09-032

USE OF CODE REVIEW IN SOFTWARE ENGINEERING TO CONTRIBUTE TO THE DEVELOPMENT OF SELF-REGULATED LEARNING

Puertas, Eloi (1)

(1) Universitat de Barcelona

Nowadays, the development of any software project is a team task. A common practice used by application development teams to improve the quality of their code and catch bugs quickly is Peer Code Review. The Peer Code Review consists once a piece of software has been completed, it is reviewed by other developers who offer feedback on the quality of its code, quality of the tests, possible errors... The use of Peer Code Review among peers in Software Engineering subjects contributes, apart from improving the quality of the code, to developing the competence of self-regulated learning through feedback that is carried out among peers. This communication details two teaching experiences carried out in two different subjects of the Computer Engineering degree at the University of Barcelona: "distributed software" and "software engineering".

Keywords: Peer Code Review; Software engineering; self-regulated learning

USO DE REVISIÓN DE CÓDIGO EN INGENIERÍA DEL SOFWARE PARA CONTRIBUIR AL DESARROLLO DEL APRENDIZAJE AUTOREGULADO

En la actualidad, el desarrollo de cualquier proyecto de sofware es una tarea de equipo. Una práctica que suelen usar los equipos de desarrollo de aplicaciones para mejorar la calidad de su código y detectar errores de forma rápida es el Peer Code Review (Revisión de Código entre iguales). El Peer Code Review consiste en que el desarrollador de una pieza de código, una vez ha acabado su implementación, es revisada por otros desarrolladores que le ofrecen feedback sobre la calidad de su código, calidad de las pruebas realizadas sobre el código, posibles errores…. El uso de Peer Code Review entre pares en asignaturas de desarrollo del Software contribuye, a parte de mejorar la calidad del código realizado, a desarrollar la competencia del aprendizaje autoregulado mediante el feedback que se realiza entre iguales. En esta comunicación se detallan dos experiencias docentes llevadas a cabo en dos asignaturas diferentes del grado de Ingeniera Informática de la Universidad de Barcelona: “software distribuido” e “ingeniería del software”.

Palabras clave: Revisión de Código; Ingenieria del Software; Aprendizaje autoregulado

Correspondencia: Eloi Puertas. Correo: epuertas@ub.edu

1. Introducción

Todavía hoy en día en las carreras de Ingeniería Informática hay un fuerte componente de programación en muchas de las asignaturas ya sean en cursos de formación básica como obligatorios u optativos. Tradicionalmente la tarea de programación en las asignaturas de las titulaciones universitarias se ha visto siempre como una actividad creativa y práctica que el alumno debía realizar con su dedicación y esfuerzo, echando mano de toda la documentación que tuviera a su alcance.

Sin embargo, desde hace ya tiempo las tareas de desarrollo de aplicaciones en el ámbito laboral se han convertido en un trabajo de equipo coordinado entre diferentes perfiles profesionales. Más aún con la instauración de metodologías ágiles (Scrum, Kanban, Lean) en muchas empresas, donde la responsabilidad individual y la comunicación entre los demás miembros del equipo son primordiales.

Es por este motivo que en los diferentes cursos universitarios de programación es importante que el alumno vaya adquiriendo competencias tanto de trabajo en equipo como de ser capaz de dar *feedback* de calidad a sus compañeros. Una forma de que adquieran estas competencias es usando herramientas que faciliten la metodologia de trabajo y que también se encontraran más adelante en el mundo laboral. Para ello, usaremos la plataforma Github (github 2022) que nos va a permitir que los alumnos puedan gestionar el código de su proyecto y colaborar entre ellos. Tal y como se ha venido demostrando en la literatura (Zimmerman 2001, Cano-García 2020) el uso de *feedback* entre iguales mejora la competencia del aprendizaje autorregulado, es decir, el alumno toma consciencia de que es lo que sabe y que le queda aún por aprender sobre un tema.

En esta comunicación se detallan dos experiencias llevadas a cabo en dos asignaturas diferentes del grado de Ingeniera Informática de la Universidad de Barcelona: “software distribuido” e “ingeniería del software”. En las siguientes secciones vamos a describir estas asignaturas, así como la metodología docente usada para que los alumnos adquieran estas competencias centrándonos en la aplicación del *peer review code* (Indriasari 2015).

