Fecha de recepción 08/09/2021
Fecha de aceptación: 04/04 /2022
Pp. 101- Pp.115

Enseñanza de los Sistemas Operativos mediante la programación en proyectos de código abierto: una experiencia pedagógica

Carlos Gómez
cgomezch@ucab.edu.ve
Universidad Católica Andrés Bello, Caracas, Venezuela

Resumen

En este artículo se narra una experiencia pedagógica desarrollada en el curso de Sistemas Operativos, donde se asignó a los estudiantes una actividad de investigación que consistía en contribuir a un proyecto de software libre o código abierto. Esta actividad se diseñó como una aplicación del método instruccional de aprendizaje por proyectos. Se esperaba que los estudiantes escogieran un proyecto de software de código abierto, eligieran una tarea, hicieran los cambios necesarios y los enviaran mediante un *pull request* al repositorio. A pesar de que el número de estudiantes en esta experiencia fue pequeño, dos equipos realizaron exitosamente los *pull requests*. Se ofrecen una apreciación de la experiencia y recomendaciones para instalaciones futuras de esta actividad.

Palabras clave: código abierto, software libre, sistemas operativos, enseñanza, educación

Teaching Operating Systems through programming in open source projects: a pedagogical experience

Abstract

This article narrates a pedagogical experience developed in the Operating Systems course, where students were assigned a research activity that consisted in contributing to a free or open source software project. This activity was designed as an application of the project-based learning instructional method. Students were expected to pick an open source software project, choose an activity, perform the required changes, and submit a pull request to the repository. Although the number of students in this experience was small, two teams successfully performed the pull requests. An appreciation of the experience and recommendations for future installations of this activity are offered.

Keywords: Free and open source software, operating systems, teaching, education

Fecha de recepción 08/09/2021
Fecha de aceptación: 04/04 /2022
Pp. 101- Pp.115

Ensino de Sistemas Operacionais por meio de programação em projetos open source: uma experiência pedagógica

Resumo

Este artigo narra uma experiência pedagógica desenvolvida no curso de Sistemas Operacionais, onde os alunos realizaram uma atividade de pesquisa que consistia em contribuir para um projeto de software livre ou open source. Esta atividade foi concebida como uma aplicação do método instrucional de aprendizagem por projetos. Esperava-se que os alunos escolhessem um projeto de software de código aberto, escolhessem uma tarefa, fizessem as alterações necessárias e as enviassem por meio de uma solicitação pull ao repositório. Embora o número de alunos nessa experiência tenha sido pequeno, duas equipes concluíram com êxito as solicitações de pull. Uma apreciação da experiência e recomendações para futuras instalações desta atividade são oferecidas.

Palavras-chave: código aberto, software livre, sistemas operacionais, ensino, educação

i. INTRODUCCIÓN

El presente artículo narra una experiencia académica desarrollada dentro de la asignatura de Sistemas de Operación de la Universidad Católica Andrés Bello (UCAB), en la cual se permitió a los estudiantes realizar un "proyecto abierto", que consistía en seleccionar un proyecto existente de código abierto (por ejemplo algún proyecto disponible en *GitHub*) y contribuir al repositorio de dicho proyecto, agregando alguna funcionalidad al software, mejorando el código fuente o arreglando *bugs*. Según el conocimiento que se tiene, ésta es una experiencia pedagógica que se realiza por primera vez en la carrera de Ingeniería Informática de la UCAB **[1]**.

Esta experiencia se diseñó como una aplicación del método instruccional de aprendizaje por proyectos, dentro del cual se espera que los estudiantes desarrollen proyectos de su interés, motivados por iniciativa propia y donde los proyectos deberían
tener alguna relevancia para el entorno o la comunidad del estudiante. Según este método, se espera que los proyectos creen las condiciones necesarias para que los estudiantes aprendan los contenidos y las herramientas relevantes, pero mediante un proceso más auténtico que se da a través de la práctica y no una instrucción guiada y dominada por el docente.

La unidad curricular de Sistemas de Operación está enmarcada dentro del área de conocimiento denominada Organización y arquitectura del computador, según las áreas de conocimiento definidas en las recomendaciones curriculares de Ingeniería de Computación de la ACM **[2]**. Según este currículo, las competencias de conocimiento de esta área son la medición del rendimiento de las computadoras, la organización de los procesadores, los sistemas de memoria, las tecnologías de entrada y salida, las arquitecturas multinúcleo y las arquitecturas de los sistemas distribuidos, entre otros.

Específicamente dentro de la UCAB, la asignatura de Sistemas de Operación es una materia obligatoria de nivel intermedio que forma parte del programa de Ingeniería Informática, ubicada en el quinto semestre y

dictada mediante dos horas de clases teóricas a la semana y dos horas de práctica.

De acuerdo con las recomendaciones de la ACM, la enseñanza de la Ingeniería de Computación debe incluir la preparación de los estudiantes para la práctica profesional. Dentro de las características de la disciplina de Ingeniería de Computación, la ACM establece que:

A diferencia de profesiones como el Derecho o la Medicina, la Ingeniería no suele requerir un título avanzado para trabajar en el campo. Por ello, los programas de licenciatura en Ingeniería Informática deben incluir no sólo conocimientos básicos dentro del campo, sino la capacidad de aplicarlos a la solución de proyectos realistas. Específicamente, el contexto social de la Ingeniería debe integrarse en la enseñanza del Diseño de Ingeniería, incluyendo el uso de las mejores prácticas y el equilibrio entre los requisitos técnicos, fiscales y sociales. Además de la profesionalidad, la preparación adecuada abarca elementos tanto técnicos (capacidad de diseño, experiencias de laboratorio, uso de herramientas de Ingeniería) como no técnicos (trabajo en equipo, comunicación). **[2]**

Aunque estas características hacen referencia a la carrera de Ingeniería de Computación en general y no a una asignatura específica, el programa de Sistemas Operativos de la UCAB claramente especifica que la comunicación, el trabajo en equipo, la capacidad de investigar, la habilidad de abstraer y analizar, y el manejo adecuado de las tecnologías son competencias de la asignatura. El cuadro 1 presenta las competencias generales y específicas dentro del programa de Sistemas de Operación de la UCAB **[3]**. Además, la justificación de la asignatura enuncia que dicha unidad curricular "contribuye con el desarrollo de las competencias generales: aprender a aprender con calidad y aprender a trabajar con el otro, en particular en lo que respecta al desarrollo de la capacidad de análisis, abstracción, la formulación y resolución eficaz de problemas, ya sea en forma individual o en equipo".

