Trabajos de Carreras de Informática en Comunidades de Código Abierto

Rosita Wachenchauzer

Facultad de Ingeniería, UBA y Universidad Nacional de Tres de Febrero
rositaw@gmail.com

Resumen En este trabajo se presentarán algunas modalidades posibles de participación de los estudiantes de informática (tanto de licenciatura como de ingeniería) en las comunidades de software de código abierto y su valor educativo, analizados desde una perspectiva organizacional y de innovación.

Palabras Clave: comunidades de software de código abierto, trabajo final, práctica profesional, educación y capacitación

1. Economía del Software de Código Abierto

Los orígenes del modelo de desarrollo de software de código abierto se pueden situar en los años '50 y '60 del siglo pasado, cuando se crearon los primeros grupos de usuarios de *"mainframes"*, con el objetivo de compartir soluciones a problemas difíciles, incluyendo problemas de software (por ejemplo, en 1955 se constituyó SHARE, un grupo de usuarios de *"mainframes"* de IBM que trabajaban en la industria aeroespacial; y en 1961 se constituyó DECUS, un grupo de usuarios de máquinas de Digital Equipment Corporation). Pero fue en la década de los '90, con la aparición de Linux, cuando el modelo de código abierto comenzó a atraer la atención no sólo de los científicos e ingenieros, sino también de los economistas, ya que, como una manera de producir y distribuir software, tuvo un fuerte impacto en el mundo comercial.

En los párrafos que siguen se analizarán diversos aspectos económicos y organizacionales de las comunidades de software de código abierto.

1.1. Asignación de tareas

Ljunberg (en [7]) describe a las comunidades de código abierto como comunidades de programadores altamente entrenados, basadas en Internet y la Web, que desarrollan colectivamente software cuya calidad es, muy a menudo, muy superior a la del software comercial propietario. El mismo artículo analiza el modelo de desarrollo de software de código abierto usando la metáfora del bazar, que fue introducida por Raymond en [12]: un mercado donde la gente llega y se va, vende, compra e intercambia bienes.

El modelo de desarrollo de software lo analiza a lo largo de cinco dimensiones: economía del don (*gift economy* en inglés) y el intercambio de conocimientos

científicos, procesos de software, diseño participativo guiado por los usuarios, organizaciones virtuales y modelos de negocios, y concluye que el modelo del bazar de los proyectos de código abierto influirá en el futuro de las organizaciones basadas en el conocimiento en términos de gerenciamiento, relaciones con los clientes y modelo de negocios.

Con respecto al modelo de gerenciamiento (y las posibilidades de participación) Ljunberg señala que a menudo existe una jerarquía estricta en dichas comunidades, con un equipo o un dictador benévolo en la cúpula, luego co–desarrolladores en el nivel siguiente, y luego la comunidad de colaboradores. Las mejores contribuciones ganan, y los individuos que las proveen consiguen reconocimiento y reputación por su aporte.

También con respecto a la asignación de tareas, Crowston et al. descubrieron (véase [3]) que la auto–asignación es muy frecuente: los desarrolladores comentan los errores encontrados o los agregados que quisieran realizar, y piden permiso a la comunidad para realizar la tarea.

1.2. Economía del Don

Ljunberg [7] afirma que los fundamentos de la economía del don son la obligación a dar, la obligación a recibir y la obligación a retribuir por los dones recibidos. Se espera, de manera tácita, que en el futuro se retribuya a la comunidad sin ninguna demanda de repago. En una economía del don el estatus social de una persona se determina no por lo que esa persona posee o controla sino por lo es capaz de dar.

En una comunidad de código abierto la única medida disponible de éxito es la reputación entre los pares. Esta reputación se gana dando o compartiendo software de calidad, conocimiento o soluciones a problemas. Una buena reputación es un premio en sí misma, pero también es una manera de atraer la atención de los demás. Esta atención puede servir para tener estatus, oportunidades de trabajo o dinero.