2. Caso de estudio. Software Distribuido

La primera experiencia se ha llevado a cabo en la asignatura de Software Distribuido del grado de Ingeniería Informática de la Universidad de Barcelona. Se trata de una asignatura obligatoria de sexto semestre, y por lo tanto los alumnos ya han cursado varias asignaturas sobre diseño e implementación de proyectos software en los semestres anteriores. El número aproximado de alumnos que cursan la asignatura anualmente es de 75.

La carga es de 6 créditos donde la mitad son de trabajo práctico en el aula de informática donde se les pide que desarrollen dos aplicaciones distribuidas: la primera es una aplicación cliente-servidor con un protocolo de comunicación ad-hoc y la segunda es una aplicación web desarrollada sobre un servicio web *RESTful*. Estas aplicaciones se construyen desde cero y el estudiante ha de tomar las decisiones principales de diseño e implementación de la solución. Para cada proyecto se destina un mes y medio con 6 o 7 sesiones en el laboratorio con acceso

al profesorado. La última sesión se realiza siempre un testeo cruzado entre las diferentes soluciones realizadas por los alumnos.

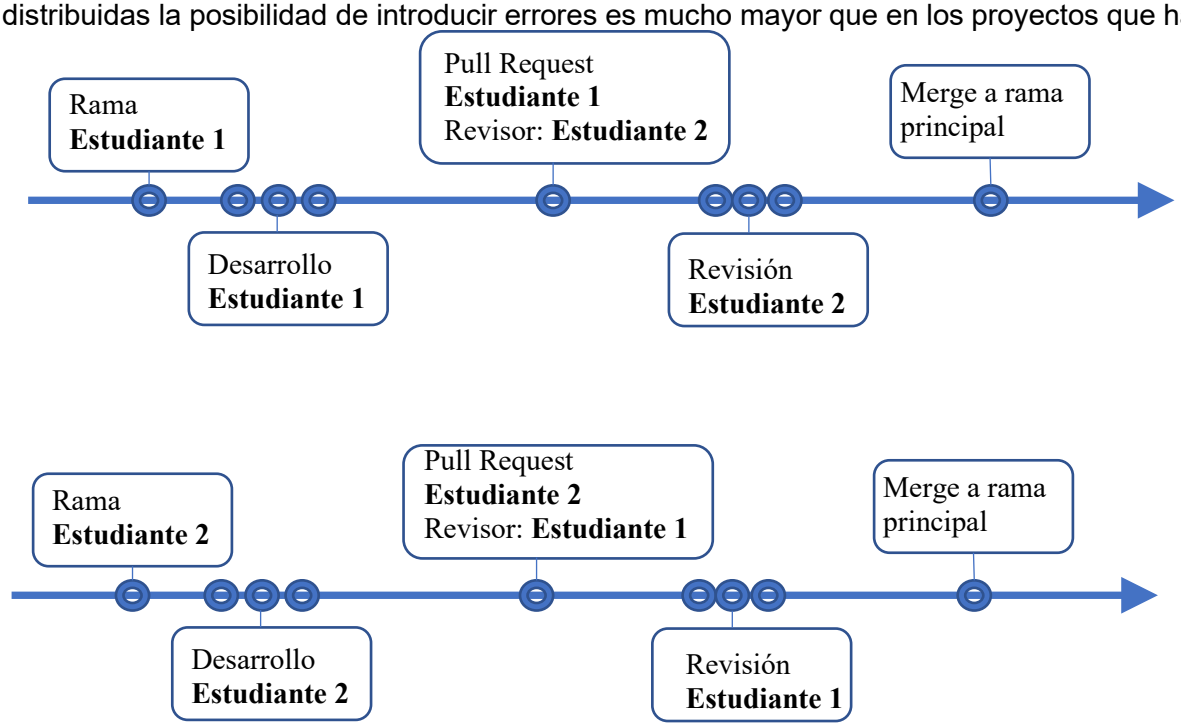
FIGURA 1 METODOLOGIA DE PROGRAMACIÓN CON PEER CODE REVIEW ENTRE PARES DE ESTUDIANTES.