Tabla I. Competencias de la asignatura Sistemas de Operación, Ingeniería Informática UCAB.

Competencia general 1: Aprender a aprender con calidad
Unidades de competencia
Abstrae, analiza y sintetiza información
Se comunica eficazmente de forma oral y escrita
Realiza investigaciones
Trabaja en forma autónoma
Competencia general 2: Aprender a trabajar con el otro
Unidades de competencia
Participa y trabaja en equipo
Competencia general 3: Aprender a interactuar en el contexto global
Unidades de competencia
Maneja adecuadamente las tecnologías de información y comunicación

El presente artículo se estructura de la siguiente forma. En primer lugar, se describe el método instruccional de aprendizaje por proyectos y su relevancia en el contexto de la carrera de Ingeniería de Computación. Seguidamente, se explica en qué consiste el software de código abierto y qué implica contribuir a un proyecto de esta clase. Luego se presentan los antecedentes, se explica el diseño de la experiencia pedagógica, se exponen los proyectos realizados por los alumnos, se discuten los resultados de la experiencia y se presentan las conclusiones y recomendaciones.

ii. APRENDIZAJE POR PROYECTOS

El aprendizaje por proyectos es un método instruccional que se centra en el desarrollo de un proyecto que los estudiantes no solamente elaboran, sino que tienen un grado considerable de liderazgo en dicho proyecto. El primer paso en las etapas de preparación del proyecto es la elección del tema, el cual no debe ser impuesto por el docente, sino que debe surgir de los intereses propios de los estudiantes, de su curiosidad, de sus aspiraciones y del mundo (su mundo) **[4]**. Para facilitar la elección del tema, el docente puede realizar una serie de actividades motivadoras, tales como clases, lecturas, videos, preguntas de investigación, charlas de expertos, salidas de campo, entre otras. O bien, el docente puede proponer una serie de temas a escoger, pero estando siempre abierto a las inquietudes, intereses y sensibilidades de los estudiantes, sin excluir que los estudiantes puedan elegir su propio tema.

El proyecto es una actividad protagonizada por los estudiantes, donde idealmente pueden ellos mismos formar los equipos, definir los objetivos, elaborar la planificación, seguir el avance y asignar los roles dentro del equipo. No se trata de darle total libertad a los estudiantes, sino que el docente es como un tutor que guía a los estudiantes e interviene cuando es necesario aportar algún conocimiento o asistir en la gerencia del proyecto.

El proyecto no es una actividad meramente académica, aislada del medio social del estudiante. Al contrario, se espera que el proyecto tenga algún impacto en el entorno, en la familia o en la comunidad del estudiante. De hecho, el proyecto no termina con el curso o el año académico, sino que debería producir un resultado tangible que la comunidad pueda aprovechar y que los estudiantes del próximo año puedan seguir desarrollando. Así, el aprendizaje por proyectos no es una experiencia aislada, sino que funciona mejor si se implementa en una institución de forma continua, de modo que las nuevas cohortes de estudiantes puedan seguir aportando al proyecto y así producir un impacto con varios años de alcance.

De acuerdo con Lacueva **[4]**, la propuesta de trabajar por proyectos de investigación ha formado parte de la educación desde hace más de un siglo.

La historia de los proyectos en educación puede remontarse al menos a la Italia de finales del siglo XVI, donde los estudiantes avanzados de arquitectura debían presentar "progetti" como parte de su entrenamiento: propuestas de edificación o construcción. ***[5,6]*** *A lo largo del siglo XVII y principios del XVIII la iniciativa se extendió primero a Francia y luego a otros países europeos, cubriendo no sólo el campo de la arquitectura sino también el de la ingeniería, y ganando en sistematicidad. (…) Para la segunda mitad del siglo XIX, el proyecto como trabajo libre de aplicación al final de un curso o de una carrera había arribado ya a Estados Unidos. (…) Incluso, algunos lo consideraron no una actividad terminal, sino una iniciativa presente desde el inicio de los estudios, dando sentido al aprendizaje de las diversas destrezas.*

El aprendizaje por proyectos es una técnica aplicable a la formación de ingenieros en computación en nuestros días. Las recomendaciones curriculares de la ACM para dicha carrera hacen énfasis en que se debe formar a ingenieros que puedan asumir directamente el ejercicio profesional al graduarse, y especifica no sólo las competencias técnicas, sino las competencias complementarias a desarrollar, entre ellas la comunicación, el trabajo en equipo y el aprendizaje continuo. Dicho currículo también menciona la posibilidad de aplicar laboratorios abiertos a lo largo de la carrera:

El plan de estudios de Ingeniería de Computación suele contener experiencias abiertas en las que tienen lugar la verdadera investigación y desarrollo. Se podría considerar como la "máxima experiencia de laboratorio". Las experiencias de diseño concluyentes o al final de la carrera suelen encarnar este aspecto abierto. En estas

situaciones, un docente y un equipo de estudiantes deciden un área de exploración y, una vez decidida, el equipo de estudiantes comienza el proceso de investigación y de diseño. Los programas de estudio suelen proporcionar un espacio dedicado donde los equipos pueden reunirse y trabajar. Estos espacios suelen contar con instalaciones modernas y ofrecer espacio suficiente para dispositivos electrónicos (por ejemplo, robots) y otros equipos necesarios para el proyecto en cuestión. **[2]**

iii. PROYECTOS DE CÓDIGO ABIERTO

En castellano, el término software de código abierto se suele utilizar intercambiablemente con el término software libre. En realidad, existe una diferencia "filosófica" entre los dos conceptos dado que software libre se refiere más específicamente a programas disponibles bajo la Licencia Pública General GNU (GNU GPL) u otras licencias análogas promovidas por la *Free Software Foundation*. El software de código abierto se refiere de forma más general a programas que son desarrollados de manera abierta y colaborativa, de modo que cualquier persona pueda acceder al código fuente del software y modificarlo. En otras palabras, los programas disponibles bajo la GPL son de código abierto, pero también lo son aquellos distribuidos con la licencia MIT, Apache, Licencia Pública de Mozilla, BSD, entre otras.

