Análisis de la comunicación en las comunidades de software libre

Agustín Ventura Carrasco

1. INTRODUCCIÓN

La eclosión en la década de 1990 de las filosofías relativas al Free and Open Source Software (FOSS[1]) definió y dio uniformidad a un concepto tan antiguo como el mismo desarrollo de software en plena época de expansión de un modelo privativo.
Estas filosofías tienen como parte fundamental y estructuradora que el software es desarrollado por una comunidad. Frente a las propuestas comerciales en las que el software es desarrollado por una empresa y vendido a unos clientes, el FOSS propone que los usuarios y los desarrolladores forman una única comunidad que desarrolla el software de manera conjunta.
Al desdibujarse las líneas que separan a los usuarios de los desarrolladores, tan sólo cabe hablar de usuarios y sus distintos roles, ya que habitualmente los desarrolladores de un proyecto FOSS son sus primeros usuarios. Tradicionalmente se ha defendido que en la comunidad FOSS son tan importantes los desarrolladores como otro tipo de usuarios, ya que se espera de estos una variedad de tareas que permiten enriquecer el proyecto en conjunto como son la documentación (guías de inicio, tutoriales, artículos), el reporte de errores (a través de listas de correos o de gestores de defectos) o simplemente la retroalimentación a través de opiniones.
Tal y como explica Eric S. Raymond en su libro “La catedral y el bazar” [1] el desarrollo de software libre ha seguido tradicionalmente dos modelos de gestión, el modelo de la catedral o el modelo del bazar. En el de la catedral, como puede ser el desarrollo de Emacs[2] o GCC[3], el código se libera en cada versión, pero entre versión y versión tan sólo un grupo de desarrolladores tiene acceso a los fuentes.
En el modelo del bazar, el desarrollo del software se produce en la esfera pública, siendo accesible a todos los interesados en todo momento y pudiendo estos colaborar libremente. Un ejemplo de este modelo de desarrollo sería GNU/Linux[4].
Con el paso de la década de los 2000 el modelo del bazar se constituyó como el referente en el desarrollo FOSS ya que si se desea conformar una comunidad y atraer contribuciones resulta más indicado que el de catedral, que es más distante al usuario.
En la research proposal "Very Large Community-based Software Development" [2] se analizan varios proyectos de GitHub y se comparan con lo propuesto en “La catedral y el bazar”, mostrando resultados bastante decepcionantes. Los proyectos analizados están más próximos al modelo de catedral que al de bazar. Es patente la falta de democracia, la inexistencia de unas reglas de gobernancia claras y el bajo número de usuarios involucrados en los proyectos, incluso en aquellos más populares [3]. Parece

[1] https://en.wikipedia.org/wiki/Free_and_open-source_software
[2] https://www.gnu.org/software/emacs/
[3] https://gcc.gnu.org/
[4] http://www.linux.org/

entonces que se debe cambiar el enfoque de los proyectos, de centrarse en el código a centrarse en las personas.
Una forma de evaluar este cambio de enfoque es analizar la comunicación en los proyectos. En comunidades más abiertas se espera un modelo deliberativo [4] y horizontal, mientras que en comunidades más cerradas o estratificadas nos encontraremos con un modelo más vertical y más centrado en una persona o un grupo de personas (los administradores).
Proponemos por tanto establecer un marco de referencia, una serie de indicadores que permitan medir de manera sencilla el estado de salud comunicativo de un proyecto para poder estudiar su evolución, compararlo con otros proyectos y evaluar la eficacia de medidas tomadas para aumentar la calidad de la comunicación dentro de la comunidad del proyecto.
En resumen, los objetivos de este proyecto son:

- Estudiar la comunicación en los proyectos alojados en GitHub.
- Ampliar la herramienta Gitana ([5]) para poder acceder a las incidencias y eventos relacionados en GitHub y almacenarlos.
- Definir y explotar métricas sobre esta información para evaluar la comunicación en el proyecto.

El resto del presente documento está organizado de la siguiente manera:

- En la sección 2, Antecedentes, se introduce el estado actual del desarrollo de software libre en GitHub.
- En la sección 3, Estado del arte, se revisan diversos estudios realizados utilizando los datos de GitHub y se analiza el marco teórico para el análisis de la comunicación.
- En la sección 4, Diseño, se detalla el diseño de la herramienta propuesta para extraer la información de GitHub.
- En la sección 5, Implementación, se presentan los aspectos más relevantes de la implementación de la herramienta.
- En la sección 6, Indicadores, se proponen los indicadores basados en el marco teórico que servirán para evaluar la comunicación en los proyectos.
- En la sección 7, Aplicación, se muestran los indicadores para dos proyectos y se comparan.
- En la sección 8, Conclusión, se recapitula sobre el trabajo realizado y se presentan futuros trabajos a realizar.
- En la sección 9, Bibliografía, se presentan los materiales referenciados a lo largo del documento.

2. ANTECEDENTES

Una parte fundamental dentro de las tareas propias del desarrollo de aplicaciones es el control de versiones. Un adecuado sistema de control de versiones permite que el equipo de desarrollo disponga siempre de la última versión de código desarrollado, aplicar sus cambios sobre esa versión y guardar un histórico de los cambios realizados. El sistema de control de versiones es un punto de sincronización de todo el equipo que además guarda un histórico y por tanto ofrece una oportunidad singular en cuanto al estudio y caracterización del mismo. Esta sincronización se puede realizar de dos formas, mediante sistemas centralizados (CVCS) como CVS o Subversion que implican la existencia de una copia maestra y unos permisos de acceso a ella o mediante sistemas distribuidos (DVCS) como Git o Mercurial, en los que no hay copia maestra y los usuarios intercambian las modificaciones.

GitHub[5] es una plataforma que permite alojar repositorios Git (y Subversion) y que además de las habituales facilidades ya descritas, aporta funcionalidades de red social. Los usuarios pueden *seguir* a otros usuarios (y ser informados de la actividad de estos usuarios), pueden *observar* proyectos (y ser informados de las novedades en estos proyectos) y pueden *marcar con una estrella* proyectos (sería el equivalente al *me gusta* en Facebook). Además los usuarios pueden hacer un *fork* de un proyecto (crear una copia de un proyecto para realizar modificaciones) y pueden solicitar que se incluyan sus modificaciones en un proyecto a través de un *pull request*, que un miembro del proyecto aceptará o rechazará.

Resumiendo, el principal cambio de enfoque de GitHub a plataformas anteriores es que permite que cualquier usuario contribuya activamente al proyecto, no sólo los miembros del mismo.

El efecto de la plataforma en sí en el desarrollo de proyectos, si bien no ha sido revolucionario sí que ha contribuido a aumentar la transparencia en los proyectos open source (cualquier usuario puede ver si las colaboraciones se aceptan, las discusiones en torno a ellas y los criterios de inclusión) y además se comprueba un interesante efecto colateral, la no monopolización del código, es decir, se fomenta la participación equitativa entre todos los desarrolladores del proyecto a la base de código [6].

En la actualidad GitHub es el mayor centro de alojamiento y desarrollo de proyectos de software libre y en él se puede encontrar proyectos tan diversos como el kernel de Linux[6], herramientas de desarrollo de Disney[7], frameworks de desarrollo como Ruby on Rails[8] o libros gratuitos[9].

[5] https://github.com/about
[6] https://github.com/torvalds/linux
[7] https://github.com/disney
[8] https://github.com/rails/rails

GitHub tiene otra particularidad que lo hace especialmente atractivo para su estudio, una API[10] pública que permite a cualquier usuario acceder a los datos de los proyectos y usuarios de forma automática y en formato JSON para poder ser procesados y explotados automáticamente por parte de otras aplicaciones.
Gitana [5] es una herramienta de código abierto que nos permite inspeccionar en profundidad repositorios git, analizando los cambios realizados en el proyecto (incluso al nivel de línea dentro de un archivo), sus autores, branches y tags.
Para ello vuelca esta información en una base de datos relacional para facilitar el análisis de los datos como pueden ser: quienes son los usuarios que más colaboran, cuales son los archivos con más modificaciones, cuantos archivos tiene de media un branch o cada cuanto tiempo se crea un tag.
Al ser software libre y estar diseñado para su extensión, se propone la implementación de una extensión para una vez analizado un repositorio alojado en GitHub, analizar sus incidencias, autores y comentarios e incorporar esta información a los datos ya extraídos.
Con esta información sobre las incidencias en la base de datos de Gitana [5] podremos implementar unos indicadores que nos informarán del estado de la comunicación en el proyecto.

3. ESTADO DEL ARTE

La apertura de la API de GitHub ha abierto una ventana en cuanto a la realización de estudios sobre los proyectos y el código, ya que permite fácilmente analizar el código fuente y otros factores relativos a los proyectos como las relaciones entre los usuarios, la gestión de incidencias en el proyecto o el uso de herramientas para la división del trabajo como las menciones y las etiquetas.
En esta sección examinamos distintos estudios que ahondan en las relaciones sociales existentes en GitHub y estudiadas a través de la API y en los estudios existentes centrados en el uso del gestor de incidencias de GitHub y sus características por parte de los usuarios y por último repasamos un estudio en el que se propone un marco para el desarrollo de la actividad deliberativa.

3.1 El componente social en GitHub

Tsay, Dabbish y Herbleb [7] relacionan los usuarios con sus contribuciones al proyecto para responder a una pregunta fundamental dentro del FOSS, ¿son realmente las comunidades una meritocracia? El análisis de repositorios y pull requests les permite establecer una serie de criterios en función de los cuales son más o menos aceptados

[9] https://github.com/vhf/free-programming-books
[10] https://developer.github.com/v3/

los pull requests, que incluyen la corrección técnica (como tener tests) o que el usuario pertenezca al círculo social del proyecto, poniendo en tela de juicio la caracterización meritocrática tradicional de las comunidades open source. Hay que tener en cuenta que previamente, tan sólo los usuarios con permisos de *commit* podían contribuir a los proyectos (como en Google Code), por lo que no se podía evaluar la contribución de usuarios externos ya que por definición todas las contribuciones eran de usuarios internos. En función de estos resultados y si bien parece que las contribuciones de usuarios externos siguen siendo limitadas, sí que resultaría interesante analizar si los usuarios influyen en el proyecto a través de otras vías, como puede ser la comunicación y discusión de incidencias, aunque no hagan contribuciones técnicas directas.