Dentro de las diferentes competencias que se van a desarrollar se encuentra la capacidad de trabajo en equipo, es por ello que es importante que el trabajo práctico se realice en grupos. En este caso vamos a realizar las prácticas por parejas, para así poder poner en práctica metodologías ágiles de programación como el *pair programing* (Williams 2010) o *extreme programming* (Beck 2000) centrado en el *Test-Driven Development* (TDD) (Beck 2003)

Para el desarrollo de las aplicaciones los alumnos van a disponer de entornos de desarrollo (*IDEs*) adecuados al proyecto a desarrollar, así como un repositorio para cada equipo en Github. Para gestionar la creación y poder hacer luego el seguimiento de estos repositorios se ha usado la plataforma Github Classroom (github classroom 2020) que proporciona la aplicación Github al colectivo de profesores.

La metodologia de programación que van a seguir los alumnos será la siguiente:

Antes de nada, se les requiere que el proyecto lo deben desarrollar usando Test-Driven Programming, que ya han aprendido anteriormente en otras asignaturas. Para los proyectos que se van a implementar, la parte de testing es la más importante ya que en las aplicaciones distribuidas la posibilidad de introducir errores es mucho mayor que en los proyectos que han



desarrollado hasta ahora.

A continuación, se muestra a los alumnos cómo se debe hacer la colaboración entre los dos miembros del equipo. Por un lado, pueden desarrollar en *pair* o *extreme programming* cuando,

por ejemplo, están en clase o en remoto trabajando síncronamente sobre el mismo código. Para que quede constancia del *feedback* que se estén dando mutuamente, se les pide que hagan *peer code review* del código desarrollado por el otro miembro, como mínimo una vez por semana siguiendo las instrucciones que se les han dados previamente. En la figura 1 se puede ver el diagrama de la metodología usada. Primero les pedimos que cada alumno trabaje en una rama de código separada, desarrollando partes del proyecto independientes, por ejemplo, un alumno puede avanzar en la definición de unos mensajes del protocolo y el otro avanzar en el código del cliente. Cuando quieran integrar la solución de su rama a la rama principal lo harán mediante un *Pull Request* de github, especificando a su compañero como revisor del *Pull Request*. El revisor una vez recibe la notificación de que debe revisar el código que le han enviado, este debe comprobar que:

- el código en la rama del compañero funciona correctamente.
- pasa los tests de forma satisfactoria.
- sigue la guía de estilo del lenguaje de programació.
- sea entendible.
- esté bien documentado.

Si durante la revisión encuentra algún error o defecto se lo comenta en el mensaje del *Pull Request*. Se pide también que se ponga énfasis en la forma como este *feedback* se está dando, es importante ser asertivo y respetuoso con el trabajo realizado por el otro en los mensajes que se envíen. En la figura 2 se puede ver un ejemplo de *feedback* entre alumnos usando *pull request*.