1.3. Código Abierto e Innovación Abierta

West and Gallagher, en [15], definen innovación abierta como la acción de explorar de manera sistemática un amplio conjunto de fuentes internas y externas de innovación y promover la integración conciente de esa innovación con los recursos de la firma y la explotación amplia de esas oportunidades a través de múltiples canales. El paradigma de innovación abierta a menudo se contrasta con el modelo de integración vertical tradicional, en el cual las actividades de I+D interna conducen a productos desarrollados internamente que son luego distribuidos por la firma. Señalan que un ejemplo cada vez más popular de innovación abierta es el software de código abierto, en particular el desarrollo de Linux. Según los autores, el desarrollo de software de código abierto es un claro ejemplo de cómo la innovación abierta puede cambiar una industria, ya que involucra la colaboración entre firmas, proveedores, clientes y constructores de productos relacionados, de manera de aunar I+D en software y producir una

tecnología compartida, mientras que las políticas de propiedad intelectual del código abierto implican la diseminación de esta tecnología compartida sin costo (o con un costo muy bajo).

La estrategia del código abierto como innovación abierta se basa en: los derechos compartidos para el uso de la tecnología y el desarrollo colaborativo de la tecnología. Las firmas manejan un ecosistema complejo para combinar innovaciones internas y externas, creando una arquitectura compleja que al mismo tiempo crea y captura valor.

1.4. Caracterización de las Comunidades

Henri y Pudelko, en [6], afirman que todas las comunidades virtuales son comunidades de aprendizaje, y establecen un marco para el análisis de las comunidades virtuales como comunidades de aprendizaje. Proponen cuatro tipos diferentes de comunidades dependiendo de dos variables: (a) la fuerza del vínculo social y (b) la intencionalidad de reunión. También incluyen, como un resultado, lo que aprende cada individuo al incorporarse a la comunidad. La clasificación que proponen (con cohesión creciente por un lado e intención creciente de reunirse por el otro) es la siguiente:

1. Comunidades de interés: Los miembros se reúnen alrededor de un tópico de interés común. Tienen baja cohesión y baja intencionalidad de reunión. El resultado del aprendizaje es la construcción de conocimiento para uso individual. Como ejemplo proponen los foros virtuales.
2. Comunidades de interés con un objetivo: Los miembros se reúnen alrededor de un tópico de interés común y tienen que conseguir un resultado específico en un tiempo determinado. Tienen mayor cohesión y mayor intencionalidad de reunión que en el caso anterior. El resultado del aprendizaje es la construcción de conocimiento para uso colectivo a partir de sistemas de conocimiento diversos. Como ejemplo proponen las fuerzas de tarea.
3. Comunidades de aprendizaje: Estas comunidades operan en contextos institucionales y están compuestas por estudiantes que dependen de instructores que los guían. La cohesión y la intencionalidad de reunión tienen valores de medio a alto. El resultado del aprendizaje es la construcción de conocimiento mediante la realización de actividades socialmente contextualizadas.
4. Comunidades de práctica: Se organizan alrededor de profesionales que trabajan en organizaciones en las cuales realizan actividades similares. Tienen alta cohesión y alta intencionalidad de reunión. El resultado del aprendizaje es la apropiación de nuevas prácticas y el desarrollo de la participación.

En particular, los autores proponen a las comunidades de código abierto como comunidades de interés con un objetivo. Su meta es desarrollar y mantener un software particular.

1.5. Trabajo Real en una Comunidad Virtual

En 1998 Peter F. Drucker publicó un artículo seminal sobre la estructura de las organizaciones, “Management’s New Paradigms” [4], que considera que

