Tema: 1. Docencia a distancia en la educación superior; 2. Evaluación en la educación a distancia; 6. Experiencias docentes en líneas

"Uso del sistema de control de versiones Git en reemplazo de un sistema de administración de la enseñanza"

Gunnar Wolf [a,b]

[a] *Facultad de Ingeniería, UNAM*
[b] *Instituto de Investigaciones Económicas, UNAM*
*****Gunnar Wolf. Dirección de correo electrónico: gwolf@gwolf.org*

R E S U M E N

Desde hace varios años, pero con especial fuerza a partir del inicio de la contingencia por COVID, los profesores de educación superior hemos acudido a herramientas telemáticas que nos ayuden a gestionar los trabajos con que nuestros alumnos practican y demuestran su avance en clase — Tareas, prácticas, proyectos, exposiciones y demás. Los sistemas que, en el transcurso de las últimas dos décadas, más se han utilizado para tal fin toman el nombre de Sistemas de Administración de la Enseñanza (LMS, por sus siglas en inglés); el más conocido es indudablemente Moodle, pero hay una gran cantidad de opciones a éste.

Desde hace varios años, y a partir de la observación empírica de un rechazo por parte de los alumnos al uso de estos sistemas, el autor tomó la decisión de reemplazar por completo su uso por el del sistema de control de versiones Git, específicamente para la enseñanza a alumnos de la carrera de Ingeniería en Computación, para la materia de Sistemas Operativos (6º semestre plan 2016 FI-UNAM).

Esta ponencia presenta la justificación para la elección de esta plataforma, la evaluación de la experiencia tras ocho semestres empleándola, invitando a otros docentes a considerar la adopción de esta metodología.

Palabras Clave: Sistema de control de versiones, sistema de administración de la enseñanza, Git, Aprendizaje en línea

1 Introducción

La experiencia que presenta este trabajo se ubica dentro de la enseñanza de la materia *Sistemas Operativos* de la carrera de *Ingeniería en Computación*, impartida en la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.

1.1 Antecedentes: Marco tecnológico

El desarrollo de software, a cualquier escala seria, siempre ha requerido de la coordinación de esfuerzos entre desarrolladores, y resultaría natural que herramientas con éste fin hayan sido desarrolladas desde muy temprano en la historia de la computación. El modelo de desarrollo que se seguía hasta la década de 1960 era muy diferente, pero conforme se popularizó el uso interactivo de las computadoras mediante los sistemas de acceso compartido y las terminales interactivas,esta necesidad se hizo patente. El primer documento que conocemos que cubre sistemas de control de versiones (SCVs) menciona (traducción propia):

> Dado que *sccs* representa un cambio tan radical de los métodos convencionales de control de código fuente, resultó claro cuando comenzamos a desarrollarlo (a fines de 1972) que presentar un artículo con la especificación no sería suficiente para "vender" el sistema a los proyectos de software para los que estaba pensado; debíamos tener un prototipo funcional. [1]

Tal como lo supusieron los autores, la idea tardó varios años en lograr la tracción necesaria para ser común en el desarrollo de proyectos de software; hacia mediados de la década siguiente, las principales herramientas para este fin eran *sccs* y *RCS* [2]; hacia fines de los ochenta, y motivado porque los tres participantes de un proyecto tenían horarios incompatibles entre sí, se desarrolló *CVS*, inicialmente un conjunto de funciones ayudantes sobre RCS, que facilitó el desarrollo paralelo y sin coordinación explícita [3]. Varios años más tarde, resulta claro: "una forma de predecir si un proyecto de software tendrá éxito es preguntar, ¿utilizan sus desarrolladores un sistema de control de versiones para coordinar su trabajo?" [4]

Los grandes proyectos de software libre iniciados a principios de los noventa (particularmente, los sistemas operativos Linux, FreeBSD, NetBSD y OpenBSD, y una gran cantidad de aplicaciones) iniciaron su desarrollo empleando CVS como punto nodal.

