. Agregar niveles de dificultad para que sea adaptable no sólo para estudiantes, sino también para desarrolladores con mayor experiencia.
4. En el código fuente cada tipo de *smell* se puede materializar de forma diferentes y pueden existir varias formas de solucionarlos. Una línea de

trabajo futuro es diseñar juegos educativos que simulen efectos de *smells* específicos y las distintas formas de resolverlos.

REFERENCIAS

- [1] M. Fowler. "CodeSmell", p. 1. 2016. Fecha de consulta: junio de 2019. URL: <https://martinfowler.com/bliki/CodeSmell.html>
- [2] R. Contreras Espinosa. "Juegos digitales y gamificación aplicados en el ámbito de la educación". RIED. Revista Iberoamericana de Educación a Distancia. Vol. 19 N° 2, pp. 27-33. 2016.
- [3] V. Romero Kutzner, M. Gómez y A. Ricarte-Sabater. "Juegos educativos y cooperativos: como mejorar el aprendizaje de la Biología y Geología en grupos disruptivos de 1° de la ESO". V Jornadas Iberoamericanas de Innovación Educativa en el Ámbito de las TIC y las TAC. Las Palmas, España. 2018.
- [4] M. Rojas, L. Jaimes y M. Valencia. "Efectividad, eficacia y eficiencia en equipos de trabajo". Espacios. Vol. 11. 2017.
- [5] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts and E. Gamma. "Refactoring: Improving the Design of Existing Code". Addison-Wesley Professional. 1999. ISBN: 0201485677.
- [6] A. Marín, Y. Trujillo y D. Buedo. "Apuntes para gestionar actividades de calidad en proyectos de desarrollo de software para disminuir los costos de corrección de defectos". Ingeniare. Revista chilena de ingeniería. Vol. 27 N° 2, pp. 319-327. 2019.
- [7] R.C. Martin. "Clean Code: A Handbook of Agile Software Craftsmanship". Pearson Education. 2008. ISBN: 0132350882.
- [8] M. Feathers. "Working Effectively with Legacy Code". Prentice Hall. 2004. ISBN: 0131177052.
- [9] A. Yamashita and L. Moonen. "Do Developers Care about Code Smells? An Exploratory Survey". 20th Working Conference on Reverse Engineering (WCRE). Koblenz, Alemania. 2013. ISBN: 978-1-4799-2931-3. DOI: 10.1109/WCRE.2013.6671299.
- [10] R. Kober and A. Tarca. "For Fun or Profit? An Evaluation of a Business Simulation Game". AAANZ 1999 Conference (Accounting Association of Australia & New Zealand). Melbourne, Australia. 1999. ISBN: 1328780.
- [11] A. Baker, E. Navarro and A. Van Der Hoek. "An Experimental Card Game for Teaching Software Engineering Processes". 16th Conference on Software Engineering Education and Training. Madrid, España. 2003. ISBN: 0-7695-1869-9. DOI: 10.1109/CSEE.2003.1191379.
- [12] K. Klassen and K. Willoughby. "In-Class Simulation Games: Assessing Student Learning". Journal of Information Technology Education. Vol. 2, pp. 1-13. 2000. ISSN: 1539-3585.
- [13] J.H. Andrews. "Killer App. A Eurogame about Software Quality". 26th International Conference on Software Engineering Education and Training (CSEE&T). San Francisco, USA. 2013. ISBN: 978-1-4673-5140-9. DOI: 10.1109/CSEET.2013.6595269.
- [14] N. Brown, P. Kruchten, E. Lim, R. Nord and I. Ozkaya. "Hard choice: A game for balancing strategy for agility". 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T). Honolulu, USA. 2011. DOI: 10.1109/CSEET.2011.5876149.
- [15] R. Atal and A. Sureka. "Anukarna: A Software Engineering Simulation Game for Teaching Practical Decision Making in Peer Code Review". Asia Pacific Software Engineering Conferences. New Delhi, India. Vol. 1519, pp. 63-70. 2015.
- [16] S. Morrison, S. Dighans, T. Daniels, C. Marmon and C. Izurieta. "Technical Debt Reduction Using a Game Theoretic Competitive Source Control App. oach". CUR Journal. Winter 2012. Vol. 33. 2012.
- [17] S. Arai, K. Sakamoto, H. Washizaki and Y. Fukazawa. "A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools". 6th International Workshop on Empirical Software Engineering in Practice (IWESEP). Osaka, Japón. 2014. ISBN: 978-1-4799-6666-0. DOI: 10.1109/IWESEP.2014.17.
- [18] C. Zapata Jaramillo, M. Gomez Álvarez and G. González Calderón. "Riskware: A Game for Teaching Software Project Risk Management". Developments in Business Simulation and Experiential Learning. Vol. 40. 2013.