afirmaciones tales como “hay una sola manera de organizar los negocios”, “los principios de gerenciamiento se aplican solo a organizaciones comerciales” y “hay una sola manera de gerenciar a las personas” son (lo eran ya en 1998) obsoletas. Sostiene que el estudio de las organizaciones siempre se basó en la premisa de que hay o debe haber una única “correcta” forma de organización. Lo que se presenta como la “correcta organización” ha cambiado más de una vez, paro siempre continúa la búsqueda del paradigma único de organización. Es claro, según Drucker, que la organización no es un absoluto sino solo un instrumento para que la gente se vuelva productiva trabajando junta. En ese sentido, una cierta estructura organizacional es la adecuada para realizar ciertas tareas, en ciertas condiciones, en cierto momento. Señala que, en particular, en las organizaciones basadas en el conocimiento, incluso los trabajadores a tiempo completo tienen que ser tratados como voluntarios: “Como los voluntarios de las iglesias, los trabajadores del conocimiento poseen sus propios medios de producción: su conocimiento.” Y sigue: “Los trabajadores del conocimiento deben ser tratados como socios, y no solo de palabra. A un socio no se le dan órdenes, se lo persuade”.

Reflexionando sobre las palabras de Drucker, Markus et al. [10] afirman que las empresas de alta tecnología están librando una guerra de talentos. Las compañías buscan captar talentos y energías dispersas en las “comunidades de práctica” y al mismo tiempo ha aumentado el número de trabajadores del conocimiento auto–empleados como *freelancers*. Según estos autores, el movimiento de código abierto es un ejemplo de cuál es la relación que se debe establecer con los empleados. También Crowston et al. [3] citan a Drucker y señalan que los grupos de código abierto son ejemplos extremos de equipos auto–organizados distribuidos, que no son inconsistentes con las nuevas tendencias en el reclutamiento y moitivación de los profesionales, y con el desarrollo de equipos distribuidos.

1.6. Conclusiones

Resumiendo, una comunidad de código abierto se puede categorizar como una comunidad de interés con un objetivo: desarrollar y mantener un software particular, donde se practica la innovación abierta. Sus participantes son parte de la construcción de un conocimiento para uso colectivo, a partir de diversos sistemas de conocimiento. Tienen una estructura jerárquica, pero suele trabajarse mucho por auto–asignación: los desarrolladores comentan los errores encontrados o los agregados que quisieran realizar, y piden permiso a la comunidad para realizar la tarea. Las mejores contribuciones son incorporadas al producto, y los individuos que las proveen consiguen reconocimiento y reputación por su aporte. Este reconocimiento les puede servir para tener estatus, oportunidades de trabajo o dinero. Además, la participación en una comunidad de código abierto brinda una experiencia real de la modalidad de trabajo con la que las empresas de alta tecnología enfocan sus desarrollos.

2. Experiencias de Inmersión en Comunidades de Código Abierto

La participación de estudiantes de carreras de Informática en proyectos de código abierto es entonces una buena idea porque les permite ser parte un grupo de innovación abierta y, si sus contribuciones son aceptadas, atraer la atención de sus pares en la comunidad, lo cual podrá redundar en estatus, oportunidades de trabajo o dinero. Esta participación además les provee la posibilidad de ser parte de la construcción de un conocimiento para uso colectivo, a partir de diversos sistemas de conocimiento. Pero hay más: como afirman en [10] y [3], esta participación les permitirá entrenarse en una modalidad de trabajo que refleja cada vez más la manera como las empresas de alta tecnología enfocan sus desarrollos.

2.1. Algunas experiencias en personales

En la Facultad de Ingeniería de la UBA, los estudiantes de Informática pueden optar por uno de dos modalidades: Trabajo Profesional o Tesis. El Trabajo Profesional es una tarea supervisada de 12 créditos semanales, que puede ser individual o colectiva, en la cual se ataca un problema real con herramientas profesionales, mientras que la Tesis es una tarea individual supervisada de 24 crédito semanales, en la cual se desarrolla una solución original a un problema conocido o aplica una solución conocida a un problema nuevo.

En los siguientes párrafos veremos dos ejemplos de supervisión de trabajos de fin de carrera (un Trabajo Profesional y una Tesis) en comunidades de código abierto y se presentará luego una metodología que permita incluso usar esa inmersión como práctica profesional supervisada.