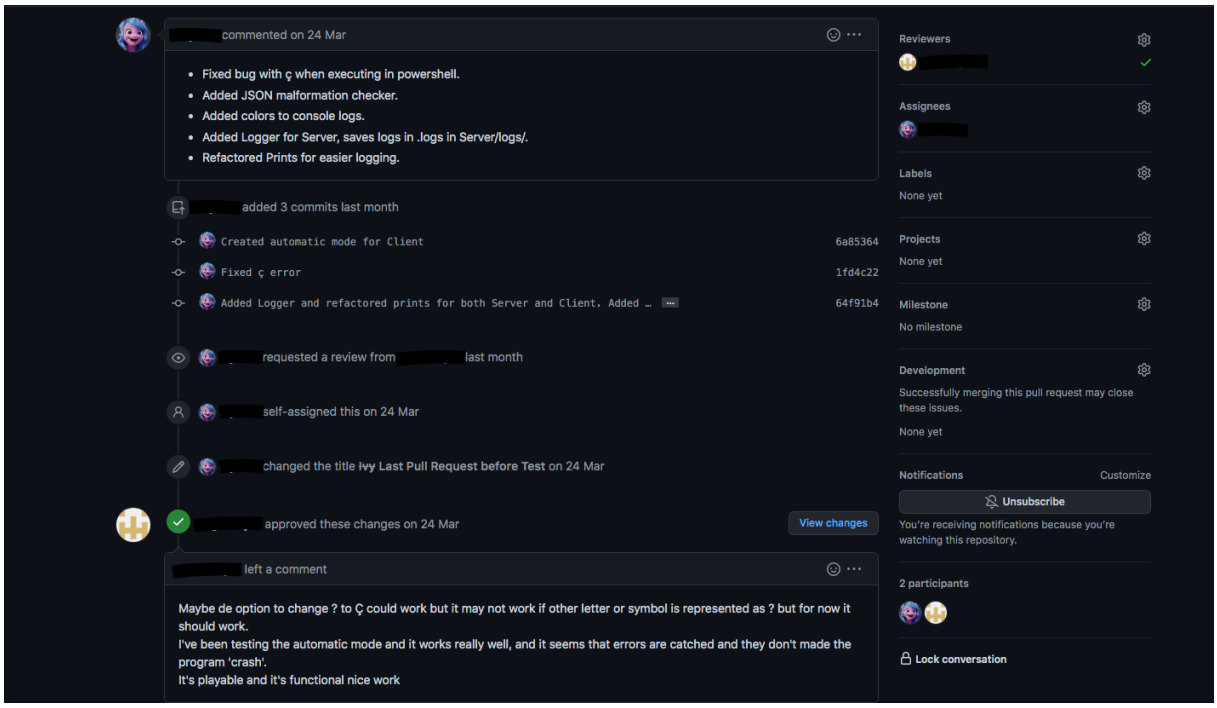


FIGURA 2 EJEMPLO DE PULL REQUEST ENTRE DOS ALUMNOS

Finalmente, antes de la última entrega se dedica la última sesión a hacer un test cruzado entre las aplicaciones de los diferentes grupos. Por ejemplo, en la primera práctica, se puede usar el cliente de un grupo con el servidor de otro, ya que el protocolo es el mismo para toda la clase. Este testeo cruzado también se desarrolla como un *peer code review* pero en este caso no se mira el código sino la funcionalidad de las aplicaciones de los demás contra la propia. Para recoger este *feedback* usamos una de las herramientas que vienen en el campus virtual de la asignatura (basado en Moodle) que se llama taller (Cox 2012) y permite hacer *peer evaluation* entre los alumnos de un curso.

Se ha comprobado que haciendo este trabajo de *peer reviewing* los alumnos son más conscientes del trabajo realizado por el otro y aprenden tanto del código que se revisa como del que desarrollan. Al provocar este *feedback* constante entre los alumnos también se fomenta que los dos alumnos participen de forma equitativa en la resolución de la práctica y esta no recaiga más en uno u otro miembro del equipo.

2. Caso de estudio. Ingeniería del Software

El segundo caso de estudio se trata de la asignatura de Ingeniería del Software, también del grado de Ingeniería Informática del grado de la Universidad de Barcelona. En este caso la asignatura tiene como objetivo el de dar a conocer a los estudiantes las metodologías de desarrollo ágiles de proyectos de software. Para ello se va a usar la metodologia SCRUM (Rubin 2012) para desarrollar una aplicación web desde cero. Ingeniería del software se cursa en el séptimo semestre, justo antes de acabar la carrera, con un número aproximado de 70 alumnos anuales. La carga es de 6 créditos donde la mitad de trabajo práctico en el aula de informática. Para reproducir la metodología SCRUM en el aula, dividimos los grupos de prácticas en equipos de entre 5 y 7 personas

Una vez establecidos los equipos, los alumnos empiezan el desarrollo de su aplicación usando la metodologia SCRUM tal y como se puede ver en la figura 3. Primero de todo, los alumnos

