ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/321136899

De Moodle a Git: Experiencia con el uso de un sistema de control de versiones (SCV) para reemplazar a un sistema de administración de la enseñanza (LMS)

Conference Paper · October 2017

CITATIONS
0

READS
19

1 author:

Gunnar Eyal Wolf Iszaevich
Universidad Nacional Autónoma de México
33 PUBLICATIONS **7** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Strengthening trust of the cryptographic keyring for a geographically distributed project View project

Docencia en Sistemas Operativos View project

All content following this page was uploaded by Gunnar Eyal Wolf Iszaevich on 17 November 2017.

The user has requested enhancement of the downloaded file.

De Moodle a Git: Experiencia con el uso de un sistema de control de versiones (SCV) para reemplazar a un sistema de administración de la enseñanza (LMS)

Modalidad: Reporte de práctica

Gunnar Wolf
Instituto de Investigaciones Económicas, UNAM; Facultad de Ingeniería UNAM

Resumen

Los sistemas de la administración de la enseñanza (LMS) son ya de uso muy difundido tanto entre los profesores como entre los alumnos para asistir a las actividades no presenciales de la clase; si bien el esfuerzo necesario para conformar las actividades relacionadas a un curso no es trivial, al brindar no únicamente la capacidad de recoger la interacción, sino que de generar automáticamente calificaciones, resultan muy atractivos para los docentes.

Mi experiencia, particularmente a partir de experiencias personales con Moodle y Claroline, apunta a que a los alumnos les resulta incómodo y pesada la mediación de esta plataforma. Muchas veces, además, la información "naufraga" ante la atención del alumno al requerir de una jerarquización pre-establecida, a pesar de las ayudas que brindan estas plataformas.

En mi participación quiero relatar acerca de la experiencia de reemplazar por completo a Moodle mediante el uso del sistema de control de versiones Git; Git *no* está orientado de ninguna manera a su uso para la enseñanza, y si bien para el uso que damos empleamos la plataforma GitHub, no empleamos la infraestructura educativa que ésta misma ofrece.

Mi intención con esta ponencia es presentar la justificación para el elegir esta plataforma en particular para la materia *Sistemas Operativos* de la carrera de *Ingeniería en Computación* de la FI-UNAM, y relatar la experiencia a lo largo de tres semestres empleándola.

.

Palabras clave: Aprendizage en línea; Tecnología de la información (programas); Software de código abierto; Aprendizaje actvo; Plan de estudios.

ANTECEDENTES: MARCO DE ENSEÑANZA

La experiencia aquí descrita se ubica dentro de la enseñanza de la materia *Sistemas Operativos* de la carrera de Ingeniería en Computación, impartida por la Facultad de Ingeniería de la Universidad Nacional Autónoma de México. La Facultad de Ingeniería ofrece a sus profesores diversas herramientas de gestión de la enseñanza (LMS); ofrece principalmente sistemas Moodle, aunque cuenta también con instalaciones de Claroline.

El plan de estudios 2010 de la carrera[1] indica como objetivos del curso: *El alumno obtendrá las bases para administrar un sistema operativo, así como para diseñar y desarrollar software operativo*; a juicio de quien presenta este reporte de experiencia, dichos objetivos son formalistas y mínimos, por lo cual dentro de la presentación del curso[2] se incluyen los siguientes *objetivos adicionales*:

---

1 http://www.ingenieria.unam.mx/programas_academicos/licenciatura/Computacion/05/sistemas_operativos.pdf

2 https://gwolf.sistop.org/generalidades.html

- El alumno conocerá el desarrollo histórico de los sistemas operativos, lo que le llevará a comprender la razón de ser y el funcionamiento general de los diversos componentes de los sistemas operativos actuales.
- Aplicando el conocimiento obtenido sobre el funcionamiento general de los sistemas operativos, el alumno podrá sacar mejor provecho de la computadora.
    - Al emplearla como usuario final
    - Al administrarla
    - Al programar
- El alumno conocerá las principales herramientas que ofrecen los sistemas operativos libres para el desarrollo, monitoreo y administración.

La experiencia que se aborda a continuación, pues, se enmarca en la intersección de varios de estos objetivos: De los formales, el diseño y desarrollo de software operativo; de los adicionales, el conocer las herramientas que ofrecen los sistemas operativos libres para desarrollo, monitoreo y administración.