2.2. Trabajo Profesional: Un Manejador de Experimentos en la Web sobre Condor

Un manejador de experimentos en la Web es una aplicación que permite que los científicos manejen experimentos en ambientes *cluster* o *grid*. La aplicación les permite definir, almacenar, monitorear, controlar, consultar el estado y obtener resultados de sus experimentos a través de una interfaz amigable. Los científicos pueden de esa manera prescindir de los detalles técnicos computacionales en los que, a menudo, preferirían no verse envueltos. El trabajo lo llevó adelante un solo estudiante, Matías Gavinowich.

En la fase inicial se puso el foco en identificar qué aspectos eran mejorables en las herramientas ya existentes, y se armó un listado detallado clasificado por características distintivas, de modo de poder atacar en algún momento algunas de ellas. Para relevar las necesidades de los científicos de la comunidad de computación de alto desempeño, se analizaron las opiniones expresadas por ellos en las listas de discusión de la comunidad. El primer paso fue, por lo tanto, suscribirse a las listas de la comunidad y analizarlas detalladamente. Entre las

necesidades relevadas estaban: soportar aplicaciones con barrido de parámetros (*parameter sweep* en inglés), es decir aplicaciones que se ejecutan varias veces mientras el parámetro de entrada barre un rango de posibilidades, y herramientas para ayudar a realizar experimentos. Se decidió crear una herramienta para manejar experimentos en la Web que encarara esos problemas, y que se construyera sobre Condor (ver [14]), un sistema administrador de cargas para tareas con alta demanda computacional. Condor provee un mecanismo para manejar colas, políticas de planificación, esquema de prioridades, y monitoreo y administración de recursos. Los usuarios envían sus trabajos a Condor, y Condor los encola, elige cuándo y dónde ejecutarlos, monitorea su progreso y finalmente le informa al usuario el resultado. Condor es un proyecto de código abierto cuya licencia se puede encontrar en http://research.cs.wisc.edu/condor/license.html, que se puede usar para administrar un cluster de nodos dedicados (por ejemplo un cluster "Beowulf"). Una de las razones por las cuales se decidió trabajar con Condor fue porque sus usuarios discuten sus problemas en una lista de correo muy activa, condor-users, que parecía muy abierta a los recién llegados.

El diseño y desarrollo se llevaron a cabo siguiendo las buenas prácticas usuales de ingeniería de software. En estas etapas no se involucró a la comunidad, excepto para resolver dudas o consultas puntuales.

Para probar la aplicación, se envió un mensaje a las listas de discusión de computación de alto desempeño, pidiéndole a los científicos que enviaran experimentos con barrido de parámetros. Finalmente, cuando la aplicación estuvo lista, algunos miembros activos de la comunidad Condor aceptaron evaluarla y los comentarios fueron muy elogiosos. El trabajo resultante fue publicado en las 40 JAIIO, en HPC 2009 [5]. Sin embargo, Matías no continuó trabajando dentro de la comunidad Condor.

2.3. Tesis de Grado: Un Recolector de Basura para D

D es un lenguaje de programación diseñado e implementado por Walter Bright, cuya primera versión estable fue anunciada en enero de 2007. En 2008 había dos bibliotecas de tiempo de ejecución para D: Phobos, la creación original de Bright, y Tango, una creación comunitaria. Pese a que Phobos era de código abierto, otros desarrolladores tenían dificultades para contribuir a la plataforma porque Bright no aceptaba cambios, de modo que por mucho tiempo Phobos fue una creación personal. Tango apareció como una plataforma alternativa, creada por algunos distinguidos participantes de la comunidad D (en [1] y [2] se puede ver cómo evolucionaron las diferencias). El tesista, Leandro Lucarella, participó activamente en las listas de discusión de la comunidad. En este caso, no sólo se usó la lista de discusión para recabar información, sino que se volvió un activo participante de estas discusiones (incluso en las discusiones más políticas).