La década de los noventa fue testigo de un crecimiento vertiginoso tanto en los proyectos de software libre como en la capacidad de los equipos de cómputo y la universalidad del acceso a red (puede argumentarse que el primero se debió a la conjunción de los otros dos). El modelo de uso de CVS comenzó a resultar insuficiente; CVS carecía de capacidad para representar acciones tan comunes como la eliminación o el cambio de nombre de un archivo e imponía un alto costo a trabajar con archivos que no fueran de texto plano.

Entre el 2000 y 2004 se desarrolló Subversion; éste sistema seguía el mismo modelo de interacción de CVS, corrigiendo estas y otras debilidades [5]. Subversion creció en pocos años y se convirtió en una de las principales herramientas de desarrollo en el ámbito del software libre.

Sin embargo, todos los SCVs mencionados hasta este momento operan de forma centralizada. Con el crecimiento de algunos proyectos a escalas de desarrollo inimaginables hasta ese momento (8,000 desarrolladores en 20 años, 15 millones de líneas de código y más de 37,000 archivos [6]), en 2002 Linus Torvalds tomó la controvertida decisión de dejar CVS y adoptar SCV *distribuido* (SCVD) y de uso gratuito, pero no libre — Un sistema en que no existiera una copia maestra o servidor central, sino que cada copia del proyecto guardara toda la historia y estado, permitiendo sincronización entre ramas relacionadas. Dado que no había ningún SCV libre que implementara el modelo distribuido, Torvalds adoptó BitKeeper.

Si bien políticamente la adopción de BitKeeper fue difícil y un punto recurrente de fricción con los puristas de la libertad de software, es indudable que ayudó tremendamente a recuperar la velocidad en el desarrollo de Linux, que venía sufriendo por varios años [7]. BitKeeper ofrecía una licencia gratuita (no libre) para los desarrolladores de software libre, siempre y cuando no se utilizara para competir con BitKeeper mismo.

Para 2005, se suscitó una discusión acerca de si la funcionalidad que Andrew Tridgell estaba agregando a Linux consistía una violación de la licencia [8], lo cual eventualmente llevó a que Torvalds mismo iniciara el desarrollo de un SCVD, al que llamó Git.

En el mismo periodo de tiempo se desarrollaron varios otros SCVDs libres, como *Monotone*, *Darcs*, *Mercurial* o *Bazaar*. Existen además otras alternativas propietarias, como *Team Foundation Server* de Microsoft. Sin embargo, Git ha resultado claramente ganador sobre de los demás, y hoy puede verse como *lingua franca*, ya no únicamente entre los desarrolladores de software libre, sino que en el mundo de la programación en general; ha aglutinado no únicamente a una clara mayoría de proyectos entre los SCVDs, sino que ha logrado atraer a una gran cantidad de proyectos que tenían incluso más de veinte años de historia sobre SCVs centralizados [9].

De forma paralela a lo ya presentado, desde fines de los noventa comenzaron a aparecer las forjas, sitios Web dedicados al hospedaje de proyectos de software libre, a los cuales brindan recursos administrados como un espacio Web, seguidor de fallos, listas de correo — y un SCV. Según la información disponible en el mismo sitio primera de estas forjas, SourceForge, hospeda al día de hoy más de 500,000 proyectos y tiene “varios millones” de usuarios registrados. Sin embargo, el flujo de interacción en que está basado es poco amigable, lo cual lo hace apto únicamente para usuarios que ya son profesionales del desarrollo de software.

En 2008, y ya viendo un rápido crecimiento en la adopción de Git para proyectos de todo tamaño, nació *GitHub*: Una forja con un flujo de trabajo simplificado, y fuertemente centrado en el modelo de desarrollo de Git. Apenas tres años más tarde era ya la forja con mayor actividad en el mundo del software libre [10]. A la fecha de escritura del presente texto, según la información disponible en el sitio Web, hospeda a más de 100 millones de proyectos y más de 50 millones de desarrolladores.