Cabe mencionar que, por razones que claramente exceden el ámbito del presente trabajo, el autor es un firme impulsor del software libre en lo personal y en lo profesional; esto lleva a que, más allá de únicamente influir las decisiones de software utilizado, y posiblemente contraviniendo tradiciones del área ingenieril, considera las dimensiones éticas y sociales de los principios que transmite a los alumnos — incluso en la elección de herramientas y métodos de colaboración.

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. (Rochkind, 1975)

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* (Tichy, 1985); 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 (Berliner, 1990). 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?” (Reid, 2005)

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 vertigionso 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 (Collins-Sussman, 2004). 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 (Brockmeier, 2012)), en 2002 Linus Torvalds tomó la controvertida decisión de dejar CVS por un SCV *distribuido* y *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 (Henson, 2002). *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 sucitó una discusión acerca de si la funcionalidad que Andrew Tridgell estaba agregando a Linux consistía una violación de la licencia (Barr, 2005), lo cual eventualmente llevó a que Torvalds mismo iniciara el desarrollo de un SCV distribuido, al que llamó *Git*[3].

En el mismo periodo de tiempo se desarrollaron varios otros SCVs distribuidos 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 SCVs distribuidos, sino que ha logrado atraer a una gran cantidad de proyectos que tenían incluso más de veinte años de historia sobre SCVs centralizados (de Alwys, 2009).

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.[4]

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 (Finley, 2011). A la fecha de escritura del presente texto, según la información disponible en el sitio Web, hospeda a más de 68 millones de proyectos.

---

3 *En slang* británico, *Git* se utiliza para denominar a una mala persona; bromeando, Torvalds dice, “soy un bastardo egoísta, por lo que denomino a todos mis proyectos en referencia a mí mismo. Primero Linux, y ahora Git”. (McMillan, 2005)

4 Aunque la cantidad de proyectos hospedados en *SourceForge* es muy grande, incluso en sus años de mayor actividad resultaba tan poco atractivo al uso casual que se ganó el mote de *SourceForget*, “fuentes olvidadas”, por la gran cantidad de proyectos inactivos hospedados.

Ahora, si bien *GitHub* se ha vuelto nodal para el desarrollo de esta impresionante cantidad de proyectos libres, causa una cierta disonancia congnitiva 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).

TRABAJOS RELACIONADOS

El uso de un SCV como mecanismo para la entrega de trabajos en clase no es una idea novedosa ni es la primera vez que se documenta. Hay dos trabajos que apuntan en el mismo sentido que éste; la principal diferencia tanto en el método como en los resultados, estimamos, radica en la década que ha pasado desde su publicación (Reid, 2005; Milentijevic, 2008): Ambos documentan experiencias centradas en el uso de un SCV centralizado.

El trabajo de Reid implementa un repositorio por estudiante, empleando CVS. Para comenzar, esto impone ya un límite que nos resulta determinante: Parte fundamental de los SCV es la colaboración. Si cada alumno tiene un repositorio personal, realizar trabajos en equipo presenta la disyuntiva de si multiplicar los *commits* (requerir que cada uno de los participantes deposite el mismo trabajo en su depósito personal) o indicar de alguna manera al docente que un sólo proyecto *ampara* a varios alumnos. Reid incluso menciona como uno de los puntos de tensión para el uso educativo de CVS a la “necesidad de prevenir que un alumno vea el repositorio del otro”.

Milentijevic relata una experiencia construida sobre una base tecnológica que puede sustentarse ya sea en Subversion o en CVS, pero necesariamente contando con un módulo que presente al conjunto de repositorios sobre una interfaz Web. La separación en múltiples repositorios se efectúa a nivel proyecto: En el cursado de la materia, cada proyecto se presentará a los estudiantes, quienes tendrán que obtener un nuevo repositorio; el ciclo de vida se separa en actividades de inicialización, realización de la tarea, y evaluación de la tarea; la primera de estas etapas es con mucho la que más actividades demanda, lo cual habla del trabajo *burocrático* adicional requerido para éste. Esto, sin embargo, permite la flexibilidad de tareas con equipos de diferentes conformaciones, incluso dándole la tarea de *supervisor* a uno de los alumnos.

La empresa *GitHub* cuenta con un sitio creado bajo un apelativo *comunitario*[5] orientado a generar contacto entre docentes y estudiantes para discutir la tecnología en la educación. La oferta que presentan está estructurada alrededor de la creación de *organizaciones* y la operación dentro de esta de *Salones GitHub*; el flujo de trabajo que presenta es específico a esa configuración, y no refleja la colaboración sobre un flujo de trabajo en el espacio profesional. Dado que es necesario entrar mediante credenciales específicas, y por falta de tiempo para mayor profundización, no nos fue posible encontrar suficiente información para hacer un juicio informado acerca de esta modalidad de trabajo. La mera existencia de la *comunidad GitHub para la educación* es, sin embargo, suficiente para estar seguros de que hay muchas otras experiencias en este sentido.