Más allá de las diferencias legales entre los distintos tipos de licencia, el software de código abierto se basa en el principio de compartir públicamente el código y construirlo de forma colaborativa, lo cual no sería posible sin los sistemas de control de versiones, que comenzaron a ser desarrollados en la década de 1970. Los sistemas de control de versiones permiten llevar registro de los cambios hechos al código fuente, de manera que un programador pueda volver a una revisión anterior del código si así lo requiere. El sistema de control de versiones distribuido más utilizado hoy en día es *Git*, el cual fue desarrollado en 2005 por Linus Torvalds y otros desarrolladores del kernel de Linux.

En *Git*, los archivos de código fuente se administran dentro de un repositorio, que es una colección de archivos generalmente pertenecientes a un mismo proyecto, y además contiene la historia completa de estos archivos que permite volver a una versión anterior. Cuando un programador realiza una serie de cambios al código fuente que quiere preservar y/o compartir con los demás desarrolladores, realiza un *commit*, que consiste en una entrada en el historial de los archivos y expresa los cambios que se hicieron a cada línea de código. Existen casos en que un programador quiere hacer un commit pero no desea modificar el código existente en el repositorio sino crear una versión alterna. En este caso, el programador crea una nueva rama, que puede verse como una línea dentro del historial de versiones.

Dentro de un repositorio, no todos los programadores tienen acceso a modificar el código fuente, algunos podrían tener acceso de sólo lectura. Si uno de estos programadores quiere realizar cambios en el código, debe hacer un *pull request*, que es básicamente una rama que se envía para ser integrada en el repositorio y queda a la espera de ser aprobada por los desarrolladores que tienen este privilegio. Estos desarrolladores pueden decidir aceptar el *pull request*, rechazarlo o responder con comentarios generales o anotaciones sobre cada línea de código que se debe mejorar.

Existen muchos otros factores a considerar al momento de contribuir a un proyecto de código abierto. Por ejemplo, cada proyecto suele tener una serie de convenciones de código, estilos de programación, etc. que se deben seguir al realizar una contribución. Además suele haber aspectos legales, por ejemplo, algunos proyectos requieren que se indique explícitamente el *copyright* del código que se envía, o solicitan a los contribuyentes que

firmen un acuerdo legal para evitar cualquier conflicto relacionado con derechos de autor.

Los sistemas de control distribuido de versiones suelen usarse en conjunto con sistemas de seguimiento de errores o *bug tracking*. Estos sistemas permiten llevar registro de los *bugs* presentes dentro de un software, aunque en realidad su uso se ha ampliado para registrar no sólo los bugs sino también los features o características pendientes por implementar dentro de un proyecto. Es posible abrir *bugs* nuevos cuando han sido reportados, cerrarlos cuando han sido resueltos, y llevar registro de la prioridad, los casos de prueba y los comentarios en torno a cada *bug*. Los sistemas de *bug tracking* suelen estar integrados o al menos vinculados con los sistemas de control de versiones, de forma que es posible, por ejemplo, decir que el *pull request* #38 resuelve el *bug* relacionado con la barra de búsqueda.

En los inicios de la programación durante las décadas de 1950 y 1960, el concepto de software propietario no había sido desarrollado y por lo tanto todo el software era en efecto software libre y de código abierto. La mayoría de los programas eran desarrollados en colaboración entre instituciones académicas e investigadores pertenecientes al sector privado, por lo tanto, eran compartidos bajo el principio de colaboración abierta que prevalecía en la academia. Además, generalmente los programas tenían que ser recompilados y modificados para poder ejecutarse en nuevo hardware, por lo tanto, era común distribuir el código fuente junto con el programa en lenguaje de máquina.

A finales de la década de 1970 y principios de 1980, un número creciente de empresas dedicadas exclusivamente al desarrollo de software empezó a comercializar licencias de software. Esto fue consecuencia natural de la complejidad creciente del software. Además, estas empresas competían con las empresas de hardware, quienes desarrollaban software para ser incluido en sus máquinas. Por otra parte, en 1974 una comisión del Congreso de los Estados Unidos decidió que el software podía ser protegido por derechos de autor. El movimiento del software libre comenzó en la década de 1980 como una reacción ante estas licencias de software propietarias que estaban surgiendo.

En 1983 Richard Stallman comenzó el desarrollo del sistema GNU, un sistema operativo abierto compatible con UNIX, y en 1985 fue creada la *Free Software Foundation* (FSF), la cual en 1988 publicó la primera versión de la Licencia Pública General GNU (GNU GPL) **[7]**. Esta licencia plantea la libre distribución de código como un derecho pero también como una obligación, es decir, todo cambio realizado a código GPL tiene que ser distribuido bajo la misma licencia.

Algunos autores consideran que la licencia GPL es muy restrictiva porque es poco compatible con la explotación comercial del software y por lo tanto perjudica la innovación en este sector **[8]**. La organización OSI (*Open Source Initiative*) fue creada en 1998 para promover licencias de código abierto más compatibles con el entorno del software comercial **[9]**. En general, existen dos tipos de licencias de código abierto:

1. Licencias *copyleft*, como la GNU GPL, las cuales requieren que las modificaciones al código fuente sean distribuidas bajo los mismos términos.
2. Licencias permisivas, como la licencia MIT, BSD o Apache, que resguardan los derechos de autor pero permiten que las modificaciones al código sean distribuidas bajo otra licencia **[7]**.