Ahora, si bien GitHub se ha vuelto nodal para el desarrollo de esta impresionante cantidad de proyectos libres, causa una cierta disonancia cognitiva que GitHub mismo no es libre: El software con el cual opera el sitio Web, y que integra las diferentes herramientas que lo componen, es un desarrollo propietario. Hay otros servicios comparables, como *GitLab*, que incluso ofrece un modelo de colaboración y una semántica perfectamente mapeable contra la de GitHub. Resultaría probablemente más coherente con los principios personales de uso y promoción del software libre que se mencionaron al cierre de la sección anterior el desarrollar la experiencia que se relata a continuación en GitLab o alguna plataforma similar; la decisión de hacerlo en GitHub no fue tomada a la ligera, y se centra en la importancia que dicho sitio tiene para el desarrollo de software en general. Al día de hoy, prácticamente todos los proyectos libres de desarrollo están alojados en GitHub (sea que éste provee su espacio principal de desarrollo o que es elegido como un almacenamiento de respaldo o conveniencia).

1.2 Acerca del documento

Este trabajo se presenta como una re-elaboración y actualización de una versión muy preliminar, presentada en el 8º Encuentro en Línea de Educación y Software Libre EDUSOL, y publicado en extenso [11]; los principales aportes desde donde llega dicho texto son:

- Estudio realizado sobre un periodo mucho mayor
- Mayor profundización sobre trabajos relacionados
- Elaboración y reporte de una encuesta a los alumnos, validando varios de los puntos presentados

2 Trabajos relacionados

El uso de SCVs como mecanismo para la entrega de trabajos de clase no es una idea nueva. Pueden encontrarse reportes de experiencia desde 1989 [12], aunque en esa ocasión, al docente no le fue posible concluir la

implementación por la dificultad de los alumnos en comprender el sistema. Citando al autor, en traducción propia:

> Se le dio a los alumnos una copia legible por computadora del código fuente, bajo un sistema de control de versiones. Para la desilusión de los instructores, el sistema de control de versiones no fue utilizado, y se dará mayor énfasis a los beneficios de dicho sistema en futuras iteraciones del curso.

Sin embargo, no fue posible localizar información respecto a experiencias posteriores.

Varios trabajos reportan experiencias en el mismo sentido que el nuestro, aunque comprensiblemente, partiendo de diferentes herramientas, obedeciendo al punto en el tiempo en que se presentaron: En todos los casos, se refieren a VCSs centralizados [4,13,14]. Tras varios años de no encontrar literatura al respecto, una referencia mucho más cercana al trabajo que aquí se presenta puede encontrarse en [15].

2.1 Cursos empleando VCSs centralizados

Casi todas las experiencias reseñadas presentaban el uso de un VCS siguiendo un modelo de un repositorio independiente por alumno. Esto resulta antitético con el uso obtenido que se espera de la implementación para trabajar con desarrollo colaborativo: Si cada uno de los alumnos tiene un repositorio independiente, ¿cómo se gestionan las entregas de trabajos grupales? ¿Debe cada miembro del equipo enviar por separado al profesor la misma entrega? ¿O uno sólo de ellos hace la entrega, notificando *fuera de banda* al profesor dónde buscar el trabajo? Al discutir respecto a este punto, Reid y Wilson [4] mencionan incluso un importante punto que se considera nodal para la discusión: “prevenir que los alumnos vean uno el trabajo del otro”. Los sistemas de desarrollo colaborativo son, entonces, presentados en un entorno más bien *adversarial*, trabajando en contra de la colaboración natural que deberían implementar.

La implementación de Milentijevic, Ciric y Vojinovic [14] presenta una experiencia construida sobre una base tecnológica de SVN o CVS, pero necesariamente mediados por una interfaz Web. No se maneja un único repositorio a lo largo del curso, sino que se separando cada entrega: a lo largo del curso, se genera un nuevo repositorio para cada tarea o proyecto; cada alumno debe obtener el repositorio. Las entregas, explican los autores, incluyen el ciclo de vida de cada uno de estos desarrollos: Definición y asignación de tareas, trabajo en la tarea, terminación y evaluación. Cada una de estas etapas requiere de un diferente tipo de interacción entre supervisor y alumnos, lo cual supone una gran cantidad de trabajo *burocrático* (cubierto por el sistema Web) para el modelo presentado.