EN BUSCA DE UN ENTORNO APTO PARA LA MATERIA

La Facultad de Ingeniería ofrece varias instalaciones de sistemas LMS, particularmente el *cluster* llamado EDUCAFI, que cuenta con un Moodle para cada una de

5 https://education.github.community/

las divisiones de la Facultad, administrado por la Unidead de Servicios de Cómputo Académico (UNICA)

La experiencia docente que relatamos inicia en enero de 2013 con el semestre 2013-2[6]. 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 asgnada, 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*[7] para la presentación de fragmentos de código o el *Virtual Programming Lab (VPL)*[8] 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/. La estructura interna que se dio al repositorio se presenta en la Figura 1; en un principio, los tres únicos directorios del repositorio eran los relativos a las entregas de los alumnos (*tareas*, *prácticas* y *proyectos*), con cada una de las entregas ubicada en un subdirectorio numerado dentro de éstos. Agregamos a ésto, además, 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.

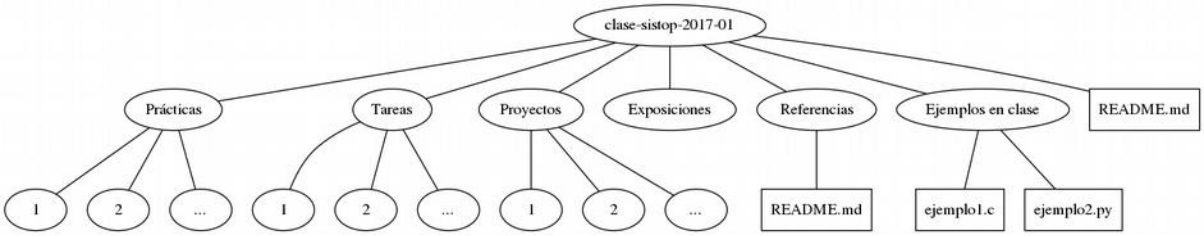

Figura 1. Estructura básica del repositorio *Git* a dos niveles de profundidad; los nodos ovalados indican directorios, los rectangulares indican archivos.

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:

---

6 Los calendarios lectivos anuales en México son de agosto a junio, como es común en el hemisferio norte. La mayor parte de las facultades de la UNAM trabajan en plan semestral, pero numeran sus semestres siguiendo al año que *inicia*, por tanto, el semestre 2013-1 fue de agosto a noviembre de 2012, y el 2013-2, de enero a mayo de 2013.

7 *https://moodle.org/plugins/filter_geshi*

8 *https://moodle.org/plugins/mod_vpl*

> *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 2[9]. Fuera de este punto, la estructura empleada se ha mantenido durante tres iteraciones del cursado.

Dado que los alumnos no están familiarizados con Git, ni con esquema alguno de 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.

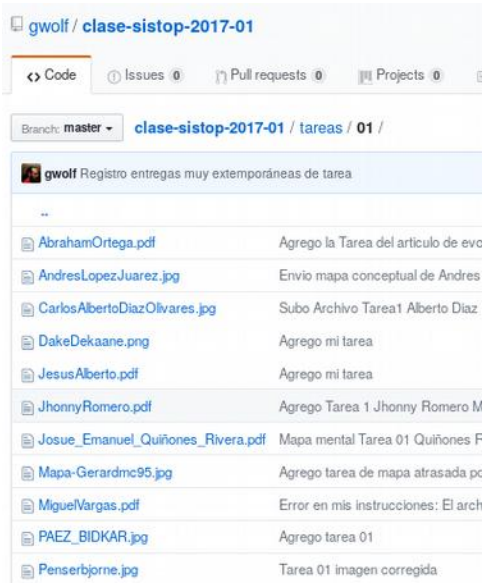

Figura 2. Ejemplo de directorio de entrega de tarea en el primer semestre de aplicación. Nótese que muchos de los alumnos denominaron a sus archivos empleando pseudónimos, no nombres reales, dificultando registrar las respectivas calificaciones.

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* 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*.
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*, un desarrollador puede estar trabajando sobre distintas características de forma independiente. El manejo de ramas

9 Puede consultarse en línea en https://github.com/gwolf/clase-sistop-2017-01/tree/master/tareas/01

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, generados por la compilación o por la ejecución de un programa. Para evitar que los alumnos se confundan con los cambios en dichos archivos, y para llevarlos hacia buenas prácticas del desarrollo colaborativo, se incluye esta práctica.