Resulta interesante como los datos que ofrecen los estudios permiten realizar una fotografía mucho más certera sobre el perfil de los usuarios y desarrolladores de proyectos open source y como la comunidad no es tan abierta ni meritocrática como habitualmente se ha asumido, sino que hay una fuerte componente social.

Lima, Rossi y Musolesi [8] convierten en un grafo las relaciones de *follow* entre usuarios y la relación entre usuarios y proyectos para analizarlos y descubrir características únicas de GitHub como que la relación de *follow* no suele ser recíproca, que el tener mucha actividad no implica tener muchos seguidores o que los usuarios suelen interactuar en grupos de proximidad geográfica. Podemos asumir que según este estudio los usuarios que colaboran en un mismo proyecto es muy probable que se conozcan en persona y hasta tengan algún tipo de relación, aunque en este caso resulta sorprendente que las relaciones de *follow* no sean recíprocas. Una posible explicación sería que estas relaciones indiquen una posible jerarquía social. En base a este estudio tenemos que considerar que posiblemente haya otras formas de comunicación implícitas en un proyecto, como pueden ser contactos directos entre desarrolladores a través de email o de chat. Sin embargo no podemos afirmar nada en cuanto a usuarios no involucrados en el desarrollo del proyecto como aquellos que únicamente notifican incidencias.

Otra forma de análisis es considerar que dos usuarios se encuentran conectados cuando comparten un proyecto y dos proyectos cuando comparten un usuario. Este enfoque es el tomado por Thung, Lo y Xiang [9] y según los grafos resultantes GitHub resulta ser una red social especialmente densa, donde el camino más corto medio entre dos desarrolladores es de 2,47 frente al camino más corto en Facebook es de 5. El posible motivo es que al ser una red técnica sus usuarios tienen más contacto. Esta alta densidad puede indicar una fuerte endogamia en la participación en el proyecto que merece la pena ser estudiada a nivel comunicativo.

3.2 El gestor de incidencias de GitHub

Entre otras facilidades, GitHub ofrece un gestor de incidencias integrado al crear un proyecto en la plataforma. El análisis de este gestor de incidencias puede aportarnos información valiosísima sobre la estructura comunicativa entre la comunidad de desarrolladores y usuarios del proyecto. Si bien existen otros medios de comunicación en muchos proyectos (como foros o listas de correos), la funcionalidad de un gestor de incidencias es particularmente relevante ya que será usado por aquellos usuarios más implicados en la herramienta, tanto desarrolladores como usuarios avanzados.
Sin embargo, en su estudio sobre el uso de gestores de incidencias en GitHub, Bissyandé, Lo, Jiang et al [10] indican que el uso del gestor de incidencia integrado es bastante limitado, siendo utilizado por tan sólo el treinta y tres por ciento de los proyectos y tan sólo el ocho por ciento tienen más de cien incidencias detectadas. Además parece que el uso es bastante interno, ya que los desarrolladores suelen ser los principales informadores. Por otra parte sí que aprecian una relación entre el número de *forks* y *watches* del proyecto y el número de incidencias registrado, es decir, aquellos proyectos con una comunidad más activa (y por tanto más interesantes de estudiar) sí que utilizan más frecuentemente el gestor de incidencias.
Diversos estudios se centran en el uso de este gestor de incidencias, que si bien es poco usado, resulta de gran interés cuando se utiliza. Kikas, Dumas y Pfahl [11] utilizan la evolución de incidencias registradas en cuatro mil proyectos de GitHub a lo largo de 3 años para extraer conclusiones sobre el comportamiento de las mismas. Es interesante comprobar que conforme los proyectos envejecen, se reduce el número de nuevas incidencias abiertas (quizás por madurez técnica o al cerrarse el proyecto e ir incorporando menos funciones), pero crecen las incidencias no resueltas de manera lineal en el tiempo (es decir, se van acumulando). El tiempo de vida de las incidencias que sí se resuelven se mantiene bastante establece en el tiempo. Resumiendo, a lo largo de la vida de un proyecto, cada vez se abren menos incidencias y las que se abren en general se resuelven igual de rápido. Por otra parte siempre queda un remanente de incidencias no resueltas. A un nivel comunicativo, resulta muy interesante este remanente de incidencias no resueltas, ¿son foco de debate?¿vuelven a colaborar los usuarios implicados en ellas?
Hay funcionalidades del gestor de incidencia que parece que afectan positivamente a la gestión de la comunidad y a la comunicación como es categorizar las incidencias mediante el uso de etiquetas (*labels*). Cabot, Cánovas et al [12] estudian utilizando el conjunto de datos ofrecido por GHTorrent[11] la influencia de etiquetar las incidencias. En primer lugar sus resultados son coincidentes con los de Bissyandé, Lo, Jiang et al [10],

[11] http://ghtorrent.org/

de los casi cuatro millones de proyectos estudiados, tan sólo el 3,25% utilizan etiquetas en issues. En cuanto al uso de etiquetas, el 45% de los proyectos usan tan sólo una y el 25% dos. En los proyectos en los que se usan etiquetas, se suelen etiquetar el 60% de las issues y se pueden establecer “familias” de etiquetas que tienden a ser usadas en conjunto. A pesar de estos datos, el uso de etiquetas es muy positivo para los proyectos, en general las incidencias con etiquetas suelen implicar a más usuarios y se resuelven más rápidamente.
Otra característica positiva en cuanto a la comunicación en el proyecto es el uso de la @ para mencionar usuarios (y estos reciben una notificación al respecto). Zhang, Wan y Yu [13] estudian el uso que se hace de estas menciones en Ruby on Rails como primer paso para hacer una caracterización más genérica. Sus resultados son que se usan ampliamente las menciones (en el 52% de las incidencias), fundamentalmente en los comentarios de las mismas y para recabar la atención de otros desarrolladores. Las incidencias con menciones suelen tener más comentarios y más usuarios implicados, aunque suelen tomar más tiempo para su resolución, lo cual se puede relacionar con una mayor dificultad. Además, aquellas incidencias con menciones suelen hacer un mayor uso de etiquetas y números de versiones (*milestones*). Por último los autores también realizan un ranking de los usuarios del proyecto según el número de menciones, lo cual abre la puerta a establecer grafos entre ellos.
Es interesante ver como dos características del gestor de incidencia influyen directamente en la comunicación del proyecto al menos a un nivel cuantitativo, ya que tanto el uso de etiquetas como el de menciones aumenta la interacción de usuarios. Podemos interpretar que el gestor de incidencias del proyecto es el primer frente de debate relativo al proyecto. En él habitualmente se discuten los errores que se producen y las posibles mejoras. Esta actividad comunicativa es además facilitada por la estructura del gestor de incidencias de GitHub que permite añadir comentarios a las issues, adjuntos, etiquetar o hacer referencias a otras issues, creando una red comunicativa entre todos los usuarios.

3.3 La actividad deliberativa

Desde un punto de vista comunicativo resulta interesante el estudio de Friess y Eilders [4] en el que proponen un marco para evaluar la actividad deliberativa de un medio de discusión online.
La actividad deliberativa sería la forma de comunicación idónea dentro de un proyecto de software libre ya que según los autores se define como una forma de comunicación racional, interactiva y respetuosa. También se fomenta el que los usuarios estén dispuestos a escuchar los argumentos opuestos y a ceder en sus propias opiniones,

todo ello cualidades deseables no solo comunicativamente sino como parte de un proyecto de ingeniería.
En el marco propuesto se establecen tres niveles con una serie de características determinadas que podemos aplicar a GitHub.

Nivel de entrada institucional

Son las condiciones que se deben dar para fomentar una comunicación deliberativa.
Tiene las siguientes características:

- Inclusividad: El gestor de incidencias de GitHub es abierto, así que cualquiera que se sienta afectado puede participar.
- Igualdad: Todos los usuarios argumentan en igualdad de condiciones, no hay opiniones más visibles ni discriminaciones según sea miembro del proyecto o no.
- Apertura: El gestor es accesible a cualquiera registrado, así que el debate es público.
- Comunicación asíncrona: Al ser un formato foro, la comunicación es asíncrona a diferencia de los chats.
- Visibilidad del contenido: Las incidencias y comentarios se publican instantáneamente para todos los usuarios así que todo el contenido es fácilmente visible.
- Moderación: Hay un moderador que es el propietario del proyecto, de él depende por tanto la actividad de moderación y si desea ser un censor o fomentar el debate.
- Identidad: Todos los usuarios están claramente identificados y hablan por sí mismos.
- Información: El formato de las incidencias y los comentarios permite incluir fácilmente referencias externas o referencias a otras incidencias.
- División de tareas: El sistema de etiquetado de las incidencias, usuarios asignados y seguimiento de las incidencias permite dividir eficientemente el trabajo y que los usuarios sigan las que más les interesan.
- Interacción horizontal: La interacción entre los usuarios es completamente horizontal, pudiendo interaccionar los unos con los otros sin límite alguno.
- Poder percibido: El poder percibido por los usuarios en cuanto al cambio del proyecto debe ser amplio, ya que se pueden comunicar directamente con el equipo de desarrollo.

Nivel de rendimiento comunicativo

Este nivel establece las condiciones para que se pueda desarrollar una auténtica actividad deliberativa. Mientras que en el nivel anterior las condiciones podían venir

más o menos impuestas o fomentadas por GitHub, en este caso nos encontramos con una serie de condiciones que dependerán completamente de los usuarios:

- Racionalidad: El debate debe ser racional y los miembros deben atender a razones.
- Interactividad: Los usuarios deben atender los comentarios de los otros y responderlos.
- Civismo: Los usuarios deben reconocerse mutuamente y respetarse.
- Constructividad: El objetivo final de los usuarios debe ser alcanzar un acuerdo.