La propuesta de Glassy [13] contrasta con las dos anteriores, dado que su metodología no busca el uso de un VCS como un mecanismo para la entrega de tareas y proyectos, sino que como una herramienta para evaluar las prácticas de desarrollo de sus alumnos. Cada alumno crea su repositorio Subversion, realiza el trabajo, y entrega por correo electrónico al repositorio entero. Glassy analiza los patrones de *commits* que forman parte del repositorio, y documenta patrones no demasiado distintos de los que se presentarán en la Figura 3 de la Sección 3.

2.1 Cursos empleando VCSs distribuidos

La sección de *Trabajos relacionados* de Glassy [13] hace referencia a un curso en su misma universidad (Montana Tech) en primavera de 2005, basado en *Darcs*. Este sería el primer caso de un curso sobre un VCSD, y la experiencia sería muy interesante de analizar, pero no fue posible encontrar mayor información al respecto.

GitHub opera una comunidad denominada *GitHub Education Community*, dedicada a generar contacto entre docentes y alumnos y a discutir el uso de la tecnología de SCVDs para fines educativos. Ofrece además ayuda organizacional para repositorios Git, mediante *organizaciones* que operan *Salones GitHub*; el flujo que presenta es específico a la configuración propuesta y no refleja al trabajo colaborativo en un espacio profesional. Cada alumno trabaja y realiza sus entregas en su propio repositorio. GitHub Classrooms se desarrolló originalmente como software libre, pero fue convertido en un proyecto interno y cerrado de GitHub a inicios de 2020 [16].

Glassey publicó [17] una comparación de herramientas para llevar control de grupos, enfocándose en cursos implementados sobre Git y que utilizan GitHub, pero enfocado en las herramientas desarrolladas por los distintos docentes para facilitar el seguimiento de las entregas por parte de los alumnos. El trabajo que se presenta aquí no tiene ni requiere de herramientas externas para este fin — si bien sólo se ha empleado con grupos medianos (hasta 40 alumnos), es opinión del autor que, con mucho, la mayor carga en tiempo para el docente es la calificación de las entregas, no el dar seguimiento a si los proyectos ya fueron entregados o no.

Una parte particularmente relevante del artículo de Glassey es la comparación entre los *modelos de distribución* de los diferentes sistemas: Siete de ocho flujos que revisa distribuyen los trabajos a los alumnos siguiendo el modelo que denomina *ORpSpA* (*One Repository per Student per Assignment*, un repositorio por alumno por entrega), y el restante, *OBpP* (*One Branch per Problem*, una rama por problema). La implementación aquí reseñada es independiente y novedosa respecto a todos los casos de la revisión de Glassey.

3. Implementación

La Facultad de Ingeniería ofrece varias instalaciones de sistemas LMS, particularmente el *cluster* llamado EDUCAFI, que cuenta con un Moodle para cada una de las divisiones de la Facultad, administrado por la Unidad de Servicios de Cómputo Académico (UNICA).

La experiencia docente que relatamos inicia en enero de 2013 con el semestre 2013-2. Desde el primer curso, se tomó la decisión de emplear EDUCAFI como herramienta

parcial para la entrega de tareas y para comunicar a los alumnos de bibliografía relacionada y otras actividades que se fueran cubriendo; el uso que se dio a Moodle como LMS fue relativamente limitado, con únicamente dos ejercicios realizados a partir de cuestionarios desarrollados con calificación asignada, y sin emplear ningún módulo adicional.

En general, la administración realizada por el grupo de becarios que conforma la atención a usuarios en UNICA es buena y ágil, pero circunscripta a sus políticas de operación; particularmente, el docente solicitaba que los cursos ya impartidos se preservaran como memoria y para poder comparar el progreso con siguientes experiencias. Por esto, y por ocasionales problemas de algunos alumnos con problemas en el manejo de sus cuentas, se tomó la decisión de migrar el LMS a otro Moodle, administrado por quien escribe, del Instituto de Investigaciones Económicas. Se consideró agregar algunos módulos de utilidad para la materia, como GeSHi para la presentación de fragmentos de código o el Virtual Programming Lab (VPL) para ofrecer un entorno de desarrollo donde pudieran resolver planteamientos de programación sin depender de un entorno específico; es necesario reconocer que esto quedó en mera intención por el tiempo que requiere conocer y aprovechar el entorno Moodle más allá de un uso casual.