La Licencia Pública de Mozilla (*Mozilla Public License* o MPL) se considera una opción intermedia entre estos dos tipos de licencias.

iv. ANTECEDENTES

Varios estudios han investigado la enseñanza de la Informática a partir de la participación en proyectos de código abierto.

Sowe y Stamelos **[10]** desarrollaron una experiencia donde estudiantes de quinto semestre podían realizar *testing* dentro de proyectos de código abierto, y esta actividad representaba uno de los proyectos a ser evaluados durante el curso. Para este tipo de experiencia desarrollaron un *framework* de tres fases: primero planificar las actividades de clase y seleccionar los proyectos, segundo involucrar a los estudiantes en actividades del proyecto y tercero evaluar y puntuar la participación de los estudiantes. Diseñaron varias recomendaciones para seleccionar los proyectos, entre las cuales están elegir proyectos con más de dos desarrolladores, escoger proyectos donde el lenguaje de programación le resulta cómodo al estudiante y seleccionar proyectos con listas y foros de discusión activos.

Kussmaul **[11]** propuso un modelo de cinco pasos para integrar el software de código abierto en la formación de estudiantes universitarios en Ciencias de la Computación e Ingeniería de Software. El primer paso es utilizar proyectos de código abierto, el segundo es estudiar el proyecto como un ejemplo funcional, el tercer paso es añadir pequeñas mejoras al proyecto, el cuarto paso es construir componentes más complejos y el quinto paso es aprovechar el proyecto de código abierto para otros propósitos. El autor aplica esta estrategia en asignaturas introductorias (Computación 1 y 2), así como en asignaturas avanzadas y en el proyecto de grado de la carrera.

Ellis *et al.* **[12]** ofrecen una serie de consejos para involucrar a los estudiantes en proyectos de código abierto e identifican cinco fases de este proceso, las cuales son (1) entender las diferencias entre el software de código abierto y la cultura académica; (2) identificar un proyecto y los tutores; (3) sincronizar el ambiente académico con aquel del software de código abierto; (4) la participación estudiantil; y (5) concluir el proyecto haciendo el correcto cierre de la actividad.

Papadopoulos *et al.* **[13]** realizaron un estudio a lo largo de cuatro años de involucrar a los estudiantes de dos asignaturas del área de Ingeniería de Software en proyectos de código abierto. La participación en estos proyectos es optativa y cada estudiante adopta un rol específico que puede ser ingeniero de requerimientos, *tester*, desarrollador o diseñador/analista. También aplicaron al final de cada actividad un cuestionario a los estudiantes con preguntas abiertas y cerradas para medir la calidad de la experiencia. En general, los estudiantes de todos los cuatro grupos tuvieron una opinión muy positiva sobre la actividad, además, más de la mitad de los estudiantes involucrados se comunicaron directamente con los encargados del proyecto.

Smith *et al.* **[14]** desarrollaron un método sistemático para seleccionar proyectos de código abierto aptos para la enseñanza de la Ingeniería de Software. Los autores determinaron varios criterios para elegir los proyectos, entre ellos el tamaño del código, el lenguaje de programación, el dominio de aplicación, la modularidad del diseño, la presencia de actividad reciente, la calidad de la documentación y la facilidad de compilación. Se dedicó un verano completo de 12 horas al día para encontrar proyectos que cumplieran con los criterios seleccionados y se eligieron 16 proyectos, los cuales fueron satisfactorios para los estudiantes. Los autores concluyen que el proceso de seleccionar los proyectos requiere de una cantidad abrumadora de tiempo.

Nascimiento *et al.* **[15]** realizaron un estudio de casos de tipo mixto para determinar si los estudiantes de Ingeniería de Software perciben su contacto con los proyectos de código abierto como una experiencia del mundo real. El estudio incluye tres casos

donde los alumnos debían participar en proyectos de código abierto introduciendo modificaciones al código, realizando *testing* y documentando los requerimientos de software, respectivamente. Los estudiantes perciben que trabajar en proyectos de código abierto tiene cercanía con el trabajo en la industria, dado que se le da continuidad a proyectos existentes, hay que manejar software carente de documentación y se trata de una aplicación real del contexto estudiado. Identificaron algunas dificultades al enfrentarse a un proyecto de código abierto, como el que puede ser intimidante al inicio, dificultades con los proyectos extensos y dificultad de entender el código desarrollado por terceros.

v. EL PROYECTO ABIERTO

En el curso de Sistemas de Operación se indicó a los estudiantes que podían elaborar un proyecto abierto como parte de la evaluación, realizando este proyecto en lugar del proyecto que se asigna habitualmente, el cual plantea resolver un problema usando programación concurrente de UNIX en C. Dado que se trataba de una experiencia piloto, la participación en el proyecto abierto era optativa. Al inicio del curso, ocho estudiantes de treinta optaron por el proyecto abierto, y se distribuyeron en cuatro equipos de trabajo independientes. Aunque el número de participantes en esta experiencia no fue muy amplio, se consideró suficiente como una primera experiencia de la cual se pueden obtener aprendizajes para el futuro.

El proyecto abierto se dividió en las siguientes etapas:

1. Escoger el proyecto. Los estudiantes debían escoger el proyecto de código abierto al cual querían contribuir. Se sugirieron cuatro proyectos a los alumnos (*Mozilla*, *GNOME*, *LibreOffice* y *GIMP*) y se les indicó que podían elegir cualquier otro proyecto de su interés.
2. Escoger la actividad. Los estudiantes debían entregar un informe que especificara el proyecto y los issues o *bugs* específicos que iban a resolver. Seguidamente, los alumnos debían comunicarse con los encargados del proyecto para indicarles su interés en resolver estos issues.
3. Descargar el código fuente del proyecto. Esto incluye también compilar el código, lo cual implica instalar las dependencias y otras herramientas como el editor integrado de código (IDE).
4. Resolver el *issue*. Esto abarca identificar las partes del código que se deben modificar, aprender sobre los conceptos, métodos o tecnologías relacionadas, entender varias clases y funciones del programa, realizar los cambios y hacer las pruebas.
5. Hacer el *pull request* al repositorio. Los estudiantes debían enviar el *pull request*, responder a las observaciones, hacer las correcciones y mejoras necesarias y repetir este proceso tantas veces como fuera necesario, hasta que el código fuera finalmente aprobado.