Ambas plataformas tenían prácticamente el mismo recolector de basura, que necesitaba ser reescrito: era muy elemental, ignoraba todos los resultados recientes en el tema e introducía grandes retrasos en la computación. Pese a que Tango no era la biblioteca estándar, decidimos trabajar con esa comunidad por tratarse

de un grupo más abierto y así poder recibir realimentación sobre los resultados. Leandro analizó las fortalezas, debilidades y limitaciones del recolector de basura existente y, con esta información construyó un nuevo recolector (CDGC: D Concurrent Garbage Collector) [8]. Las etapas de análisis, diseño, desarrollo y testeo se realizaron sin involucrar a la comunidad pero, una vez que el artefacto estuvo en funcionamiento, ellos apoyaron el benchmarking del mismo, dado que el objetivo era precisamente reducir la latencia. La comunidad aportó un conjunto de casos de prueba para poder decidir si se conseguía la reducción esperada de la latencia. Los resultados fueron muy buenos: en algunos casos la latencia se redujo en un factor 200. Cuando se considera el tiempo total de la aplicación, la reducción varió entre 17 veces para aplicaciones pequeñas y 3 veces en el caso de una aplicación real grande.

Se pudo comprobar además la afirmación de Ljunberg en [7] respecto la economía del don y los resultados esperados por fuera de esta economía: en este caso surgió una oportunidad laboral, y el flamante graduado está trabajando en una firma (europea) de la comunidad de D. Además, una rama experimental de D2 (la segunda versión de D) incluye a CDGC. Además, recientemente, las comunidades de D volvieron a unirse, y Leandro fue invitado a presentar su recolector en DConf 2013, adonde el propio Bright participó y se interesó por el proyecto (véase el video de la presentación, incluyendo las preguntas de Bright en [9]).

2.4. Conclusiones

De estas dos experiencias se pueden extraer algunas lecciones:

- El primer paso es siempre un buceo en las listas de discusión de la comunidad para encontrar posibles contribuciones.
- Mientras se realiza este buceo, es muy importante verificar si la comunidad está activa (esto se puede controlar fácilmente inspeccionando las nuevas versiones y las listas de discusión).
- Hay que verificar si la comunidad está realmente abierta a las contribuciones externas (la experiencia con Phobos nos mostró que no siempre es así).
- El rol de la comunidad en general será el de proponer el problema, ayudar a resolver cuestiones específicas, facilitar el acceso a benchmarks y discutir (y tal vez aceptar) los resultados, pero los colaboradores tienen mucho trabajo por fuera de la comunidad: el análisis, el diseño, el desarrollo y las pruebas las tienen que hacer ellos, con la guía del tutor.

3. ¿Por qué no Prácticas Profesionales Supervisadas Virtuales?

En 2009, el Ministerio de Educación publicó la Resolución 786/2009 [11] que establece los estándares para la acreditación de los programas de educación superior de grado ligados a la Informática, incluidos los títulos de Ingeniería

(Ingeniería en Computación e Ingeniería en Informática). Para todos los títulos de Ingeniería, la resolución establece que "[d]ebe acreditarse un tiempo mínimo de 200 horas de práctica profesional en sectores productivos y/o de servicios, o bien en proyectos concretos desarrollados por la institución para estos sectores o en cooperación con ellos. Las horas de práctica supervisada deben aplicarse a tareas que contribuyan al perfil del profesional que se está formando. Generalmente corresponderán a actividades pertenecientes al área de Tecnologías Aplicadas."

La lógica detrás de este requisito es que los futuros ingenieros dediquen 200 horas a practicar en un ambiente real de Ingeniería. La propuesta que se hace en este trabajo es que la práctica se pueda hacer también en una comunidad de software de código abierto, pese a no ceñirse estrictamente a la categoría de "sectores productivos o de servicios".

Ya vimos en 1.5 que [10] y [3] ven al estilo de trabajo dentro de las comunidades de código abierto como el ejemplo de la manera como los trabajadores de la sociedad del conocimiento deben ser tratados, y también como un ejemplo de equipos auto–organizados, representativos de la modalidad de trabajo de la industria de alta tecnología más innovadora. Por lo tanto esta experiencia puede no solo ser enriquecedora sino que también será representativa de una modalidad de trabajo en Ingeniería: aquélla que se desarrollará en las empresas más avanzadas.

Por otra parte, la práctica profesional en comunidades de código abierto agrega el valor de aprender a cooperar y a comprometerse socialmente.