Hacia el final del semestre 2016-2, se tomó la decisión de no buscar ofrecer a los alumnos herramientas como las que encontrarán en un entorno realista de desarrollo de software, sino que llevarlos a emplearlas en realidad — Git. Tras evaluar los puntos presentados en el marco tecnológico, en la preparación del semestre 2017-1 se preparó un repositorio Git llamado *clase-sistop-2017-01*, y se alojó en el espacio personal GitHub del docente, recibiendo la dirección https://github.com/gwolf/clase-sistop-2017-01/; en iteraciones posteriores, la dirección se simplificó para ayudar a los alumnos a encontrarla, y se ubicaron en el espacio de la *organización* de la Facultad (p.ej. https://github.com/unamfi/sistop-2020-2).

Con respecto a la estructura interna que se dio al repositorio, en un principio, contempló únicamente a los tres únicos directorios del repositorio para las entregas de los alumnos (tareas, prácticas y proyectos), con cada una de las entregas ubicada en un subdirectorio numerado dentro de éstos. Poco tiempo después, a ésto se agregó un directorio para las exposiciones, que si bien siguen lógicas de entrega distintas, pueden enmarcarse en el flujo general. Al poco tiempo, se hizo obvio que el repositorio Git resultaba también un medio ideal para transmitir contenido a los alumnos — Los ejemplos de código realizados en clase, referencias a sitios Web u otros recursos mencionados, etc.

Al presentar este esquema de trabajo, se explicita a los alumnos bajo qué supuestos se estructuran las entregas de la forma que se hace, mediante el siguiente planteamiento:

> *El repositorio del curso es un proyecto de desarrollo al que te interesa contribuir. Cuando el docente genera una nueva actividad, naturalmente "aparece" un importante defecto (un* bug*, en términos de desarrollo de software): tu participación no está registrada para esa actividad. Lo que debes hacer es proveer un parche para ese defecto. Para hacerlo, actualiza tu copia local del proyecto, desarrolla los pasos que corrijan al* bug*, y envíaselo al docente para que éste lo incorpore al proyecto principal mediante un pull request.*

El primer semestre que se utilizó Git no se pidió a los alumnos seguir un estándar de nombres para sus entregas, por lo que se presentó la complicación de rastrear los archivos de vuelta al alumnos que los había generado; esto puede apreciarse en alguno de los directorios de entrega de tareas, como el que presenta la Figura 1.

Figura 1: Ejemplo de directorio de entrega de tarea en el primer semestre de aplicación. Nótese que varios de los alumnos denominarion a sus archivos empleando pseudónimos, no nombres reales, dificultando la relación de las calificaciones.

Además de dicha adecuación, la estructura de los repositorios utilizados semestre a semestre se ha mantenido básicamente inalterada para los semestres subsecuentes.

Dado que, en general, los alumnos no están familiarizados con Git al iniciar el curso (ni con ningún otro SCV), fue considerado necesario el presentar su funcionamiento

básico mediante tres prácticas; se explicó a los alumnos que, para efectos de calificación, si bien la entrega de tareas es obligatoria (un alumno que no entregue la totalidad de tareas pierde derecho a exención, y un alumno que no entregue el 80% de tareas pierde derecho al examen final en primera vuelta), las prácticas son opcionales, y su efecto únicamente es el de subir la calificación obtenida.

Las prácticas relativas al uso de Git llevan son:

1. *Uso de Git y Github:* Consta de ocho etapas, con instrucciones completas y detalladas, al estilo tutorial. Pide al alumno que genere una cuenta en GitHub en caso de no tenerla, que haga un *fork* (copia en su espacio personal, donde tiene permisos de modificación) del repositorio principal, lo *clone* a su computadora personal, genere un archivo con su nombre dentro del directorio de entrega de dicha práctica, lo envíe a su *fork* en GitHub, y notifique al docente de sus cambios mediante un *pull request* (solicitud de incorporar los cambios de un *fork* en la rama principal).