De los cuatro equipos conformados inicialmente, tres pudieron llevar a cabo todas las actividades del proyecto abierto. A continuación se describe el trabajo realizado por cada uno de estos equipos.

A. Proyecto Nathaly-Raúl

Este equipo comenzó resolviendo un issue de la aplicación *GNOME To do*, un programa de lista de tareas para Linux. Pasaron varias semanas tratando de compilar el código, pero dado que la documentación estaba incompleta y no recibieron respuesta de los desarrolladores, no lograron compilarlo.

El docente les sugirió que trabajaran en un issue de *Leafpic*, una galería de fotos de código abierto para Android. El issue consistía

en implementar un cuadro de diálogo que permitiera reducir la resolución de la foto antes de compartirla.

Este proyecto resultó más fácil de compilar, además que existía buena comunicación con los encargados del proyecto. Gracias a esto y después de varias semanas de trabajo, el equipo logró implementar la funcionalidad exitosamente.

B. Proyecto Jose-Luis

Este equipo comenzó investigando sobre una biblioteca de *Twitch*, un servicio de videos en vivo de la empresa Amazon. Los desarrolladores estaban pasando a la siguiente versión de la biblioteca y el API no era estable, por lo cual no fue posible trabajar en este proyecto. El equipo también consideró un issue del videojuego "2048", un popular juego de lógica disponible en Android, pero el reporte del issue había sido hecho por un usuario y no estaba bien especificado.

Finalmente el equipo decidió resolver varios issues de la app para estudiantes del IIT Patna (Instituto Hindú de Tecnología, la India). Primero resolvieron dos issues muy sencillos que consistían en arreglar unas pocas líneas de código. Luego asumieron un issue de dificultad mediana que consistía en importar *Dagger* 2, una herramienta de *Google* para la inyección de dependencias.

Se mantuvo una buena comunicación con el encargado de la aplicación y los *pull requests* fueron aceptados. Cuando estaba finalizando la actividad, el encargado de la aplicación fue admitido en un proyecto de *Google Summer of Code* que le exigía dedicación exclusiva y por lo tanto le prohibía trabajar en otros proyectos, incluyendo el de IIT Patna. Como consecuencia, el repositorio de la app de IIT Patna paso de público a privado y esto impidió que el equipo siguiera contribuyendo a este proyecto.

C. Proyecto Abel-Ramón-Daniel

Este equipo asumió un bug en *Mozilla Firefox* presente en la ventana de marcadores. Cuando hay un marcador seleccionado y se hace doble clic en un área vacía de la lista, no debería suceder nada. Sin embargo, lo que sucede es que se abre el marcador seleccionado. Este bug fue reportado hace más de siete años y no ha sido resuelto.

Los estudiantes se comunicaron con el equipo de *Mozilla* a través de *Bugzilla* y recibieron ayuda en el proceso de compilar y empezar a trabajar con el código. Implementaron una solución al problema y subieron un *pull request*, sin embargo, les respondieron que la solución enviada no resolvía del todo el problema y no estaba implementada de la manera adecuada. Adicionalmente el equipo de *Mozilla* envió varias orientaciones sobre los cambios necesarios en el código, sin embargo, hasta la fecha el equipo no ha resuelto el bug.

Al final del proyecto abierto, los equipos realizaron una exposición en clase donde presentaron su trabajo y los aprendizajes obtenidos, las cuales fueron grabadas con el fin de recaudar información para esta investigación. También se realizó una encuesta escrita a los estudiantes que participaron en el proyecto abierto, con preguntas de selección y preguntas abiertas. Se indicó a los alumnos que contestaran la encuesta de forma individual y que la encuesta era anónima. Se realizó también una entrevista a cada equipo donde se discutía el trabajo realizado en cada una de las fases del proyecto así como los aprendizajes obtenidos. Igualmente se recaudó información en la correspondencia que se mantuvo con los estudiantes, en las conversaciones públicas y privadas que ellos tuvieron con los encargados del proyecto, en la discusión de los *pull requests* y en el diario del docente.

vi. DISCUSIÓN

A lo largo de la experiencia se identificaron varias dificultades, problemas comunes que tuvieron los equipos y situaciones que surgen en este tipo de experiencia académica. A continuación se discuten estos temas.

A. Elegir el proyecto

El proyecto abierto es una oportunidad para que los estudiantes trabajen en proyectos de código abierto de su interés, en aplicaciones o juegos que conozcan, en problemas relacionados con sus aspiraciones futuras. Sin embargo, resultó difícil que los alumnos propusieran un proyecto que surgiera de su propia iniciativa. De los cuatro equipos conformados inicialmente, tres escogieron alguno de los proyectos sugeridos por el docente.

Una posible solución a este problema sería realizar actividades exploratorias al inicio del curso para familiarizar a los estudiantes con la variedad de proyectos de código abierto existentes. Una de las preguntas de la encuesta era ¿qué podría hacer el docente para que le resulte más fácil a los estudiantes conseguir un proyecto? Una de las respuestas fue:

Explicar en breve que se puede trabajar en estos proyectos: páginas web, finanzas, juegos, criptografía, biología, multimedia (audio / imágenes / etc.).

También se puede preguntar al estudiante sobre qué tecnologías maneja y qué tanto quiere contribuir en open source para indicarle opciones más concretas y decidir si un proyecto está por encima de lo que pueden hacer en el plazo dado o por debajo de las exigencias. Para esto, el docente deberá revisar, aunque sea por encima cómo es ese proyecto: si la comunidad es activa, si el código tiene al menos buen patrón de diseño, etc.

(Respuesta a las preguntas abiertas de la encuesta escrita)

Otras de las sugerencias de los alumnos fueron “preguntarle a los estudiantes y tomar en cuenta sus habilidades en diversos lenguajes” y “preguntarle a los estudiantes sus futuras aspiraciones”.