3.1. Guía para las Prácticas Profesionales Supervisadas en Comunidades de Código Abierto

Teniendo en cuenta las experiencias descriptas en 2.4, propondremos un formato para que dicho trabajo sea una experiencia productiva para la comunidad y también para los futuros profesionales. Se pone de manifiesto que en esta propuesta, todo el peso de la supervisión recae en los docentes (en el caso de empresas habrá una supervisión mayor por parte de los jefes directos).

1. Asignar tutores (profesores) para supervisar a quienes desee realizar prácticas profesionales en comunidades de código abierto.
2. Seleccionar una comunidad de acuerdo al interés y la formación previa de cada estudiante.
3. Verificar la apertura y actividad de la comunidad.
4. Obtener (por inspección directa o haciendo preguntas) una lista de problemas que la comunidad desea resolver.
5. Seleccionar un tema que pueda ser resuelto en el tiempo disponible (200 horas en este caso).
6. Investigar y diseñar una posible solución.
7. Hacer una propuesta a la comunidad, y encontrar un *sponsor* en ella.
8. Resolver el problema usando un enfoque profesional, supervisado y aprobado por el tutor.

9. Mantener una bitácora para documentar todos los pasos y todo el tiempo consumido (un blog on–line es un buen recurso, porque puede compartirse con el tutor y con el *sponsor*).
10. Involucrarse en la comunidad: participar en las listas de discusión en la medida de lo posible.
11. Cuando los resultados sean aprobados por el tutor, enviárselos al sponsor para su evaluación, y finalmente a la comunidad.
12. Presentar un informe completo con todas las actividades realizadas.

3.2. Una vuelta de tuerca: un modelo para replicar

Una modalidad como la presentada anteriormente no deja de ser individual, falta aún la experiencia colectiva de trabajo y de pertenencia a un equipo.

Tal vez la experiencia más interesante y abarcadora de participación de estudiantes de grado en proyectos de código abierto es UCOSP (Undergaduate Capstone Open Source Project) cuyos detalles se pueden ver en [13]. Se trata de un proyecto que se lleva adelante desde 2009, y en el cual participan universidades de todo Canadá. A lo largo de este tiempo han reunido a más de 300 estudiantes de más de 20 universidades.

Cada año se selecciona un grupo de proyectos (en algunos casos se continúa con proyectos de años anteriores) y a cada uno de esos proyectos se le asigna un equipo de trabajo. Los equipos se arman con estudiantes de varias universidades, los que participan juntos, a través de la Web, en una experiencia de ingeniería de software distribuido, dirigidos por un mentor, que puede provenir tanto de la universidad como de la industria.

Las comunidades en las que han trabajado incluyen, entre otras, a Thunderbird, Ingres y Eclipse. Tienen también el apoyo de Google.

Los estudiantes tienen una reunión inicial con su mentor, para conocerse y planificar las actividades. Luego, el resto de las tareas se desarrollan de manera distribuida.

En las conclusiones del artículo, los autores refieren como resultados positivos no sólo la experiencia adquirida por los estudiantes en términos tecnológicos sino también la participación en un proyecto distribuido y la posibilidad de entrar en contacto con las comunidades que los albergan, con vistas a trabajos futuros.

3.3. Conclusiones

Hay experiencias concretas que indican que la inmersión en las comunidades de código abierto le permiten a los estudiantes de Informática llevar adelante interesantes experiencias en la resolución de problemas reales al mismo tiempo que les posibilitan relacionarse con grupos que pueden llegar a valorar y a premiar su participación. Esa experiencia tiene además el valor agregado de fomentar el compromiso y la cooperación. En ese sentido se han usado en trabajos de fin de carrera.

Los análisis realizados por estudiosos de las organizaciones indican que esas inmersiones brindan además la posibilidad de participar en una organización

sumamente sofisticada, representativa de lo que serán las organizaciones en la sociedad del conocimiento. Este resultado sugiere entonces que se pueden llegar a utilizar también como prácticas profesionales supervisadas, y se propone una metodología de trabajo para las mismas, que aún no ha sido validada, y que valdría la pena discutir.