2. *Ramas paralelas de desarrollo:* Una característica central del éxito de Git es que facilita trabajar en distintos aspectos de forma muy eficiente: Mediante el uso de *ramas temáticas*, un desarrollador puede estar trabajando sobre distintas características de forma independiente. El manejo de ramas permite que los alumnos puedan comenzar desde temprano a desarrollar sus proyectos y exposiciones (actividades que se busca que se repartan a lo largo del tiempo) sin que esto les impida hacer entregas menores, como otras prácticas o tareas.

3. *Eliminando archivos innecesarios:* Una importante provisión de todo SCV es la capacidad de ignorar los archivos autogenerados, resultantes de la compilación o ejecución de un programa. Esta práctica se incluye para evitar que los alumnos se confundan con los cambios en dichos archivos, y para llevarlos hacia buenas prácticas del desarrollo colaborativo.

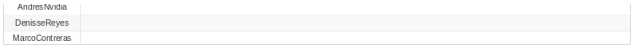

Figura 2: Gráfica de la red de interacciones entre repositorios para el semestre 2017-1. Cada punto representa un commit*. Pueden observarse patrones que indican fechas de entrega (entre los días 15 y 16), el anuncio de una nueva tarea a realizar (día 17), manejo de líneas paralelas de desarrollo por parte de los alumnos (bifurcación de la línea azul alrededor del día 10 y de la amarilla el 17). Las líneas inferiores corresponden a alumnos que dejaron algún trabajo pendiente, al no haber enviado un pull request, o no haber cumplido con los cambios requeridos en alguno de ellos.*

En las experiencias que se ha tenido con el manejo de Git se ha dedicado también tiempo de clase para explicar el modelo; si bien no se hicieron mediciones precisas, el tiempo total dedicado se estima en menos de dos horas y media – Aproximadamente una sesión de clase en el transcurso del semestre. El modelo de interacción se ilustra en la Figura 3, aunque se invita a consultar la versión pública e interactiva de dicha gráfica para explorarla y comprender mejor los diversos eventos que registra.

4. Resultados

Git no es un LMS, ni con el presente trabajo se busca presentar su aplicabilidad de forma genérica como si lo fuera. La experiencia que aquí se relata se circunscribe necesariamente al escenario planteado: Una materia orientada al desarrollo de software, y un compromiso personal del docente de formar a los alumnos en el uso de las herramientas de desarrollo colaborativo que muy probablemente serán valiosas para su desempeño profesional.

Si bien la experiencia ha sido exitosa desde el punto de vista subjetivo del docente, en conversaciones informales con algunos alumnos, y en los comentarios vertidos en las evaluaciones docentes que la Facultad habitualmente realiza al término del semestre, se aplicó una encuesta anónoma a los alumnos de los semestres 2017-2 a 2020-2; no se envió al primer grupo en usar Git (2017-1) por no contar con sus direcciones electrónicas. La encuesta se generó en la plataforma gratuita en línea *QuestionPro.* La participación en la encuesta se buscó enviando un único correo a los 224 alumnos que formaron parte de alguno de los grupos en cuestión; la encuesta se mantuvo abierta para su llenado por una semana durante el mes de noviembre de 2020, y 30 de los alumnos enviaron sus respuestas. La distribución de respuestas sobre los grupos se presenta en la Tabla 1.

Tabla 1: Participación en la encuesta

Semestre	Alumnos totales	Respuestas recibidas	Porcentaje de respuesta
2017-2	28	2	7%
2018-1	22	4	18%
2018-2	33	2	6%
2019-1	32	3	9%
2019-2	31	8	26%
2020-1	34	10	29%
2020-2	44	5	11%

La encuesta consta de dos preguntas de selección múltiple, una pregunta de selección única, cinco preguntas de escala de Likert, y dos preguntas de texto abiertas. Se le presentaron a los alumnos divididas en tres grupos: 1. *Antes del cursado*, preguntas enfocadas a conocer a partir de qué puntos base llegaron al curso; 2. *Durante el cursado*, cómo fue su relación con Git en el transcurso del semestre, y 3. *Después del cursado*, cómo sienten que les

haya resultado el aprendizaje de Git después de cursar la materia.

4.1 Antes del cursado

Varios alumnos han cursado más de una vez la materia, razón por la cual la primera pregunta es de opción múltiple: *¿En qué semestre llevaste la materia de Sistemas operativos?* (ver Figura 3). Los 30 asistentes indicaron 35 participaciones semestrales, lo cual indica que hasta cinco de ellos fueron repitentes.