B. Relevancia del proyecto

En el proyecto tradicional, es posible diseñar los problemas en torno al contenido de la asignatura, mientras que en el proyecto abierto no es posible garantizar que los estudiantes practiquen los temas de la asignatura. En la encuesta, la pregunta ¿el proyecto abierto enseña contenidos relevantes para el curso? obtuvo un puntaje de 3,3 / 5, mientras que la pregunta ¿el proyecto abierto enseña contenidos relevantes para la carrera? recibió un puntaje de 5 / 5.

Sin embargo, bajo el enfoque tradicional seguido en otras asignaturas de la carrera, existe la preocupación entre los estudiantes de que los proyectos no son relevantes para la carrera.

Sí agarraría de nuevo el proyecto abierto porque uno puede irse por cosas que le interesan. A mí no me gusta mucho cuando en los proyectos te imponen algo que tú no quieres. Por ejemplo, hace unos semestres me mandaron a hacer un sistema de bachaqueo, una cosa así, y hacía el proyecto pero no me gustaba. En cambio cuando agarras el proyecto abierto puedes ir por lo que te gusta y por lo menos uno está más interesado haciendo lo que quiere.

(Respuesta a las preguntas abiertas de la encuesta escrita)

Por otra parte, al menos uno de los equipos aplicó en el mundo real varios de los temas principales de la asignatura:

Más que todo, lo relacionado a la materia fue la parte de manejo de memoria, porque los bitmaps son muy pesados, son bloques súper gigantes de memoria y normalmente las aplicaciones Android están muy a riesgo de sufrir el error de memory leak, que lo tienen varias. Incluso aplicaciones que ya son productos a veces tienen ese fallo. Así que para una galería de fotos esto era muy importante, porque ellos tenían que manejar todo eso.

Y, en cuanto a hilos, eso no fue tan difícil porque Java en Android Studio maneja los hilos muy bien, ya tienen funciones para eso. Lo que pasa es que el proceso de compresión consta de muchas etapas, de cuantización, de medición de color… Son siete etapas en total y es mucho procesamiento. Entonces para que mientras el usuario selecciona la opción, de una en background se fuera haciendo todo el proceso de la transformación de la imagen, se colocó en un hilo y de verdad que funcionó mejor, se notaba que era más rápida la respuesta de la interfaz y todo.

(Respuesta a las preguntas abiertas de la encuesta escrita)

A la pregunta ¿qué fue lo que más te gusto de realizar el proyecto abierto? una estudiante respondió:

Que hay colaboración para una comunidad internacional y que es un proyecto cool ya que se puede escoger hacer algo que nos gusta, acerca mucho más a la experiencia laboral.

Aunque no me fui por el que más me gustó (porque requería saber demasiadas cosas que no sé) ha sido a mi parecer, el mejor proyecto que he hecho por la uni.

(Respuesta a las preguntas abiertas de la encuesta escrita)

C. Compilar el código

Una de las mayores dificultades que enfrentaron todos los equipos fue compilar el código fuente. En particular, *GNOME* y *Firefox* son dos proyectos con una gran cantidad de archivos y líneas de código, lo cual hace más difícil la compilación. Los equipos que trabajaron con estos proyectos no lograban que los cambios que hacían en el código se viesen reflejados tras compilarlo.

Hacíamos cambios y no pasaba nada, escribimos un correo y no nos respondieron, entonces como que GNOME no se hizo nada amigable. Y aparte la computadora de Nathaly se calentaba, la mía [Raúl] se pegaba… fue complicado.

(Entrevista al estudiante Raúl)

Estas dificultades hicieron que el docente abandonara la lista de proyectos sugeridos y en su lugar publicase una lista de proyectos recomendados en Android. Generalmente estas aplicaciones son desarrolladas en Android Studio y resultan más amigables para compilar. Aún así, el proceso no fue fácil.

Hay que tener paciencia para obtener las herramientas requeridas, en este caso, porque tardamos más días de lo que esperábamos instalando todo para que funcionara, porque no paraban las instalaciones.

(Entrevista al estudiante Raúl)

D. Entender el código

Un proyecto de código abierto puede tener cientos de archivos fuente y cientos de miles de líneas de código, por lo cual es difícil para alguien nuevo entender el código y cuáles son los cambios que debe realizar.

Es necesario leer mucho el código de otros así como la documentación oficial para tener una idea general de la manera en que está organizado y programado el proyecto. Uno de verdad tiene que ir con mente muy abierta, tranquila y serena porque uno se está enfrentando a un código nuevo, grande, denso, dependiendo del caso. Entonces uno tiene que tener mucha calma para entender todo lo que tú necesitas, porque ese fue uno de los consejos que nos comenzó a dar el profesor cuando comenzamos: no traten de entender tanto todo el código sino específicamente vayan a lo que ustedes necesitan saber para poder manejarlo. Entonces por allí saber a dónde ir, saber hacer una búsqueda inteligente de qué es lo que vas a manejar, dónde tienes que concentrarte, qué tienes que cambiar o agregar, para que no te vuelvas loco en el intento.

(Presentación en clase del estudiante Raúl sobre el proyecto abierto)

También es fundamental para entender el código la comunicación con los encargados del proyecto, como se verá en la próxima sección.

E. Comunicación con los encargados

La comunicación con los encargados del proyecto resultó esencial para orientar y motivar el trabajo de los equipos, y puede hacer la diferencia entre una experiencia exitosa o fallida.

[En GNOME] había poca documentación… Entonces no era fácil para alguien que es nuevo, aparte que nos vinieron a responder la semana pasada [un mes después].

En el caso de Leafpic, fue todo por Telegram. Dijimos 'qué pena, de la nada vamos a llegar a este poco de gente a pedirle información' y cuando nos respondieron nos unieron al grupo y todo, nos dieron la bienvenida, y de verdad es muy emocionante porque uno empieza a sentir que está esa comunidad ahí. Todo un mundo por conocer, todo un poco de personas que les apasiona la programación… Tratamos de no preguntarles tantas cosas, y si les preguntábamos era algo bastante específico. Fueron muy claras sus indicaciones, ellos estaban muy muy claros, tenían mucho dominio de esto, en algunos casos nos indicaron las clases y todo, y daban respuestas más allá de las que estábamos pidiendo.