A este nivel, mientras que la interactividad si es aportada y fomentada por el gestor de incidencias, el resto de condiciones son características propias de la comunidad del proyecto.

Nivel de resultado productivo

Son los beneficios esperados de la actividad deliberativa. De nuevo dependerán de la comunidad del proyecto ya que técnicamente no hay mucha influencia posible.

- Tolerancia: la actividad de debate público debe aumentar las tolerancia de los usuarios.
- Eficacia: El discutir las incidencias o mejoras del proyecto a través de un sistema con estas características puede redundar en un mejor diseño e implementación y por tanto, no solo en una mejora a nivel de comunidad sino también técnica.
- Conocimientos: El debate público permite que todos los usuarios aumenten sus conocimientos sobre el proyecto en general.
- Espíritu público: La involucración de todos los usuarios a un nivel horizontal y el debate racional y con referencias externas así como la implementación de las acciones acordadas aumentan el espíritu público y de comunidad de los usuarios.
- Aceptación: Es maś sencillo que las decisiones sean aceptadas por los usuarios cuando han sido deliberadas.
- Transformación de preferencias y consenso: La deliberación puede y debe producir una transformación en las preferencias de los usuarios que redunde en un mayor consenso en el seno de la comunidad.
- Legitimidad: Todas las condiciones anteriores aumentan la legitimidad de las decisiones tomadas para la gestión del proyecto.

Podemos considerar este marco como un punto de partida para evaluar el nivel de comunicación deliberativa existente en un proyecto y si este nivel redunda en una mejor gestión de la comunidad.

4. DISEÑO

El principal objetivo a nivel de diseño es ser capaz de leer las incidencias y todas sus dependencias desde GitHub e incorporarlas a la base de datos de Gitana [5] para posteriormente poder explotarla.
Este proceso consta de tres fases bien diferenciadas, tal y como se muestra en la Figura 1.

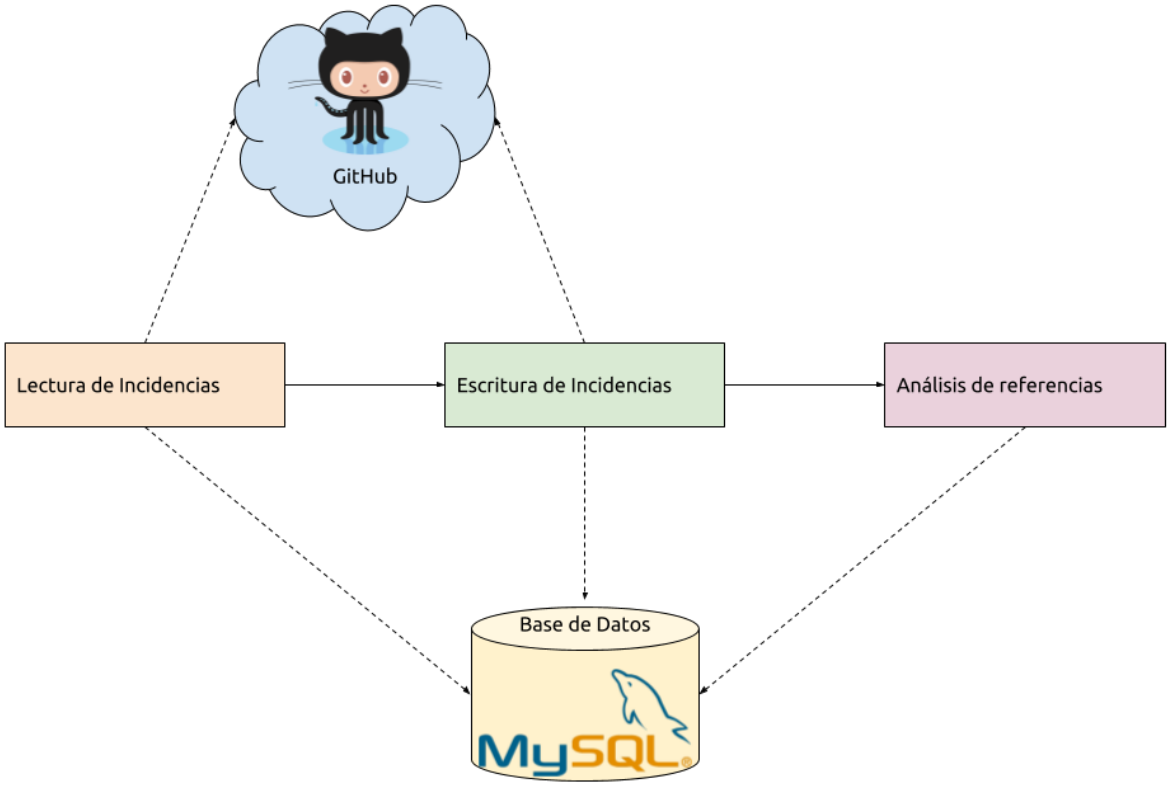

Figura 1. Estructura del proceso de la aplicación

A continuación se describen cada uno de los componentes de este proceso desde un punto de vista de diseño del mismo.

4.1 Base de Datos

El esquema de base de datos que modela las incidencias y sus entidades relacionadas y a su vez engarza con el propuesto por Gitana [5].

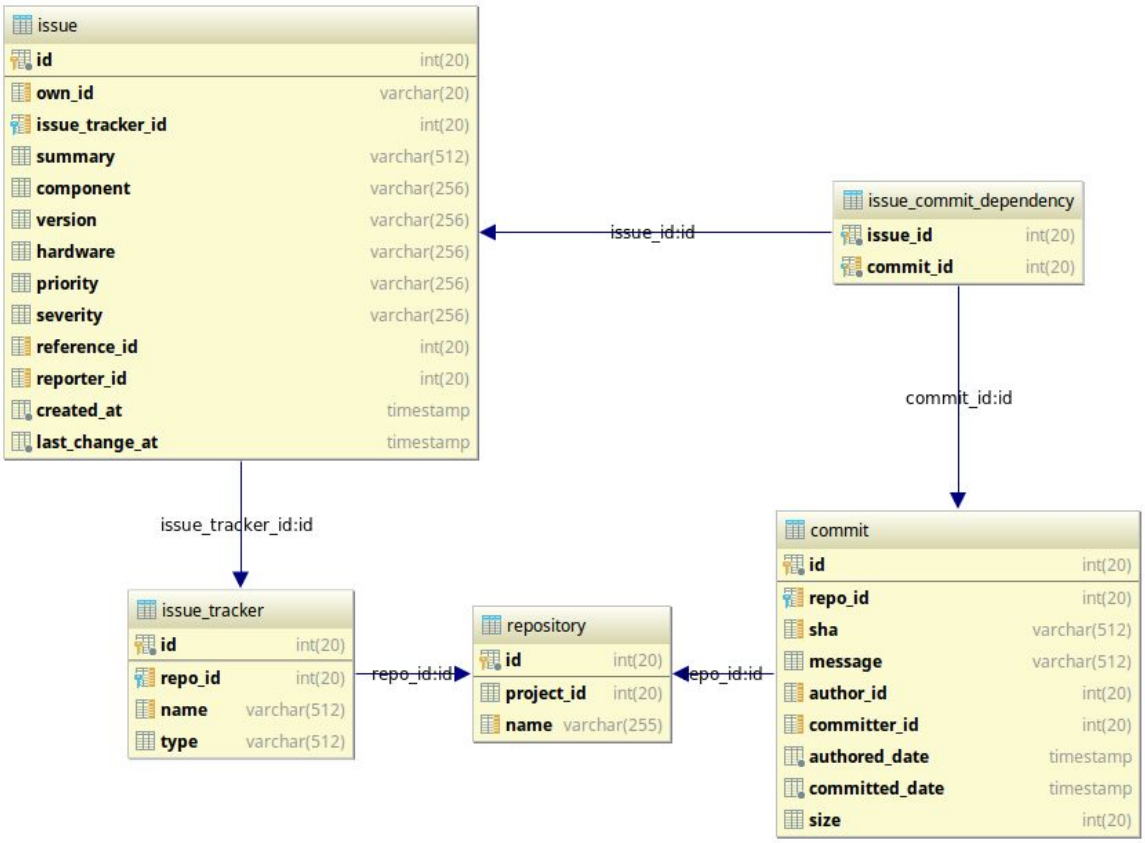

Figura 2. Base De Datos. Incidencias, repositorios y commits

Tal y como se muestra en la Figura 2, un repositorio de git (repository) puede tener varios gestores de incidencias (issue_tracker) asociados y a su vez éstos tienen varias incidencias (issue). Un gestor de incidencias también puede tener commits (commit) y una incidencia puede referenciar un commit (issue_commit_dependency), con lo cual queda establecida la relación fundamental entre incidencia, repositorio y commit.