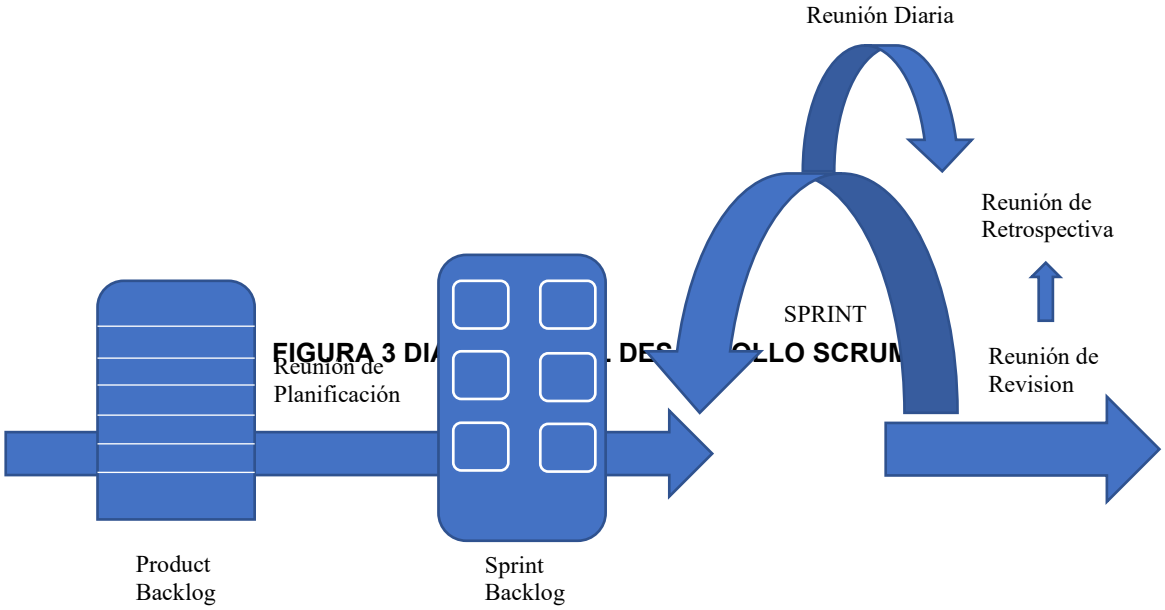


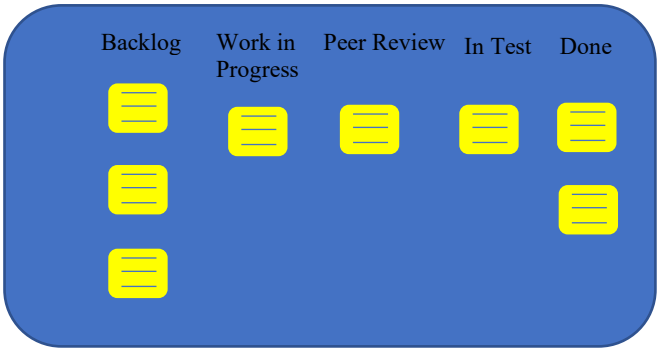
FIGURA 3 DIA[illegible] DES[illegible]OLLO SCRUM

empiezan a definir el *backlog* de su aplicación web, es decir, qué características va a tener su aplicación. Esto lo llevan a cabo mediante la definición de historias de usuario. Para que la carga de trabajo sea similar para todos los equipos el profesorado escoge los temas concretos de las diferentes aplicaciones web. Para escribir las historias de usuario se va usar la aplicación de gestión de proyectos Trello (trello 2022).

La metodología que se va a seguir dentro de cada sprint, que son de 15 días es la siguiente: Primero se hace la reunión de planificación donde los estudiantes priorizarán los elementos del *blacklog* y escogerán cuales van a implementar en el siguiente sprint. A partir de aquí, los alumnos van a transformar los elementos en tareas de código para ser realizadas, especificando las condiciones que se deban dar para que se dé la tarea como completada (a ser posible, que puedan ser comprobadas mediante test automatizados). Para poder organizar y visualizar su trabajo durante el sprint, los alumnos utilizaran la metodología Kanban (Kirovska 2015), se usará la herramienta de proyectos de Github para diseñar el tablero *kanban* y asignar las diferentes tareas (*issues* de github) a cada miembro del grupo.

En la figura 4 podemos ver un ejemplo de diseño de tablero *kanban*, con las líneas usadas comúnmente por los alumnos.