En las tres 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[10] para explorarla y comprender mejor los diversos eventos que registra.

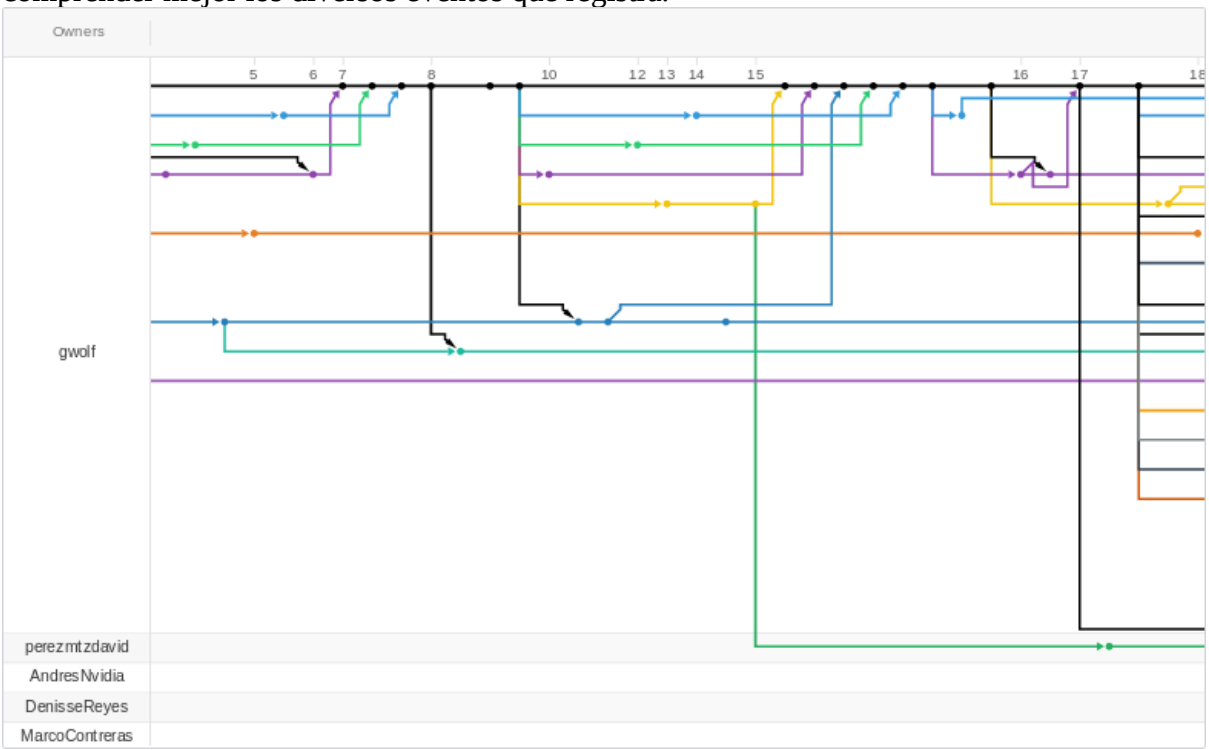

Figura 3. Gráfica de la red de interacciones entre repositorios para el semestre 2017-1. Cada punto representa un *commit*. Pueden observarse patrones que indican fechas de entrega (entre los días 15 y 16), el anuncio de una nueva tarea a realizar (día 17), manejo de líneas paralelas de desarrollo por 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.

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, con uno de los objetivos (si bien es un objetivo adicional y *personal* del docente) enfocado a la familiarización con herramientas de desarrollo colaborativo de software en que se basa el

10 La gráfica de actividad de la red de un repositorio puede encontrarse agregando */network* a la dirección de cualquier repositorio en *GitHub*, por lo que, para las tres experiencias presentadas: https://github.com/gwolf/clase-sistop-2017-01/network, https://github.com/gwolf/sistop-2017-2/network y https://github.com/gwolf/sistop-2018-1/network

software libre — aunque, cada vez más, estas herramientas se van convirtiendo en base del desarrollo de software en general, sin adjetivos.

La experiencia de utilizar a *Git* como base tecnológica, y a *GitHub* como plataforma de colaboración, ha resultado exitosa. No hay medidas objetivas de esto — Es la apreciación personal del autor del presente artículo. Podrían citarse algunas de las evaluaciones de fin de ciclo por parte de distintos alumnos, algunos agradeciendo el uso de Git, otros quejándose del mismo.