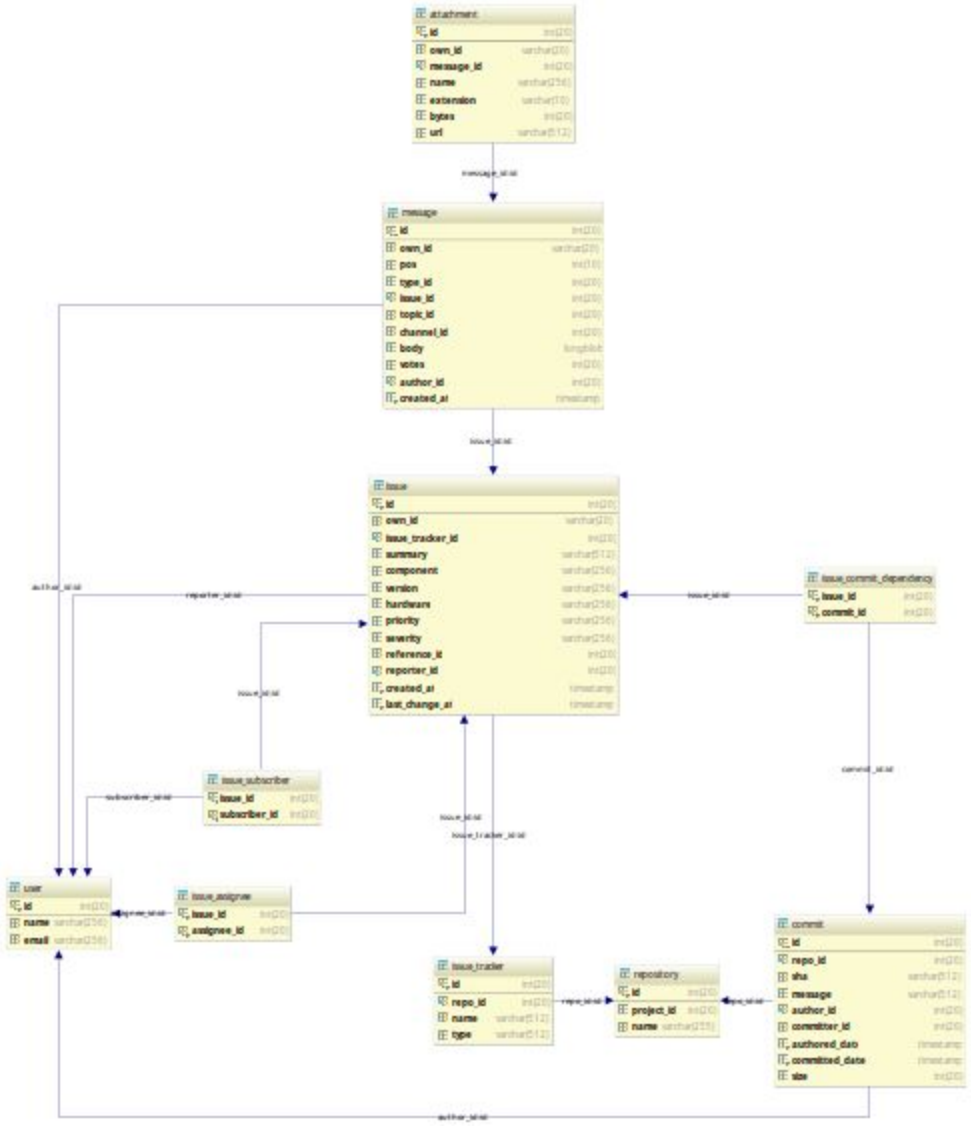

Figura 3. Base De Datos. Incidencias, usuarios y mensajes

En la figura 3 se muestra la relación de la incidencia con los usuarios. Bien como asignados (issue_assignees) a ella bien como suscritos (issue_subscriber) para ser notificados de las novedades. Igualmente se muestra como un usuario puede ser el autor de varios commits y de varias incidencias.
Por otra parte, toda la información textual sobre la incidencia se guarda en la forma de mensajes (message) debidamente relacionados con su autor, así que toda incidencia tiene al menos un mensaje correspondiente al texto de la incidencia y puede tener varios comentarios asociados con sus correspondientes autores. A su vez, un mensaje puede tener adjuntos (attachment) que contribuyen a enriquecer el discurso.

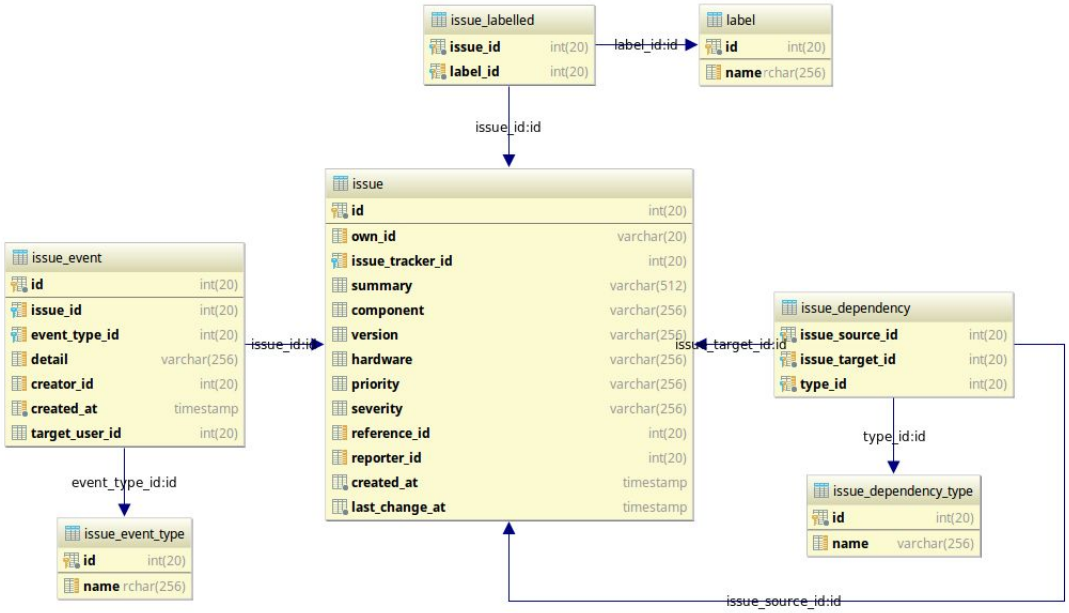

Figura 4. Base De Datos. Incidencias, etiquetas, eventos y dependencias

En la figura 4 mostramos como una incidencia puede contar con etiquetas (label e issue_label) para su categorización y también tiene una serie de eventos (issue_event e issue_event_type) que describen su historia (si se ha cerrado, si se ha mencionado a un usuario, etiquetas añadidas y retiradas…)
Por último, los usuarios pueden referenciar una incidencia en otra y ésto es recogido mediante una dependencia (issue_dependency) que tiene una incidencia origen de la referencia y otra destino.

4.2 Acceso a GitHub

El acceso a la API de GitHub se realiza a través de tokens. Estos tokens son una cadena alfanumérica relacionada con un usuario y que puede ser creada y revocada por el mismo usuario. Por cada conjunto de token y usuario GitHub permite seis mil peticiones a la hora a su API. Gitana [5] permite trabajar con cuantos tokens se deseen añadir, utilizándolos para acceder en paralelo y por tanto acelerar el proceso de lectura. Por tanto se debe contar con al menos un token, pero es recomendable utilizar cuantos sean posibles.

En caso de que algún token alcance su límite de lecturas en el transcurso de una ejecución, el proceso esperará automáticamente una hora para volver a tener disponible todas las peticiones.
Todo el acceso a la API de GitHub se ha centralizado en una clase que devuelve la información requerida en forma de diccionario de datos con los campos oportunos, así una incidencia es devuelta como un conjunto de pares clave, valor que contienen su autor, identificador y resto de información.

4.2 Lectura de incidencias

El proceso de lectura de incidencias funciona de dos formas, bien lee las posibles nuevas incidencias en GitHub y las incorpora a la base de datos de Gitana [5], bien actualiza la información de las que ya hay en la base de datos para incorporar nuevos comentarios, eventos, y en general cualquier tipo de modificación que se haya realizado desde el último análisis.
El acceso a incidencias a través de la API de GitHub se realiza a través de páginas de incidencias, conteniendo hasta treinta incidencias cada página. La primera etapa, el proceso de lectura consiste en utilizar todos los tokens proporcionados a la herramienta para acceder a estas páginas y confeccionar una colección de todas las incidencias presentes en el sistema.
Con este conjunto como base se determinarán bien las incidencias ya analizadas y que se desean actualizar, bien las que no estan presentes en la base de datos y que se desean incorporar.

4.3 Escritura de incidencias

La escritura de incidencias tomará el conjunto de incidencias a incorporar en la base de datos (las nuevas o las que se desean actualizar) y las tratará en paralelo según el número de procesos especificados en la herramienta (por defecto cinco).
El proceso consiste en dividir todas las incidencias a tratar entre el número de procesos que se desean utilizar y asignar a cada proceso uno de estos paquetes de incidencias. Posteriormente cada proceso leerá una por una las incidencias asignadas y las irá introduciendo en la base de datos. Si el proceso a realizar es una actualización, previamente se borrará la información ya presente y se reinsertará la incidencia al completo.
Por cada incidencia el proceso escribe su autor (en caso de no existir previamente), sus datos propios (id asignado en GitHub, fecha de creación, milestone, etc…), sus etiquetas en caso de tenerlas, sus comentarios junto con sus autores, todos los eventos asociados a la incidencia (generando además una descripción en lenguaje natural para cada uno de ellos) y por último, los usuarios asignados si los tiene.

Un punto fundamental de la escritura de incidencias es el reconocimiento del autor de la incidencia. En la aplicación estamos utilizando dos fuentes de datos distintas, por un lado el repositorio git y por otro lado GitHub, un riesgo inherente es que un usuario tenga distintas credenciales en ambos sistemas. Para atenuarlo, si el autor de una incidencia no consta en la base de datos según su login o su email de GitHub, se cargarán los commits de ese usuario en el repositorio de GitHub y se buscarán esos commits en la base de datos local junto con su autor. Así, si el autor es un desarrollador quedará identificado en función de sus credenciales en el repositorio de git.

4.4 Análisis de referencias

Al utilizar paralelismo, no podemos garantizar el orden de inserción de las incidencias en la base de datos, pudiendo insertarse primero las más recientes y al final las más antiguas, por lo tanto, al escribir una incidencia no podemos buscar si hace referencia a otras ya que se puede dar el caso de que las incidencias referenciadas aún no estén en la base de datos. Esto hace necesario una tercera etapa, el análisis de las referencias entre las incidencias.