FIGURA 4 EJEMPLO DE TABLERO KANBAN



En este caso, la metodología docente que vamos a usar para practicar la competencia de trabajo en equipo es la siguiente: Cada miembro se asigna una tarea de las que hay en el *sprint backlog* en el tablero de *kanban*. A continuación, se asigna hasta dos compañeros que desempeñen tareas parecidas para que hagan el *peer code review* de la tarea cuando ésta esté realizada. En esta revisión, los compañeros deberán comprobar el funcionamiento del código mediante los tests desarrollados en la tarea, así como comprobar que la tarea se pueda dar por finalizada correctamente. Durante el *sprint*, se realizarán las reuniones diarias, que son reuniones para sincronizar el trabajo del grupo, para así tener información de cuando se ha terminado una tarea y por lo tanto se puede hacer ya su revisión o si se puede empezar a desarrollar una tarea que estaba pendiente de que otra finalizara. Estas reuniones no deberían durar más de 10 minutos y aunque lo ideal es que se realizaran presencialmente, como los alumnos pueden no coincidir a diario, se realiza mediante la herramienta de mensajería Slack (slack 2022). Los alumnos acuerdan una hora límite en la cual deben de responder 3 preguntas básicas sobre su progreso: ¿Qué hiciste ayer?, ¿Qué harás hoy?, ¿Te has encontrado con algun impedimento? Hay que tener en cuenta que los alumnos no tienen porqué trabajar en el proyecto a diario, por lo tanto, puede ser normal que no hayan avanzado, pero no por ello se hayan encontrado con ningún impedimento.

Al finalizar el sprint, todo el equipo realiza la reunión de revisión en clase con el profesor. En esta reunión se comprueba los avances realizados en el sprint mediante una pequeña *demo*. También se aprovecha para hacer la reunión de retrospectiva, donde los alumnos discuten sobre el *feedback* recibido por el profesor en la revisión y se responden las siguientes cuestiones: ¿Qué ha funcionado bien durante este sprint?, ¿Qué podria haber ido mejor?, ¿Que mejoras se pueden proponer para el siguiente Sprint? Finalmente se prepara el siguiente sprint realizando la reunión de planificación y así poder empezar con las nuevas tareas al día siguiente.

4. Conclusiones

Se han presentado dos experiencias llevadas a cabo dentro del grado de Ingeniería Informàtica de la Universidad de Barcelona. En la experiencia realizada en la asignatura de Software Distribuido se ha focalizado en el trabajo entre pares, mejorando así el aprendizaje

autorregulado y el realizar revisiones de código dando *feedback* de calidad. Por otro lado, la experiencia realizada en la asignatura de Ingeniería del Software se ha trasladado al aula la forma de trabajo realizada en las metodologías Scrum y Kanban, adaptándolas al entorno docente y permitiendo a los alumnos un trabajo en equipo eficiente y colaborativo.

Así pues, la metodologia docente usada en estas asignaturas permite que los estudiantes puedan adquirir las competencias de trabajo en equipo, aprendizaje autorregulado y comunicación. Además, esta metodología se puede exportar a cualquier otra asignatura de programación, ya sea de cursos básicos o avanzados, ya que las herramientas usadas son ampliamente usadas por las empresas y todas ellas o bien disponen de licencias para estudiantes o son de código abierto.

5. Referencias

Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesle professional.

Beck, K., (2003). *Test-driven development*. Boston: Addison-Wesley.

Cano-García, E., & Pons-Seguí, L. (2020). The Promotion of Self-Regulated Learning Through Peer Feedback in Initial Teacher Education. *International Journal of Technology-Enabled Student Support Services (IJTESSS), 10*(1), 1-20. http://doi.org/10.4018/IJTESSS.2020010101

Cox, J., Posada, J. & Waldron, R. (2012). Moodle Workshop activities support peer review Year 1 Science: present and future. 10.13140/2.1.3232.2564.

github. (2022). *GitHub*. Retrieved May 26, 2022, from https://github.com/

github classroom. (2022). *GitHub Classroom*. Retrieved May 26, 2022, from https://classroom.github.com/

Indriasari, T.B, Luxton-Reilly, A. & Denny, P. (2020). *A Review of Peer Code Review in Higher Education*. ACM Trans. Comput. Educ. 20, 3, Article 22 (September 2020), 25 pages. DOI:https://doi.org/10.1145/3403935

Kirovska, N., & Koceski, S. (2015). Usage of Kanban methodology at software development teams. *Journal of applied economics and business*, *3*(3), 25-34.

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley.

slack. (2022). *GitHub*. Retrieved May 26, 2022, from https://slack.com/

trello. (2022). *GitHub*. Retrieved May 26, 2022, from https://trello.com/

Williams, L. A. (2010). *Pair Programming*. Encyclopedia of software engineering, *2*.

Zimmerman, B. & Schunk, D. (2001). *Self-regulated learning and academic achievement: Theoretical perspectives.* Mahwah, NJ: Lawrence Erlbaum Associates.

Comunicación alineada con los Objetivos de Desarrollo Sostenible