Sería interesante, además, poner en la agenda con vistas al futuro un posibilidad más amplia: ¿por qué no un UCOSP para la Argentina, que incluyera tanto trabajos profesionales como prácticas profesionales supervisadas?

Referencias

1. Attractive Chaos, Timeline of the D Programming Language (2012) http://attractivechaos.wordpress.com/2012/02/28/timeline-of-the-d-programming-language/ (visitado 12 Mayo 2013).
2. Burrel, T. Standard Library Concerns (Phobos / Tango), digitalmars.D discussion list (18 Feb 2008) http://www.digitalmars.com/d/archives/digitalmars/D/Standard_Library_Concerns_Phobos_Tango_65925.html (visitado 12 Mayo 2013).
3. Crowston, K. et al. Self-organization of teams for free/libre open source software development. Information and Software Technology, Vol 49, No. 6, 564–575 (2007).
4. Drucker, P.F. Management's New Paradigms. Forbes, Vol 162, No. 7, 152–177 (1998) http://www.forbes.com/forbes/1998/1005/6207152a.html (visitado 12 Mayo 2013).
5. Gavinowich, M. Web Experiment Manager: On the Distinctive Features Useful in an Application for Experiment Management. En Contributions to HPC 2009, Jornadas Argentinas de Informática. (Mar del Plata, 24-28 Agosto 2009) http://www.40jaiio.org.ar/node/152.
6. Henri, F. and Pudelko, B. Understanding and analysing activity and learning in virtual communities. Journal of Computer Assisted Learning, Vol 19, No. 4, 474–487 (2003).
7. Ljunberg, J. Open source movements as a model for organising. European Journal of Information Systems. Vol 9, No. 4, 208–216 (2000).
8. Lucarella, L.M. Recolección de Basura en D. Tesis de Grado en Ingeniería en Informática. Departamento de Computación. Facultad de Ingeniería. Universidad de Buenos Aires (2010) http://materias.fi.uba.ar/7500/lucarella-tesisingenieriainformatica.pdf (visitado 12 Mayo 2013).
9. Lucarella, L.M. Concurrent Garbage Collection for D. DConf 2013 (Menlo Park, CA, May 1- 3, 2013) http://dconf.org/2013/talks/lucarella.html (visitado 13 Julio 2013).
10. Markus, M.L., Manville, B. & Agres, C.E. What makes a virtual organization work? Sloan Management Review, Vol 42, No. 1, 13–26 (2000).
11. Ministerio de Educación, Resolución 786/2009. Apruébanse los contenidos curriculares básicos, la carga horaria mínima, los criterios de intensidad de la formación práctica y los estándares para las carreras de Licenciatura en Sistemas –Sistemas de Información– Análisis de Sistemas, Licenciatura en Informática (2009) http://infoleg.mecon.gov.ar/infolegInternet/anexos/150000-154999/154121/norma.htm (visitado 12 Mayo 2013).
12. Raymond, E.S., The Cathedral and the Bazaar. version 3.0 (2002) http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html (visitado 12 Mayo 2013).

13. Stroulia, E. et al. Teaching distributed software engineering with UCOSP: the undergraduate capstone open-source project. In Proceedings of the 2011 community building workshop on Collaborative teaching of globally distributed software development, pp. 20-25. ACM (2011) http://www.academia.edu/561514/Teaching_Distributed_Software_Engineering_with_UCOSP_The_Undergraduate_Capstone_Open-Source_Project (visitado 13 Julio 2013).
14. Tannenbaum, D. W., Miller, K. and Livny, M. Condor - A Distributed Job Scheduler, in Thomas Sterling, ed., Beowulf Cluster Computing with Linux, Cambridge: The MIT Press (2002).
15. West, J. and Gallagher, S. Patterns of Open Innovation in Open Source Software, in Henry Chesbrough, Wim Vanhaverbeke, and Joel West, eds., Open Innovation: Researching a New Paradigm. Oxford: Oxford University Press (2006).