Este último paso consiste en, una vez presente toda la información de GitHub en la base de datos local, leer todos los mensajes de las incidencias de la base de datos y analizarlos para encontrar referencias entre las incidencias que serán escritas en la base de datos. Esta etapa al ser realizada sin acceder a la API de GitHub es mucho más rápida que las anteriores, normalmente tardando del orden de segundos.

5. IMPLEMENTACIÓN

Dado que la aplicación es una extensión de Gitana [5], se utilizan las mismas tecnologías que usa Gitana [5], en concreto Python y MySQL.
El uso de Python permite una fácil portabilidad entre sistemas operativos, una rápida curva de aprendizaje y una fácil disposición de recursos de todo tipo, desde librerías hasta tutoriales. En concreto se puede destacar la madurez del acceso a MySQL y de la gestión de procesos para el paralelismo.
Se plantean dos posibilidades para el acceso a la API de GitHub, bien acceder realizando las peticiones a bajo nivel, a través del cliente HTTP y tratando las respuestas JSON para extraer la información deseada o bien utilizar una librería ya existente, PyGitHub[12].
Por simplicidad se opta por utilizar PyGitHub. El uso de esta librería comporta dos grandes ventajas:

1. Aporta un modelo de GitHub orientado a objetos que construye automáticamente a partir de las peticiones. Es decir, se puede acceder a un repositorio y devuelve un objeto de tipo Repository que tiene entre otros atributos, un owner de tipo NamedUser. Este modelo se parece bastante a nuestro modelo relacional, teniendo entre otros objetos Issue, Label o IssueComment.
2. El acceso a las propiedades de los objetos es perezoso. El diseño de PyGitHub hace que los objetos se completen inicialmente con los datos devueltos por la API y que tan sólo se realicen llamadas a la API al acceder a atributos no inicializados. Por ejemplo, cuando se obtiene un objeto de tipo Issue a través de la API, éste no contiene comentarios asociados, sino que se cargan al llamar al método get_comments de la Issue.

Sin embargo, la librería también tiene dos grandes limitaciones que influyen directamente en la gestión de tokens de GitHub:

1. Cada objeto guarda una referencia al token con el que se leyó originalmente y esta referencia se propaga con las cargas. Si leemos una incidencia y más tarde cargamos sus comentarios, estos comentarios se cargarán usando el token original de la incidencia y quedarán asociados a él. Esto hace muy difícil controlar las peticiones pendientes para un token para poder conformar un pool de tokens para el acceso a la API. Como solución se ha optado por detectar el agotamiento del token ante un respuesta fallida por parte de GitHub y esperar una hora a que el token vuelva a tener sus peticiones disponibles.

[12] http://pygithub.readthedocs.io/en/latest/index.html

2. El diseño de la librería es completamente sin estado. PyGitHub funciona como un cliente REST que no guarda estado y por tanto, en el contexto de nuestra aplicación, genera multitud de llamadas innecesarias, como por ejemplo para traer los detalles del mismo usuario autor de una incidencia una y otra vez. Si un usuario crea tres incidencias, en cada llamada al get_owner de la incidencia, se realizará una petición GET a la API de GitHub cuando a nuestra aplicación tan sólo le interesa conocer el nombre del usuario. Esto crea un agotamiento prematuro de los tokens que es difícilmente evitable.

En cuanto a la implementación en sí, se ha seguido un esquema similar al de otros extractores de Gitana [5].

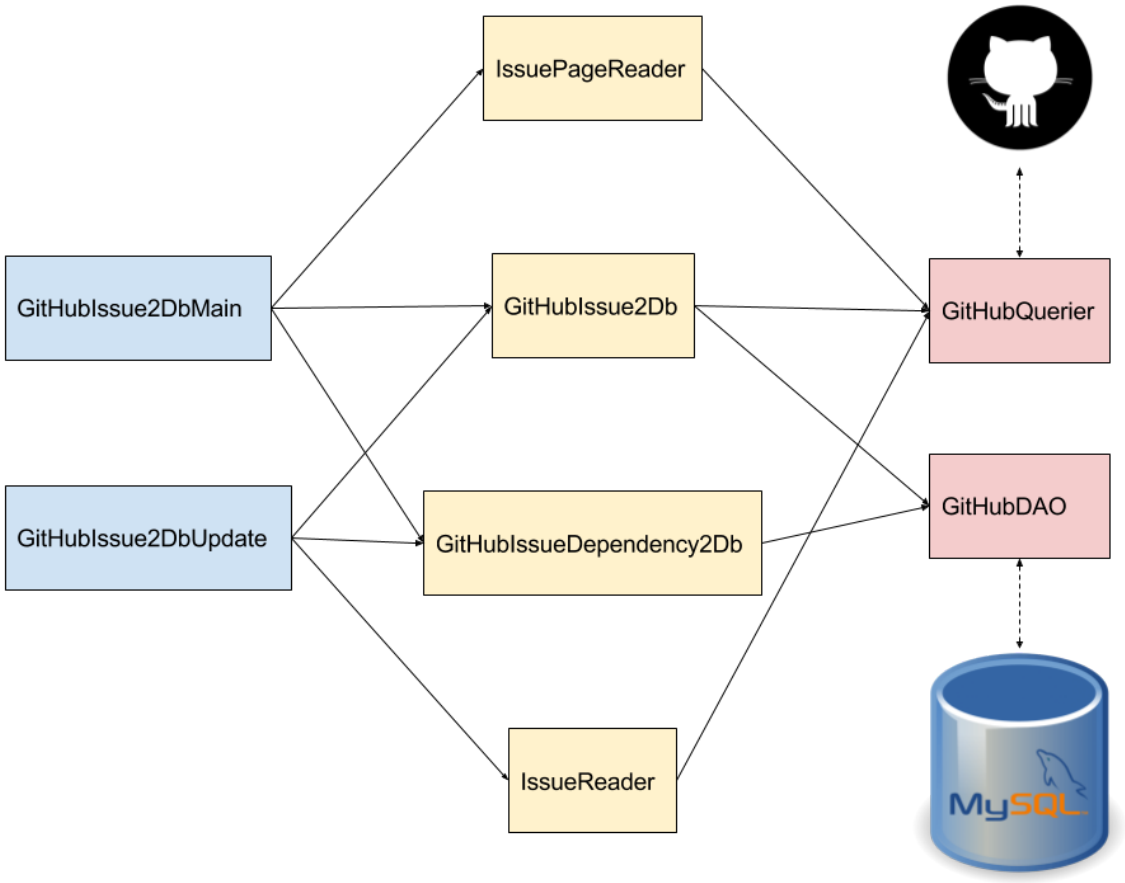

Figura 5. Estructura del programa

Tenemos dos clases principales que modelan las tres etapas señaladas en el apartado de diseño: GitHubIssue2DbMain se encarga de leer las nuevas incidencias y GitHubIssue2DbUpdate se encarga de actualizar la información de las incidencias. Ambas clases utilizan las funciones de multiprocessing_utils para coordinar la paralelización de las actividades en las tres etapas de lectura de incidencias, escritura y análisis de referencias.
Toda la interacción con la API de GitHub se centra en el archivo github_querier.py que contiene las tres clases que posibilitan la lectura de los datos de la API:
GithubQuerier es la clase fundamental que ofrece funciones tanto para leer las incidencias, como los datos relativos a ellas. Se ha optado por devolver los datos en forma de estructura de datos (diccionarios) en vez de objetos ya que ésto duplicaría prácticamente el modelo de PyGitHub sin ofrecer ventaja alguna.
Las otras dos clases, IssuePageReader e IssueReader permiten leer en paralelo páginas de incidencias o las incidencias, según se desee leer en masa o individualmente las incidencias.
Toda la interacción con la base de datos se centra en la clase GithubDAO, que es un objeto de acceso a datos sin ningún tipo de inteligencia, limitándose a ejecutar las consultas con los parámetros aportados y devolviendo resultados o logando los errores según sea oportuno.
La clase GitHubIssue2Db es la encargada de tomar cada una de las incidencias y volcarla en la base de datos, utilizando para ello GithubQuerier y GithubDAO, es decir, implementa la segunda etapa del proceso. La tercera etapa es implementada por GitHubIssueDependency2Db que utiliza GithubDAO para leer los comentarios de las incidencias, buscar referencias a otras en ellos y escribirlas en caso de existir.

6. INDICADORES

Para obtener una fotografía del estado de salud comunicativa del proyecto se establecen un total de veintinueve indicadores agrupados en dos secciones.
Una primera sección aporta una idea general para poder analizar rápidamente si el proyecto es interesante para el estudio. Se muestran datos sobre el número de incidencias, comentarios y usuarios. Así se puede decidir rápidamente si proseguir con el análisis del proyecto.
La segunda sección establece los indicadores en referencia al debate deliberativo [4] y cada sección del marco tiene una serie de indicadores que muestran el nivel del proyecto con respecto a dicha sección.
Con estos indicadores podemos comparar el proyecto con otros o evaluar la evolución del mismo a lo largo del tiempo.
A continuación se presentan estos indicadores junto con un pequeña justificación y su implementación en forma de consulta SQL contra la base de datos propuesta.