Se le preguntó también a los alumnos qué tan familiarizados estaban con Git antes de cursar la materia; los resultados se presentan en la Figura 4.

Figur

a 3: ¿En qué semestre llevaste la materia?

Puede apreciarse que, si bien la generalidad de los alumnos sabía ya *algo* respecto a Git, su familiaridad era muy baja; únicamente el 26% de los encuestados lo usaban activamente.

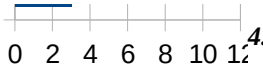

Figure ***4.2***

4: ¿Conocías a Git antes de cursar la materia?

Durante el cursado

Las preguntas de esta sección se presentaron en escala Likert de cinco puntos. El texto habitual de la escala Likert se reemplazó por equivalentes semánticos para faciltiar el acercamiento a los alumnos.

Como se describió con anterioridad, el manejo básico de Git se presenta con tres prácticas únicamente. ¿Es esto suficiente introducción como para conocer suficiente la herramienta para su uso a lo largo del semestre? La Figura 5 muestra que, efectivamente, los alumnos lo consideraron suficiente.

Figura 5: ¿Te parecieron suficiente presentación de Git las tres prácticas, mas las explicaciones en clase?

No es lo mismo el que una explicación sea suficiente a que el uso de la herramienta resulte natural. La Figura 6 muestra que, si bien el 16% consideró difícil o muy difícil usar la herramienta, la opinión mayoritaria sigue siendo claramente positiva.

Figura 6: ¿Recuerdas qué tan difícil te resultó hacer tus entregas y coordinarte con tus compañeros de equipo?

¿Qué tan bien respondió el uso de Git a la motivación original de presentar una alternativa a Moodle u otros LMSs? La Figura 7 apunta a que los alumnos resultaron, en general, muy satisfechos con este cambio.

Figura 7: ¿Qué te pareció el uso de Git en comparación con el de un LMS tradicional como Moodle (p.ej. Educafi)?

El uso de Git es sólo una parte (si bien fundamental) de la experiencia. El planteamiento se hizo siguiendo la terminología y tipos de interacción habituales en la plataforma GitHub. A excepción de uno, los alumnos indicaron que pudieron comprender el modelo de GitHub, y que mayoritariamente le ven valor futuro, como lo indica la Figura 8.

Figura 8: No únicamente fue Git: usamos este sistema mediante la interacción con el sitio Web GitHub. ¿Te pareció suficientemente clara la manera de trabajo, con las múltiples ramas, clones, manejo de usuarios, pull requests, pushes y demás?

La última pregunta de esta sección es de texto abierto, pidiendo a los alumnos que mencionen casos de uso o características que hubieran deseado que hubieran sido cubiertas. Si bien algunos mencionan puntos como el manejo de *issues*, los *stashes*, resolución de conflictos al hacer *merge*, el sentimiento general es de satisfacción con el nivel alcanzado en lo que se refiere al cursado de la materia.

4.3 Después del cursado

El proceso enseñanza-aprendizaje, sin embargo, debe evaluarse ante los resultados a largo plazo para los alumnos: ¿Qué tan útil les resultó el conocimiento adquirido? Para esto, una última pregunta presentada con un esquema Likert permite llegar con amplio optimismo al cierre de este reporte de adopción de herramienta: Como lo muetra la Figura 9, la totalidad de los alumnos le ven valor para su vida profesional, 70% de ellos aseverando que *es/será mi pan de todos los días*.

Figura 9: ¿Sientes que aprender el uso de Git te capacitó mejor para tu vida profesional?

Cerrando la encuesta, se da una última pregunta abierta pidiendo comentarios adicionales. El sentimiento es abrumadoramente positivo, con únicamente tres alumnos mencionando la dificultad que les presentó en su momento la herramienta; me permito citar a uno de ellos:

> *Ja ja pues es curioso porque lo odie durante la materia porque no entendía realmente mucho, sin embargo una vez que le agarre la onda le tome un cariño inmenso y ahora ya es como mi uña y mugre, aun no soy un máster pero ya me defiendo, de hecho en mi actual trabajo no se usaba y fui yo quien lo propuso y prácticamente capacite a mis compañeros para que aprendieran a usarlo, así que en definitiva me alegro haber tenido un acercamiento antes donde podía equivocarme y me explicaban donde fue su materia.*

Esas palabras resumen perfectamente, a fin de cuentas, la intención del docente al presentar el uso de Git: Brindar un reto intelectual que haga de los alumnos profesionales mejor capacitados para la vida profesional.