De los tres grupos que se han llevado bajo esta forma de trabajo, el primero obtuvo las mejores calificaciones en promedio desde el inicio del trabajo docente del autor, y el segundo las peores; para el segundo de estos grupos, el porcentaje de entregas era tan bajo que se tomó la decisión de modificar el esquema de calificación para no perjudicar a demasiados alumnos. El tercer grupo va, al tiempo de la escritura de este artículo, apenas a la mitad de su cursado, pero se pueden ir observando tendencias favorables.

El primer grupo tuvo 22 participantes (el docente y 21 alumnos; la totalidad de alumnos registrados registraron partiipación, así fuera mínima) que participaron, por lo menos con un *commit*; de los 868 *commits* registrados, restando los 243 efectuados por el docente (que incluyen a varios *merges*), registró un promedio de 28.4 *commits* con una desviación estándar de 26.4 (claramente muy elevada; esto es en buena medida atribuible a un sólo alumno que hizo 105 *commits* para el desarrollo de sus dos proeyctos).

El segundo grupo, con únicamente 13 participantes (el docente y 12 alumnos; desde un principio, 10 de los alumnos no hicieron ni siquiera la primera de las prácticas) registró únicamente 316 *commits*. Esto, si bien se explica parciaplmente por la menor cantidad de alumnos, también es en parte por su menor participación, y por último, por el cambio en la modalidad de evaluación, dado que uno de los proyectos entregables se cambió por un examen. Restando los 138 *commits* del docente, registró un promedio de 9.3 *commits* y una desviación estándar de 8.22.

La experiencia no ha estado libre de problemas; siempre es necesario dar un poco de guía a los alumnos, pero en nuestra experiencia, esto es así independientemente de la base tecnológica, siempre que se utiliza un sistema donde hay interacción en línea. De hecho, algo positivo que puede atribuirse al fuerte énfasis que sirvió como criterio de diseño de *Git* por la verificación de proveniencia, así como la verificabilidad que brinda *GitHub* al presentar una vista Web del repositorio de cada participante por separado, es posible que la cantidad de reclamos de un alumno ante trabajos que digan haber entregado y no fueron recibidos ha disminuído.

Pero, como reflexión final, el principal objetivo de este cambio (que se ha cumplido a la cabalidad) es el de presentar a un grupo de estudiantes una de las herramientas de desarrollo colaborativo que más importantes les serán en el futuro, y que posiblemente más cueste trabajo comprender ante una primera exposición.

REFERENCIAS

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

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

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

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

Clifton, C., Kaczmarczyk, L. C., & Mrozek, M. (2007, March). Subverting the fundamentals sequence: using version control to enhance course management. In *ACM SIGCSE Bulletin* (Vol. 39, No. 1, pp. 86-90). ACM

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

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/

Glassy, L. (2006). Using version control to observe student software development processes. *Journal of Computing Sciences in Colleges*, 21(3), 99-106.

Haaranen, L., & Lehtinen, T. (2015, June). Teaching git on the side: Version control system as a course platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 87-92). ACM.

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

McMillan, R. (2005, abril). *After controversy, Torvalds begins work on "git"*. IDG News Service. Recuperado de https://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/

Milentijevic, I., Ciric, V., & Vojinovic, O. (2008). Version control in project-based learning. *Computers & Education*, 50(4), 1331-1338.

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.

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

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

PERFIL ACADÉMICO Y PROFESIONAL DEL AUTOR

Gunnar Wolf es de formación autodidacta, validada por el acuerdo CENEVAL/SEP como equivalente a licenciatura en ingeniería en sistemas; actualmente cursa la Maestría en Ingeniería en Seguridad Informática y Tecnologías de la Información en el IPN, ESIME Culuhacán. Es Desarrollador del proyecto Debian. Labora como Técnico Académico en el Instituto de Investigaciones Económicas y como Profesor de Asignatura de la Facultad de Ingeniería, ambos de la UNAM. Es apasionado, usuario, promotor y desarrollador de software libre desde hace más de veinte años, sus intereses incluyen la seguridad informática, el software y la cultura libres.
gwolf@gwolf.org
Instituto de Investigaciones Económicas, UNAM
Cto. Mtro. Mario de la Cueva S/N

Ciudad Universitaria, Coyoacán 04510
Ciudad de México (México)

Fechas de recepción y aceptación del artículo

View publication stats