6.1 Indicadores sobre datos generales de participación

1. Número de incidencias: **select** *count*(**id**) **as 'Número de incidencias' from** issue;
2. Número de comentarios: **select** *count*(**id**) **as 'Número de comentarios' from** message;
3. Comentarios por incidencia: **select** (**select** *count*(**id**) **from** message) / (**select** *count*(**id**) **from** issue) **as 'Comentarios por incidencia'**;
4. Comentarios por usuario: **select** (**select** *count*(**id**) **from** message) / (**select** *count*(**id**) **from user**) **as 'Comentarios por usuario'**;

6.2 Indicadores sobre comunicación deliberativa

1. Inclusividad: ¿El gestor de incidencias es sólo usado por los miembros del proyecto?¿O por el contrario contribuye la comunidad? Es fundamental la implicación de la comunidad en general, así que en todo caso el cociente de ambos valores debería ser mayor que uno.
    a. Usuarios que han participado (a través de incidencias o comentarios): **select** *count*(**distinct**(u.**id**)) **as 'Usuarios que participan' from user** u **join** message m **on** m.**author_id**;
    b. Usuarios no desarrolladores: **select** *count*(**distinct**(u.**id**)) **as 'Usuarios no desarrolladores' from user** u **join** message m **on** m.**author_id where** u.**id not in** (**select** c.**author_id from commit** c);
2. Igualdad: ¿Se encuentra el gestor de incidencias monopolizado por un tipo determinado de usuario? En este sentido tan negativo es para el proyecto que la

comunicación se produzca únicamente entre usuarios no desarrolladores como únicamente entre desarrolladores. Debería haber un equilibrio.

   a. Incidencias y comentarios por usuarios desarrolladores: **select** *count*(**distinct**(m.**id**)) **as 'Incidencias y comentarios por usuarios desarrolladores' from** message m **join user** u **on** m.**author_id** = u.**id where** u.**id in** (**select** c.**author_id from commit** c);
   b. Incidencias y comentarios por usuarios no desarrolladores: **select** *count*(**distinct**(m.**id**)) **as 'Incidencias y comentarios por usuarios no desarrolladores' from** message m **join user** u **on** m.**author_id** = u.**id where** u.**id not in** (**select** c.**author_id from commit** c);

3. Identidad/Interacción horizontal: Se pretende conocer si los usuarios se reconocen entre ellos, para lo que se pueden usar las menciones. Debería haber más incidencias con menciones que sin menciones, lo cual también indicaría una adecuada gestión de la comunidad [13]
   a. Incidencias con menciones: **select** *count*(**distinct**(i.**id**)) **as 'Incidencias con menciones' from** issue i **join** issue_event ie **on** i.**id** = ie.**issue_id join** issue_event_type iet **on** ie.**event_type_id** = iet.**id where** iet.**name** = **'mentioned'**;
   b. Incidencias sin menciones: **select** *count*(**distinct**(i.**id**)) **as 'Incidencias sin menciones' from** issue i **where** i.**id not in** (**select** i.**id from** issue i **join** issue_event ie **on** i.**id** = ie.**issue_id join** issue_event_type iet **on** ie.**event_type_id** = iet.**id where** iet.**name** = **'mentioned'**);
4. Información: Se pretende evaluar como de fundamentadas están las colaboraciones, si aportan información externa. En un modelo deliberativo, la comunicación debe ser fundamentada y respaldada mediante referencias, así que una abundancia de adjuntos es en general positiva, aunque también podría indicar ruido en el canal (como la referencia a memes o artículos de opinión sin más).
   a. Incidencias y comentarios con adjuntos: **select** *count*(**distinct**(m.**id**)) **as 'Incidencias y comentarios con adjuntos' from** message m **join** attachment a **on** m.**id** = a.**message_id**;
   b. Incidencias y comentarios sin adjuntos: **select** *count*(**distinct**(m.**id**)) **as 'Incidencias y comentarios sin adjuntos' from** message m **where** m. **id not in** (**select** m.**id from** message m **join** attachment a **on** m.**id** = a.**message_id**);
5. División de tareas: ¿Se divide adecuadamente el trabajo? Se esperaría en general que las incidencias se encuentren etiquetadas y asignadas, lo cual repercutiría positivamente en el proyecto según [12].

   a. Incidencias con etiquetas: **select** *count*(**distinct**(**issue_id**)) **as 'Incidencias con etiquetas' from** issue_labelled;
   b. Incidencias sin etiquetas: **select** *count*(**distinct**(**id**)) **as 'Incidencias sin etiquetas' from** issue **where id not in** (**select distinct**(**issue_id**) **from** issue_labelled);
   c. Incidencias asignadas a un usuario: **select** *count*(**distinct**(**issue_id**)) **as 'Incidencias asignadas' from** issue_assignee;
   d. Incidencias no asignadas: **select** *count*(**distinct**(**id**)) **as 'Incidencias no asignadas' from** issue **where id not in** (**select distinct**(**issue_id**) **from** issue_assignee);

6. Poder percibido: ¿Las contribuciones son realmente relevantes?¿Se toman acciones en función de éstas?¿Varían según el perfil del usuario? Para considerar si las contribuciones de usuarios se toman en consideración, se pueden comparar las incidencias reportadas por usuarios desarrolladores con las de usuarios no desarrolladores. Sería deseable que se cerrarán más incidencias de usuarios no desarrolladores.
   a. Incidencias reportadas por usuarios con commits cerradas: **select *count*(distinct(i.id)) as 'Incidencias cerradas reportadas por usuarios desarrolladores' from issue i join issue_event ie on i.id = ie.issue_id join issue_event_type iet on ie.event_type_id = iet.id where iet.name = 'closed' and i.reporter_id in (select c.author_id from commit c);**
   b. Incidencias reportadas por usuarios sin commits cerradas: **select** *count*(**distinct**(i.**id**)) **as 'Incidencias cerradas reportadas por usuarios no desarrolladores' from** issue i **join** issue_event ie **on** i.**id** = ie.**issue_id join** issue_event_type iet **on** ie.**event_type_id** = iet.**id where** iet.**name** = **'closed' and** i.**reporter_id not in** (**select** c.**author_id from commit** c);
7. Racionalidad: ¿Se cierran las incidencias tras un debate?¿Influye el debate en el cierre de las incidencias? En general es conveniente que exista un amplio número de comentarios por incidencia cerrada, señal de que ha habido discusión pública, pero este indicador también puede reportar ruido en el canal.
   a. Media de comentarios por issue cerrada: **select** (**select** *count*(**distinct**(m.**id**)) **from** message m **join** issue_event ie **on** m.**issue_id** = ie.**issue_id join** issue_event_type iet **on** ie.**event_type_id** = iet.**id where** iet.**name** = **'closed'**) / (**select** *count*(**distinct**(i.**id**)) **from** issue i **join** issue_event ie **on** i.**id** = ie.**issue_id join** issue_event_type iet **on** ie.**event_type_id** = iet.**id where** iet.**name** = **'closed'**) **as 'Comentarios por issue cerrada'**;

   b. Media de comentarios por issue abierta: `select (select count(distinct(m.id)) from message m join issue_event ie on m.issue_id = ie.issue_id join issue_event_type iet on ie.event_type_id = iet.id where iet.name != 'closed') / (select count(distinct(i.id)) from issue i join issue_event ie on i.id = ie.issue_id join issue_event_type iet on ie.event_type_id = iet.id where iet.name != 'closed') as 'Comentarios por issue abierta';`

8. Interactividad: ¿Son frecuentes las menciones entre usuarios?¿Se menciona siempre a los mismos usuarios? Para un escenario deliberativo, los usuarios se deben reconocer entre ellos, por lo que sería deseable que el mayor número de usuarios posibles fueran mencionados, es decir, el indicador b debería ser lo más próximo posible a 1.
   a. Número de usuarios distintos con @ mention: `select count(distinct(ie.target_user_id)) as 'Usuarios distintos mencionados' from issue_event ie join issue_event_type iet on ie.event_type_id = iet.id where iet.name = 'mentioned';`
   b. Ratio frente al total de usuarios: `select (select count(distinct(u.id)) from user u) / (select count(distinct(ie.target_user_id)) from issue_event ie join issue_event_type iet on ie.event_type_id = iet.id where iet.name = 'mentioned') as 'Usuarios totales frente a usuarios mencionados';`
9. Constructividad: Se propone evaluar la utilidad del debate tomando en consideración si se consiguen cerrar las incidencias. Si todos los demás indicadores presentan un escenario deliberativo, el número de incidencias cerradas debería ser mucho mayor que el de abiertas para que el debate surta efecto.
   a. Número de incidencias abiertas: `select count(distinct(i.id)) as 'Incidencias abiertas' from issue i where i.id not in (select i.id from issue i join issue_event ie on i.id = ie.issue_id join issue_event_type iet on ie.event_type_id = iet.id where iet.name = 'closed');`
   b. Número de incidencias cerradas: `select count(distinct(i.id)) as 'Incidencias cerradas' from issue i join issue_event ie on i.id = ie.issue_id join issue_event_type iet on ie.event_type_id = iet.id where iet.name = 'closed';`
10. Eficacia: Podemos medir la eficacia de la actividad comunicativa relacionando las incidencias cerradas con referencias a commits. Sería deseable que el mayor número de incidencias cerradas cuenten con referencias a commits.
   a. Número de incidencias cerradas con referencias a commits: `select count(distinct(i.id)) as 'Incidencias cerradas con referencias a`

```
commits' from issue i where i.id in (select i.id from issue i join
issue_event ie on i.id = ie.issue_id join issue_event_type iet on
ie.event_type_id = iet.id where iet.name = 'closed') and i.id in (select
icd.issue_id from issue_commit_dependency icd);
```

b. Número de incidencias cerradas sin referencias a commits:
```
select count(distinct(i.id)) as 'Incidencias cerradas sin referencias a
commits' from issue i where i.id in (select i.id from issue i join
issue_event ie on i.id = ie.issue_id join issue_event_type iet on
ie.event_type_id = iet.id where iet.name = 'closed') and i.id not in
(select icd.issue_id from issue_commit_dependency icd);
```