5. Conclusión

Después de ocho semestres concluidos y reportados en el presente trabajo, mas el semestre actual que mantiene esta misma lógica, la prueba ha resultado indudablemente exitosa. El uso de Git no es, como ya fue dicho, natural claro para alumnos de cualquier disciplina, pero por lo menos para las materias intermedias y avanzadas de Ingeniería en Computación, se invita a los docentes a adoptar los aprendizajes aquí reportados.

REFERENCIAS

[1] Rochkind, M. J. (1975). The source code control system. *IEEE Transactions on Software Engineering*, (4), 364-370.

[2] Tichy, W. F. (1985). RCS—a system for version control. *Software: Practice and Experience*, *15*(7), 637-654.

[3] Berliner, B. (1990, enero). CVS II: Parallelizing software development. En *Proceedings of the USENIX Winter 1990 Technical Conference* (Vol. 341, p. 352).

[4] Reid, K. L., & Wilson, G. V. (2005, February). *Learning by doing: introducing version control as a way to manage student assignments. ACM SIGCSE Bulletin* (Vol. 37, No. 1, pp. 272-276). ACM.

[5] Collins-Sussman, B., Fitzpatrick, B., & Pilato, M. (2004). *Version control with subversion*. " O'Reilly Media, Inc.".

[6] Brockmeier, K. (2012, abril) *Counting Contributions: Who Wrote Linux 3.2?* Linux.com, News for the Open Source Professional. Recuperado de https://www.linux.com/learn/counting-contributions-who-wrote-linux-32

[7] Henson, V., & Garzik, J. (2002, June). Bitkeeper for kernel developers. In Ottawa Linux Symposium (p. 197).

[8] Barr, J. (2005, abril). Bitkeeper and Linux: The end of the road? Linux.com, News for the Open Source Professional. Recuperado de https://www.linux.com/news/bitkeeper-and-linux-end-road

[9] de Alwis, B., Sillito, J. (2009, mayo). Why are software projects moving from centralized to decentralized version control systems?. En *Proceedings of the 2009 ICSE Workshop on cooperative and human aspects on software engineering* (pp. 36-39). IEEE Computer Society.

[10] Finley, K. (2011, junio) *GitHub has surpassed Sourceforge and Google Code in popularity*. ReadWrite.com. Recuperado de http://readwrite.com/2011/06/02/github-has-passed-sourceforge/

[11] Wolf, G. (2019). De Moodle a Git: Experiencia con el uso de unsistema de control de versiones (SCV) parareemplazar a un sistema de administración dela enseñanza (LMS). En *Prácticas abiertas: Educación y cultura libres* (pp. 92-108)

[12] BJ Cornelius, Malcolm Munro, and David J Robson (1989). “An approach to software maintenance education”. En: *Software Engineering Journal* 4.4, pp. 233–236.

[13] Louis Glassy (2006). “Using version control to observe student software development processes”. En: *Journal of Computing Sciences in Colleges* 21.3, pp. 99–106.

[14] Ivan Milentijevic, Vladimir Ciric, and Oliver Vojinovic (2008). “Version control in project-based learning”. En: *Computers & Education* 50.4 , pp. 1331–1338.

[15] Richard Glassey. “Adopting Git/Github within Teaching: A Survey of Tool Support”. En: *Proceedings of the ACM Conference on Global Computing Education*. ACM. 2019, pp. 143–149

[16] John Britton et al. GitHub Classroom. GitHub, 2015–2019. url: https://github.com/education/classroom/

[17] Richard Glassey. “Adopting Git/Github within Teaching: A Survey of Tool Support”. In: Proceedings of the ACM Conference on Global Computing Education. ACM. 2019, pp. 143–149