(Entrevista a la estudiante Nathaly)

El proyecto abierto resultó ser una experiencia para los estudiantes de socializar con desarrolladores de otros países:

Aunque creo que no entendieron el nivel de novatos que somos, porque cuando yo les escribí, les dije que estoy estudiando, no les dije tercer año de carrera, porque no vaya a ser que era mucho, les dije segundo, y ellos me respondieron "ahh estás en la universidad, debes saber mucho", ellos ni siquiera estaban en la universidad.

Entonces eso también inspira a que nosotros pudiéramos hacer algo así, dedicarle un poco más de tiempo para ir más allá y a cosas que nos gustan.

(Presentación en clase de la estudiante Nathaly sobre el proyecto abierto)

F. El docente

Una de las principales dificultades enfrentadas por el docente fue tutorizar a los equipos, porque los temas de cada proyecto eran muy disímiles entre sí. Entre las distintas áreas con que trabajaron los estudiantes está el patrón de inyección de dependencias, el desarrollo de interfaces gráficas con XUL y el desarrollo en Android, todos ellos temas en los cuales el docente no tenía experiencia previa.

Por otra parte, fue difícil mantener un balance entre la cantidad de esfuerzo realizada por cada equipo. Dos de los equipos contribuyeron sólo unas pocas líneas de código a su repositorio, mientras que un equipo aportó más de 200 líneas de código. El proyecto abierto es una actividad poco estructurada que requiere de esfuerzo y de un trabajo automotivado, y claramente algunos equipos demostraron más esfuerzo que otros.

Otro obstáculo fue la disparidad entre el nivel de experiencia previa de cada equipo. Uno de los equipos no entendía los principios básicos de cómo compilar una aplicación de software extensa como *Mozilla Firefox*. Otro equipo tenía buenas habilidades previas de programación, pero no tenía experiencia específica en desarrollo Android. El tercer equipo ya tenía experiencia programando más allá de sus actividades como estudiantes de la carrera e incluso tenía cierta experiencia con el uso de bibliotecas de código abierto.

Para el docente la actividad representó un reto multifacético ya que implicaba tutorizar a los grupos acerca de temas que no dominaba, y al mismo tiempo, asumir cierto rol de gerente de proyectos para garantizar la correcta y continua dedicación de los estudiantes al proyecto. Esto hizo que la experiencia del proyecto abierto exigiera mucho más esfuerzo que la experiencia del proyecto tradicional.

vii. CONCLUSIONES

Una de las mayores dificultades para el docente fue lograr que los estudiantes trabajaran en temas relacionados con las competencias de conocimiento de la asignatura de sistemas operativos. Generalmente contribuir a un proyecto de código abierto requiere habilidades varias como la programación en varios lenguajes, uso de herramientas, conocimientos de Ingeniería de software, comprensión de distintos tipos de algoritmo, etc., por lo tanto es difícil enfocar la actividad dentro de los conocimientos de Sistemas Operativos, que son sólo un subconjunto de los requeridos.

Por otra parte, debido a la naturaleza abierta del proyecto, fue difícil evaluar si los estudiantes estaban haciendo el esfuerzo necesario, dedicándole el tiempo adecuado al proyecto y haciendo la investigación requerida. Además, los estudiantes no tenían una miríada de temas en los que querían trabajar, al contrario, la mayoría de los grupos se limitó a escoger uno de los temas propuestos por el docente.

Sólo los estudiantes con un alto nivel de automotivación lograron llevar a cabo la experiencia de forma exitosa. Los demás hicieron un proyecto con un alcance muy limitado o no lograron culminar exitosamente la actividad.

El proyecto requería de una serie de competencias en los estudiantes, disposición a investigar, motivación propia, persistencia, comunicación efectiva con la comunidad de desarrolladores, entre otras. Desarrollar un proyecto abierto en un contexto sociocultural donde no son comunes estas competencias es difícil.

Para llevar a cabo el proyecto abierto de forma exitosa, sería necesario formar a los estudiantes desde una etapa temprana en la carrera hacia este fin. En los cursos introductorios de informática sería pertinente explicar qué es el software de código abierto y cuáles son las prácticas de esta comunidad. También es necesario sensibilizar a los estudiantes sobre todas las aplicaciones de la computación, cómo con ella es posible contribuir a la Ciencia, a la vida cotidiana de las personas, a la industria. Desde una etapa temprana de la carrera sería necesario asignarle a los estudiantes proyectos libres con una actitud hacia la investigación, que produzcan verdaderos resultados originales y valiosos, así sea en un contexto reducido.

Sólo con una cultura académica donde se fomente desde el inicio este tipo de actividades sería posible hacer que ésta sea una experiencia con mayor tasa de participación y éxito.

viii. RECOMENDACIONES

A continuación se sugieren varias indicaciones que pudiesen mejorar la experiencia del proyecto abierto en instancias futuras. En primer lugar, el docente debería dar una lista amplia de sugerencias de proyectos, que abarque una gran cantidad de repositorios de código abierto de diferente índole, en diversos lenguajes de programación y a través de diferentes disciplinas. El docente podría diseñar una serie de actividades exploratorias que permitan al estudiante familiarizarse con el concepto de código abierto, conocer a la comunidad del software libre, aprender a explorar repositorios de su interés, comprender los diversos tipos de licencia (GPL, MIT, entre otras), familiarizarse con proyectos para los distintos sistemas operativos (GNU/Linux, Windows) y explorar proyectos dentro de varias plataformas (escritorio, móvil o *backend*).

Algunos de los estudiantes que participaron en el proyecto abierto dieron recomendaciones sobre cómo mejorar la experiencia, y uno de ellos indicó que el docente debería decir que se puede trabajar en temas tan variados como “páginas web, finanzas, juegos, criptografía,

biología, multimedia (audio / imágenes / etc.)". Esto permitiría motivar al estudiante a escoger un tema de su interés.