11. Conocimientos relativos al proyecto: Podemos inducir como de informada está la comunidad con respecto al proyecto evaluando el número de referencias cruzadas entre incidencias.
    a. Incidencias con referencias a otras incidencias:
    ```
    select count(distinct(i.id)) as 'Incidencias con referencias a otras
    incidencias' from issue i join issue_dependency id on i.id =
    id.issue_source_id;
    ```
    b. Incidencias sin referencias a otras incidencias:
    ```
    select count(distinct(i.id)) as 'Incidencias sin referencias a otras incidencias' from issue i where
    i.id not in (select distinct(id.issue_source_id) from issue_dependency
    id);
    ```
12. Aceptación: Podemos estudiar el número de incidencias que se vuelven a abrir frente a las que no. Las incidencias se deberían cerrar siempre tras debate y alcanzar un consenso, así que este número debería ser lo más bajo posible.
    a. Número de incidencias reabiertas:
    ```
    select count(distinct(i.id)) as
    'Incidencias reabiertas' from issue i join issue_event ie on i.id =
    ie.issue_id join issue_event_type iet on ie.event_type_id = iet.id where
    iet.name = 'reopened';
    ```

Los indicadores se relacionan con las secciones del marco tal y como se presenta en la siguiente tabla:

Nivel	Condición	Número de Indicador
Entrada Institucional	Inclusividad	M.1
	Igualdad	M.2
	Apertura	El gestor de incidencias de

	Comunicación asíncrona	GitHub hace que estas condiciones se cumplan siempre.
	Visibilidad del contenido	
	Moderación	
	Identidad	M.3
	Información	M.4
	División de tareas	M.5
	Interacción horizontal	M.3
	Poder percibido	M.6
Rendimiento Comunicativo	Racionalidad	M.7
	Interactividad	M.8
	Civismo	Para evaluar el civismo habría que realizar un análisis del lenguaje empleado en las incidencias y mensajes.
	Constructividad	M.9
Resultado Productivo	Tolerancia	Ídem que civismo
	Eficacia	M.10
	Conocimientos	M.11
	Espíritu público	Para evaluar estos parámetros habría que realizar un estudio del lenguaje y además histórico para evaluar si efectivamente aumenta.
	Aceptación	
	Transformación de preferencias y consenso	
	Legitimidad	M.12

7. APLICACIÓN

A continuación se muestran los resultados de los indicadores aplicados a dos proyectos alojados en GitHub, uno de tamaño medio y otro de tamaño grande.

7.1 2048

2048[13] es un clon del popular juego 1024 escrito en JavaScript y con aplicaciones para iOS y Android. Es un proyecto interesante desde el punto de vista del estudio ya que tiene una comunidad pequeña pero aparentemente muy colaborativa, tanto es así que su principal autor cedió el control del proyecto a dos mantenedores.
Extrayendo la información de las incidencias y lanzando el cálculo de indicadores se obtienen los siguientes valores.

Número Indicador	Nombre	Valor	
G.1	Número de Incidencias	400	
G.2	Número de Comentarios	1459	
G.3	Comentarios por Incidencia	3,6475	
G.4	Comentarios por Usuario	3,1109	
M.1.a	Usuarios	469	
M.1.b	Usuarios no desarrolladores	440	93,81%
M.2.a	Incidencias y comentarios por usuarios desarrolladores	529	36,25%
M.2.b	Incidencias y comentarios por usuarios no desarrolladores	930	63,75%
M.3.a	Incidencias con menciones	74	18,5%
M.3.b	Incidencias sin menciones	326	81,5%
M.4.a	Incidencias y comentarios con adjuntos	246	16,86%
M.4.b	Incidencias y comentarios sin adjuntos	1213	83,14%

[13] https://github.com/gabrielecirulli/2048

M.5.a	Incidencias con etiquetas	23	5,75%
M.5.b	Incidencias sin etiquetas	377	94,25%
M.5.c	Incidencias asignadas	6	1,5%
M.5.d	Incidencias no asignadas	394	98,5%
M.6.a	Incidencias cerradas reportadas por usuarios desarrolladores	41	13,22%
M.6.b	Incidencias cerradas reportadas por usuarios no desarrolladores	269	86,78%
M.7.a	Comentarios por incidencia cerrada	3,7	
M.7.b	Comentarios por incidencia no cerrada	5,4132	
M.8.a	Usuarios distintos mencionados	75	
M.8.b	Usuarios por usuario mencionado	6,2533	
M.9.a	Incidencias abiertas	90	22,5%
M.9.b	Incidencias cerradas	310	77,5%
M.10.a	Incidencias cerradas con referencias a commits	33	10,64%
M.10.b	Incidencias cerradas sin referencias a commits	277	89,36%
M.11.a	Incidencias con referencias a otras incidencias	53	13,25%
M.11.b	Incidencias sin referencias a otras incidencias	347	86,75%
M.12	Incidencias reabiertas	7	1,75%

Tabla 1. Indicadores para el proyecto 2048

Los indicadores generales nos informan de que si bien el proyecto es pequeño, tiene un número bastante importante de incidencias y lo que es más importante, estas incidencias tienen una media de 3,6 comentarios. Además cada usuario tiene una

media de 3,1 comentarios, así que parece un proyecto prometedor para examinar en mayor detalle.
Los indicadores específicos del marco nos muestran un proyecto bastante inclusivo con un gestor de incidencias en el que el 93,18% de los usuarios no son desarrolladores y el 63,74% de las incidencias son reportadas por los usuarios no desarrolladores, así que el proyecto resulta bastante igualitario y no se encuentra monopolizado por ningún tipo de usuario.
De las 400 incidencias, tan sólo el 18,5% tienen menciones a otros usuarios por lo que no parece que los usuarios se reconozcan entre ellos. El 16,85% tienen adjuntos, un valor que también debería ser mayor para mostrar la existencia de opiniones fundamentadas.
El apartado de división del trabajo es especialmente bajo. Tan sólo el 5,75% de las incidencias han sido etiquetadas y el 1,5% asignadas. Por otra parte hay que tener en cuenta que se trata de un proyecto de hobby, así que es relativamente normal que la comunidad no se preocupe por una gestión eficiente del proyecto.
La sección de poder percibido indica que los usuarios no desarrolladores tienen claramente un gran poder percibido, ya que de todas las incidencias cerradas, el 86,77% son reportadas por ellos. Si se considera que estos usuarios reportan el 63,74% de las incidencias, parece que se le da una prioridad más alta a las incidencias reportadas por usuarios no desarrolladores.
Los indicadores sobre racionalidad del debate arrojan buenos datos, ya que el número de comentarios por incidencia cerrada es de 3,7, muy cercano a la media de comentarios por incidencia, mientras que en el caso de incidencias abiertas se dispara hasta 5,4 lo que indica que existe un sano debate en aquellas incidencias que quedan abiertas en vez de caer en el olvido.
Hay 75 usuarios distintos con menciones, lo cual representa algo más de la sexta parte de los usuarios. Este número contrasta con el 18,5% de incidencias con menciones y podemos plantear que no hay un gran número de menciones recurrentes entre los usuarios y más bien se trata de diálogos entre los mismos usuarios de una incidencia.
Sin lugar a dudas, el debate es constructivo ya que se encuentran cerradas el 77,5% de las incidencias reportadas, pero no se puede considerar como muy eficaz ya tienen referencias a commits el 10,96% de las incidencias cerradas.
En cuanto al conocimiento de otras incidencias, tan solo el 12,92% de las incidencias tienen referencias a otras incidencias, número que debería aumentar.
Y por último, podemos observar que la comunidad muestra un nivel bastante elevado de aceptación con un 1,75% de incidencias reabiertas.
En general se puede caracterizar la comunidad del proyecto como pequeña, bien representada (hay un gran número de usuarios no desarrolladores y con bastante

influencia) y con un nivel de debate bueno según los comentarios por incidencia. Sin embargo debería mejorar en cuanto a la gestión del proyecto (etiquetado y asignaciones) y fundamentación de las incidencias (en forma de adjuntos) y conocimientos propios del proyecto entendido como referencias a otras incidencias.

7.2 Halflife

Como proyecto de tamaño grande se presenta el SDK de Halflife[14]. Este proyecto tiene varias particularidades que hacen interesante su estudio: en primer lugar pertenece a una gran empresa (Valve Software), luego podemos esperar que exista una formalización en su forma de trabajar. En segundo lugar, en su README[15] incluye una sección específica sobre reporte de incidencias y normas de conductas. Y en tercer lugar, el proyecto no solo es grande en cuanto a tamaño de código, sino que además es muy usado en otros productos derivados como Counter Strike.

Número Indicador	Nombre	Valor	
G.1	Número de Incidencias	1751	
G.2	Número de Comentarios	9234	
G.3	Comentarios por Incidencia	5,2736	
G.4	Comentarios por Usuario	9,5294	
M.1.a	Usuarios	969	
M.1.b	Usuarios no desarrolladores	961	99,17%
M.2.a	Incidencias y comentarios por usuarios desarrolladores	1420	15,37%
M.2.b	Incidencias y comentarios por usuarios no desarrolladores	7814	84,63%
M.3.a	Incidencias con menciones	469	26,78%
M.3.b	Incidencias sin menciones	1282	73,22%
M.4.a	Incidencias y comentarios con adjuntos	1237	13,4%

[14] https://github.com/ValveSoftware/halflife
[15] https://github.com/ValveSoftware/halflife/blob/master/README.md

M.4.b	Incidencias y comentarios sin adjuntos	7997	86,6%
M.5.a	Incidencias con etiquetas	1192	68,07%
M.5.b	Incidencias sin etiquetas	559	31,93%
M.5.c	Incidencias asignadas	909	51,91%
M.5.d	Incidencias no asignadas	842	48,09%
M.6.a	Incidencias cerradas reportadas por usuarios desarrolladores	0	0
M.6.b	Incidencias cerradas reportadas por usuarios no desarrolladores	960	100%
M.7.a	Comentarios por incidencia cerrada	5,3417	
M.7.b	Comentarios por incidencia no cerrada	6,9624	
M.8.a	Usuarios distintos mencionados	289	
M.8.b	Usuarios por usuario mencionado	3,3529	
M.9.a	Incidencias abiertas	791	45,17%
M.9.b	Incidencias cerradas	960	54,83%
M.10.a	Incidencias cerradas con referencias a commits	6	0,625%
M.10.b	Incidencias cerradas sin referencias a commits	954	99,375%
M.11.a	Incidencias con referencias a otras incidencias	321	18,33%
M.11.b	Incidencias sin referencias a otras incidencias	1430	81,67%
M.12	Incidencias reabiertas	78	4,45%

Tabla 2. Indicadores para el proyecto halflife

Los indicadores generales presentan un proyecto con un número bastante elevado de incidencias, comentarios por incidencias y comentarios por usuarios, todo lo cual encaja dentro de la definición del proyecto y su popularidad y trabajos derivados.
La comunidad puede parecer muy inclusiva por el número de usuarios no desarrolladores, un 99,17%, sin embargo este número tampoco es bueno, ya que puede significar poca implicación por parte de los desarrolladores en el debate público o quizás sea política empresarial.
En la sección de igualdad si que aumenta algo el peso de la comunidad desarrolladora, siendo sus incidencias y comentarios el 15,37% lo que representa un número muy elevado para el 0,83% de la comunidad que son los desarrolladores.
El reconocimiento entre usuarios parece amplio, ya que un 26,78% de las incidencias tienen menciones a usuarios, pero solo contienen adjuntos un 13,39% de las incidencias, por lo que no parecen muy fundamentadas.
En el apartado de división de trabajo, si que se encuentra bastante organizado el trabajo, ya que el 68,07% de las incidencias tiene etiquetas y el 51,91% estan asignadas. Podemos explicar estos resultados considerando que se trata de un proyecto empresarial, así que es normal que el trabajo este más organizado.
Se da un caso interesante en la sección de poder percibido, y es que ninguna incidencia de las cerradas ha sido reportada por un usuario desarrollador, pero una inspección manual de la base de datos muestra que no hay ninguna incidencia cuyo autor sea un desarrollador, cosa habitual en un proyecto en mantenimiento en el que los desarrolladores se limitan a inspeccionar las incidencias reportadas y solventarlas.
En cualquier caso, se han cerrado el 54,82% de las incidencias reportadas, lo cual es un número bastante bueno.
El indicador de racionalidad es muy bueno, con 5,3 comentarios de media por incidencia cerrada y 6,9 por incidencia abierta, el debate sigue vivo en las incidencias abiertas.
La interactividad entre usuarios es alta, el 29,82% de los usuarios ha sido mencionado en alguna incidencia.
En cuanto a la constructividad del debate, se puede considerar que bastante constructivo, con un 54,82% de las incidencias cerradas, aunque este número debería mejorar, quizás con una mayor implicación por parte de los desarrolladores.
La eficacia también es bastante baja, ya que solo seis incidencias (el 0,34%) tienen referencias a commits y el conocimiento del proyecto tampoco parece muy elevado con únicamente un 18,38% de las incidencias conteniendo referencias a otras incidencias.
Por último, hay un alto grado de aceptación, ya que el 4,51% de las incidencias han sido reabiertas.

La caracterización del proyecto parece ser la de un producto comercial en mantenimiento, en la que si bien hay una forma coherente y uniforme de trabajar (como indican el etiquetado y la asignación), la implicación de la comunidad se divide en dos partes bien diferenciadas: reportadores de incidencias y solucionadores de las mismas.

7.3 Comparativa

A continuación presentamos una tabla para comparar el resultado de los indicadores en ambos proyectos.

Número Indicador	Nombre	2048	Halflife
G.1	Número de Incidencias	400	1751
G.2	Número de Comentarios	1459	9234
G.3	Comentarios por Incidencia	3,6475	5,2736
G.4	Comentarios por Usuario	3,1109	9,5294
M.1.a	Usuarios	469	969
M.1.b	Usuarios no desarrolladores	93,81%	99,17%
M.2.a	Incidencias y comentarios por usuarios desarrolladores	36,25%	15,37%
M.2.b	Incidencias y comentarios por usuarios no desarrolladores	63,75%	84,63%
M.3.a	Incidencias con menciones	18,5%	26,78%
M.3.b	Incidencias sin menciones	81,5%	73,22%
M.4.a	Incidencias y comentarios con adjuntos	16,86%	13,4%
M.4.b	Incidencias y comentarios sin adjuntos	83,14%	86,6%
M.5.a	Incidencias con etiquetas	5,75%	68,07%
M.5.b	Incidencias sin etiquetas	94,25%	31,93%
M.5.c	Incidencias asignadas	1,5%	51,91%
M.5.d	Incidencias no asignadas	98,5%	48,09%

M.6.a	Incidencias cerradas reportadas por usuarios desarrolladores	13,22%	0
M.6.b	Incidencias cerradas reportadas por usuarios no desarrolladores	86,78%	100%
M.7.a	Comentarios por incidencia cerrada	3,7	5,3417
M.7.b	Comentarios por incidencia no cerrada	5,4132	6,9624
M.8.a	Usuarios distintos mencionados	75	289
M.8.b	Usuarios por usuario mencionado	6,2533	3,3529
M.9.a	Incidencias abiertas	22,5%	45,17%
M.9.b	Incidencias cerradas	77,5%	54,83%
M.10.a	Incidencias cerradas con referencias a commits	10,64%	0,625%
M.10.b	Incidencias cerradas sin referencias a commits	89,36%	99,375%
M.11.a	Incidencias con referencias a otras incidencias	13,25%	18,33%
M.11.b	Incidencias sin referencias a otras incidencias	86,75%	81,67%
M.12	Incidencias reabiertas	1,75%	4,45%

Tabla 3. Indicadores comparados para los proyectos 2048 y halflife

Dadas las características particulares de cada proyecto, se hace muy difícil realizar una comparativa de ambos, ya que uno es personal y otro es empresarial.
A grandes rasgos podemos decir que 2048 destaca frente a halflife en la representatividad de su comunidad (mayor equilibrio entre desarrolladores y no desarrolladores) y su eficiencia en cuanto a cierre de incidencias.
Halflife por su parte destaca en cuanto al etiquetado y asignación de incidencias (una mejor gestión) y también tiene mejores niveles de comentarios, es decir, existe más debate.

8. CONCLUSIÓN

En el presente artículo se ha presentado la situación de los proyectos de software libre y open source en cuanto a estructura comunicativa y se ha mostrado la influencia de GitHub como herramienta colaborativa de desarrollo y debate.
También se ha realizado un breve repaso de los artículos más importantes referidos a la estructura comunicativa de las comunidades en GitHub y se ha presentado un marco teórico que caracteriza la actividad deliberativa deseada en un medio de discusión online como son las incidencias de GitHub.
Este marco nos ha permitido cumplir con los otros dos objetivos propuestos:

- Ampliar la herramienta Gitana [5] para analizar la información de las incidencias en los proyectos alojados en GitHub
- Definir unos indicadores para evaluar la comunicación en la comunidad de los proyectos.

Además se han analizado dos proyectos de características muy distintas en cuanto a tamaño, enfoque y gestión de la comunidad para mostrar su caracterización según los indicadores.
Una de las limitaciones del alcance del proyecto es que tan sólo se analiza la comunicación realizada a través del gestor de incidencia de GitHub, sería deseable incorporar otras posibles fuentes de información según el proyecto como StackOverflow [16] o Slack[17] y en función de ellas refinar y ampliar los indicadores para reflejar más fielmente la realidad comunicativa de las comunidades.
También sería interesante realizar un estudio en anchura de múltiples proyectos y estudiar si existen tipologías de proyecto según su estructura comunicativa y que factores influyen en estas tipologías.
Por último, se podría realizar un análisis a lo largo del tiempo de un proyecto tratando de introducir medidas correctoras para fomentar aquellos campos en los que adolezca y que puedan ser particularmente beneficiosos, como puede ser el uso de menciones o etiquetas. En este caso los indicadores funcionarían como elemento de medida de la utilidad de las medidas correctoras.
A nivel técnico y para mejorar la velocidad y eficiencia en la gestión de tokens de la aplicación, se podría plantear eliminar PyGitHub y desarrollar una solución propia de acceso a los datos. Además no sería muy costoso ya que el soporte de JSON en Python es muy bueno.

[16] http://stackoverflow.com/
[17] https://slack.com/

9. BIBLIOGRAFÍA

1. E. S. Raymond: The Cathedral and the Bazaar. O'Reilly Media, 2001
2. J. Cabot: Very Large Community-based Software Development[http://modeling-languages.com/very-large-community-software-development-erc]
3. J. L. Cánovas Izquierdo, V. Cosentino, and J. Cabot: "Popularity will NOT bring more contributions to your OSS project," Journal of Object Technology, vol. 14, no. 4, 2015
4. D. Friess and C. Eilders, "Analyzing Crowd Discussion. A model for assessing online deliberation. Towards a more complex approach to measure and explain deliberativeness online" in The Internet, Policy & Politics Conferences, 2014
5. V. Cosentino, J. Cánovas Izquierdo, J. Cabot. Gitana: a SQL-based Git Repository Inspector. ER 2015 - 34th International Conference on Conceptual Modeling, Oct 2015, Stockholm, Sweden
6. K. Peterson: The GitHub Open Source Development Process. Technical report, Mayo Clinic, 2013.
7. J. Tsay, L. Dabbish, J. Herbsleb: Influence of Social and Technical Factors for Evaluating Contribution in GitHub. ICSE 2014.
8. A. Lima, L. Rossi and M. Musolesi: Coding together at scale: GitHub as a collaborative social network. ICWSM 2014.
9. F. Thung, D. Lo, L. Jiang: Network Structure of Social Coding in GitHub. CSMR 2013.
10. T. F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. Le Traon: Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from GitHub. ISSRE 2013.
11. R. Kikas, M. Dumas y D. Pfahl. Issue Dynamics in Github Projects. PROFES 2015.
12. J. Cabot, J.L. Cánovas, V. Cosentino, B. Rolandi: Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects. SANER 2015.
13. Y. Zhang, H. Wang, G. Yin, T. Wang, Y. Yu: Exploring the Use of @-mention to Assist Software Development in GitHub. Internetware 2015.