Otra recomendación es enfocarse en proyectos con una base de código más pequeña en lugar de grandes entornos de desarrollo como *GNOME* (un ambiente de escritorio para UNIX), el cual tiene muchas bibliotecas y la compilación de las aplicaciones es compleja, al estar muy atada al ambiente de desarrollo.

Uno de los criterios más importantes para elegir un proyecto es que tenga una comunidad de desarrollo activa. La calidad del equipo de desarrollo se puede medir observando el número de contribuidores, si el proyecto tiene cambios recientes, si el proyecto tiene un rastreador de problemas (*issue tracker*) activo y las solicitudes son atendidas oportunamente, entre otros. Las plataformas en línea de control de versiones como *GitHub* ofrecen una serie de herramientas para medir la "salud" de un proyecto, como el "pulso" que permite observar la frecuencia con que han sido realizado cambios, y la "comunidad" que permite ver cuáles son los principales contribuidores de un proyecto y cuáles son los cambios realizados por cada uno. Es fundamental que el proyecto tenga una comunidad de desarrollo activa para que puedan responder oportunamente a las dudas de los estudiantes.

Adicionalmente, se recomienda realizar una evaluación diagnóstica de los estudiantes antes de iniciar el proyecto abierto para conocer sus áreas de interés, hobbies, los lenguajes de programación que desean aprender y aquellos con los cuales tienen experiencia previa. De esta forma, se puede unir la oferta de proyectos con estos criterios para ofrecer a los estudiantes proyectos en los que tengan mayor interés de trabajar o que puedan afrontar con mayor facilidad.

ix.

AGRADECIMIENTO

Una versión preliminar de este trabajo fue realizada dentro del curso *Investigación estudiantil: posibilidades y retos* de la profesora Aurora Lacueva.

REFERENCIAS

[1] FERNANDEZ, A. K. (Jefe del Departamento de Ingeniería de Software, Escuela de Informática UCAB). Comunicación personal (2022-01-12).

[2] DURANT, E., IMPAGLIAZZO, J., CONRY, S., REESE, R., LAM, H., NELSON, V., ... & MCGETTRICK, A. (2015). CE2016: Updated computer engineering curriculum guidelines. 2015 IEEE Frontiers in Education Conference (FIE). IEEE. Recuperado de https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf

[3] UNIVERSIDAD CATÓLICA ANDRÉS BELLO (2015). Programa de la asignatura de sistemas de operación.

[4] LACUEVA, A. (2015). La investigación en la escuela -casa de la cultura. Proyectos, actividades y recursos. Saber UCV, Repositorio Institucional de la Universidad Central de Venezuela. Caracas. Recuperado de https://www.academia.edu/19807431/La_investigaci%C3%B3n_en_la_escuela_casa_de_la_cultura_Proyectos_actividades_y_recursos

[5] KNOLL, M. (1997). The project method: Its vocational education origin and international development. Journal of Industrial Teacher Education, 34(3), 59

[6] KNOLL, M. (2012). "I had made a mistake": William H. Kilpatrick and the project method. Teachers College Record, 114(2), 1-45.

[7] EUROPEAN UNION INTELLECTUAL PROPERTY OFFICE (2020). Open Source Software in the European Union. Recuperado de https://euipo.europa.eu/tunnel-web/secure/webdav/guest/document_library/observatory/documents/reports/2020_Open_Source_software/2020_OSS_Full_EN.pdf

[8] PHILLIPS, D. E. (2009). The software license unveiled: how legislation by license controls software access. Oxford University Press.

[9] OPEN SOURCE INITIATIVE (2018). History of the OSI. Recuperado de https://opensource.org/history

[10] SOWE, S. K., & STAMELOS, I. G. (2007). Involving software engineering students in open source software projects: Experiences from a pilot study. Journal of Information Systems Education, 18(4), 425. Recuperado de https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.300.6391&rep=rep1&type=pdf

[11] KUSSMAUL, C. (2009). Software projects using free and open source software: Opportunities, challenges, and lessons learned. 2009 Annual Conference & Exposition of the American Society for Engineering Education (ASEE), Austin, Texas. 10.18260/1-2--5395. Recuperado de https://peer.asee.org/software-projects-using-free-and-open-source-software-opportunities-challenges-and-lessons-learned

[12] KUSSMAUL, C. (2009). Software projects using free and open source software: Opportunities, challenges, and lessons learned. 2009 Annual Conference & Exposition of the American Society for Engineering Education (ASEE), Austin, Texas. 10.18260/1-2--5395. Recuperado de https://peer.asee.org/software-projects-using-free-and-open-source-software-opportunities-challenges-and-lessons-learned

[13] PAPADOPOULOS, P. M., STAMELOS, I. G., & MEISZNER, A. (2013). Enhancing software engineering education through open source projects: Four years of students' perspectives. Education and Information Technologies, 18(2), 381–397. Recuperado de https://www.researchgate.net/profile/Pantelis-Papadopoulos/publication/257959312_Enhancing_software_engineering_education_through_open_source_projects_Four_years_of_students'_perspectives/links/56f96be708ae38d710a2ffb5/Enhancing-software-engineering-education-through-open-source-projects-Four-years-of-students-perspectives.pdf

[14] SMITH, T. M., MCCARTNEY, R., GOKHALE, S. S., & KACZMARCZYK, L. C. (2014). Selecting open source software projects to teach software engineering. Proceedings of the 45th ACM technical symposium on Computer science education, 397–402. Recuperado de https://lisakaczmarczyk.com/wp-content/uploads/2021/07/SmithMcCartneyGokhaleKaczmarczyk-2014.pdf

[15] NASCIMENTO, D. M., CHAVEZ, C. F., & BITTENCOURT, R. A. (2018). The adoption of open source projects in engineering education: A real software development experience. 2018 IEEE Frontiers in Education Conference (FIE), 1–9. Recuperado de http://www2.uefs.br/roberto/papers/FIE2018_open_source_real_experience.pdf