UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Máster Universitario en Ingeniería en Informática

TFM del Máster Universitario en Ingeniería Informática

Análisis Visual de Revisiones de Código

Presentado por Mario Juez Gil
en Universidad de Burgos — 5 de abril de 2017
Tutores: Carlos López Nozal, Raúl Marticorena Sánchez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Máster Universitario en Ingeniería en Informática

D. Carlos López Nozal, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Mario Juez Gil, con DNI 71308224J, ha realizado el Trabajo final de Máster en Ingeniería Informática titulado Análisis Visual de Revisiones de Código.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 5 de abril de 2017

Vº. Bº. del Tutor:

D. Carlos López Nozal

Vº. Bº. del co-tutor:

D. Raúl Marticorena Sánchez

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice general III

Índice de figuras V

Índice de tablas VI

Introducción 1

Objetivos del proyecto 2

Conceptos teóricos 3
- 3.1. Conceptos relacionados con las revisiones de código 3
- 3.2. Conceptos relacionados con buenas prácticas ágiles 8

Técnicas y herramientas 11
- 4.1. Metodologías ágiles 11
- 4.2. Git 11
- 4.3. Github 12
- 4.4. Node.js 12
- 4.5. TypeScript 13
- 4.6. Visual Studio Code 13
- 4.7. Google Drive 13
- 4.8. MongoDB 13
- 4.9. Herramientas de integración continua 13
- 4.10. LaTeX 14
- 4.11. Skype empresarial 15
- 4.12. Heroku 15

Aspectos relevantes del desarrollo del proyecto 16

Trabajos relacionados **17**

Conclusiones y Líneas de trabajo futuras **18**

Bibliografía **19**

Índice de figuras

3.1. Operaciones de la inspección de software. 4
3.2. Revisiones de código en git. 6

Índice de tablas

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

Conceptos teóricos

En este apartado se van a exponer una serie de conceptos teóricos relacionados con las revisiones de código y con buenas prácticas de desarrollo ágil. Cada concepto contiene una definición y una breve descripción de su relación con el trabajo.

3.1. Conceptos relacionados con las revisiones de código

A continuación se definen diversos conceptos relacionados con las revisiones de código, las cuales son la base de este trabajo.

Revisión de código

La revisión de código es un proceso de ingeniería a través del cual se realiza una inspección del código fuente por otros desarrolladores diferentes al autor del mismo. Resulta útil para reducir los defectos de código así como mejorar la calidad del software [1].

Frente a las revisiones de código altamente estructuradas propuestas por Fagan [7], hoy en día se están adoptando metodologías más livianas con el fin de solventar las ineficiencias de las inspecciones de código. Las denominadas *Modern Code Reviews* son informales, hacen uso de herramientas, y se utilizan regularmente.

Mediante el uso de estas nuevas metodologías, las revisiones de código ofrecen un mayor número de beneficios a los equipos de desarrollo como transferencia de conocimientos, visión de equipo, o mejores soluciones a los problemas [4].

Las revisiones de código son el elemento principal de este trabajo, donde se van a obtener datos de las mismas realizadas en diversos repositorios de

software.

Inspección de software

La inspección de software, también conocida como inspección de Fagan, consiste en la búsqueda de defectos en documentos de desarrollo (código, especificaciones, diseño, etc.) por parte de personas con diversos roles mediante un proceso estructurado y bien definido, con el fin de disminuir el número de errores y aumentar la calidad del producto final [7].

Las fases de dicho proceso son las siguientes [8]:

Planificación En esta fase, los materiales que van a ser inspeccionados deben coincidir con los criterios de entrada de la inspección. Tiene lugar la elección de los participantes de la inspección. También se realiza la organización de las reuniones (hora y lugar).

Información general En esta fase se comunica a los participantes cuales son los resultados esperados. También se realiza la asignación de roles.

Preparación En esta fase tiene lugar el estudio del material por parte de los participantes y su preparación para cumplir sus roles.

Reunión de inspección Esta es la fase donde se lleva a cabo la búsqueda de defectos.

Repetición del trabajo El autor debe solucionar los defectos detectados, tras ello todo el proceso vuelve a comenzar desde la fase de planificación.

Seguimiento El moderador o equipo completo de inspección debe verificar que todos los cambios son correctos y no introducen nuevos defectos.

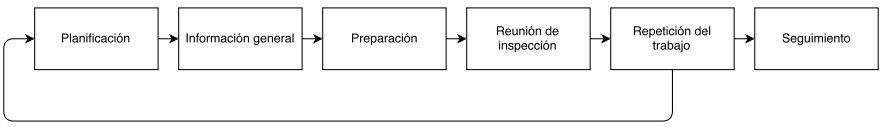

Figura 3.1: Operaciones de la inspección de software.

Además, durante el proceso de inspección de software intervienen los siguientes roles:

Autor Responsable del desarrollo del documento a inspeccionar (código).

Lector Responsable de leer el documento como si fuese a desarrollarlo él mismo.

Revisor Responsable de examinar e inspeccionar el documento con el fin de identificar posibles defectos.

Moderador Responsable de coordinar la reunión de inspección.

Revisor de código

El revisor de código tiene como responsabilidad principal examinar el código y detectar posibles defectos, ofreciendo comentarios y sugerencias de cambio al autor del código que permitan la mejora de la calidad del mismo.

Los aportes del revisor de código también pueden proporcionar nuevos conocimientos al autor del código revisado, mejorando así sus habilidades lo cual se puede traducir en una mejora progresiva de la calidad del software que desarrolle.

Se pueden distinguir dos tipos de revisores de código:

Revisor experto Es una persona, idealmente con experiencia y amplios conocimientos (aunque no es estrictamente necesario).

Robot revisor Son herramientas que permiten la revisión automática de código mediante la comprobación del código fuente para garantizar que cumpla un conjunto de reglas predefinidas, así como detectar errores [14].

Actualmente es común que las revisiones cuenten con ambos tipos de revisores, en primer lugar se realiza una revisión automática, y los resultados son utilizados por el revisor experto para ofrecer una mejor revisión con un mayor nivel de detalle.

El revisor de código es una figura importante de este trabajo. Nos interesa extraer los comentarios que registra en cada cambio puesto que pueden resultar útiles para obtener métricas que podrían permitir evaluar la calidad de los comentarios o incluso del propio revisor.

Sistema de control de versiones

Un sistema de control de versiones se encarga de registrar los diferentes cambios de un fichero o un conjunto de ficheros a lo largo del tiempo, permitiendo así recuperar versiones específicas en cualquier momento [5].

Gracias a este tipo de sistemas es posible revertir los cambios realizados en un fichero o incluso en un proyecto completo a un estado anterior, comparar los cambios, comprobar la autoría de los mismos, etc.

Los sistemas de control de versiones están en constante evolución, actualmente herramientas como Github están empezando a implementar funcionalidades para facilitar las revisiones de código. Por tanto, en este trabajo vamos a utilizar este tipo de sistemas como fuente de datos a extraer.

A continuación se muestra un diagrama donde se muestra la figura de la revisión de código y su relación con diferentes elementos de los sistemas de control de versiones (concretamente git).

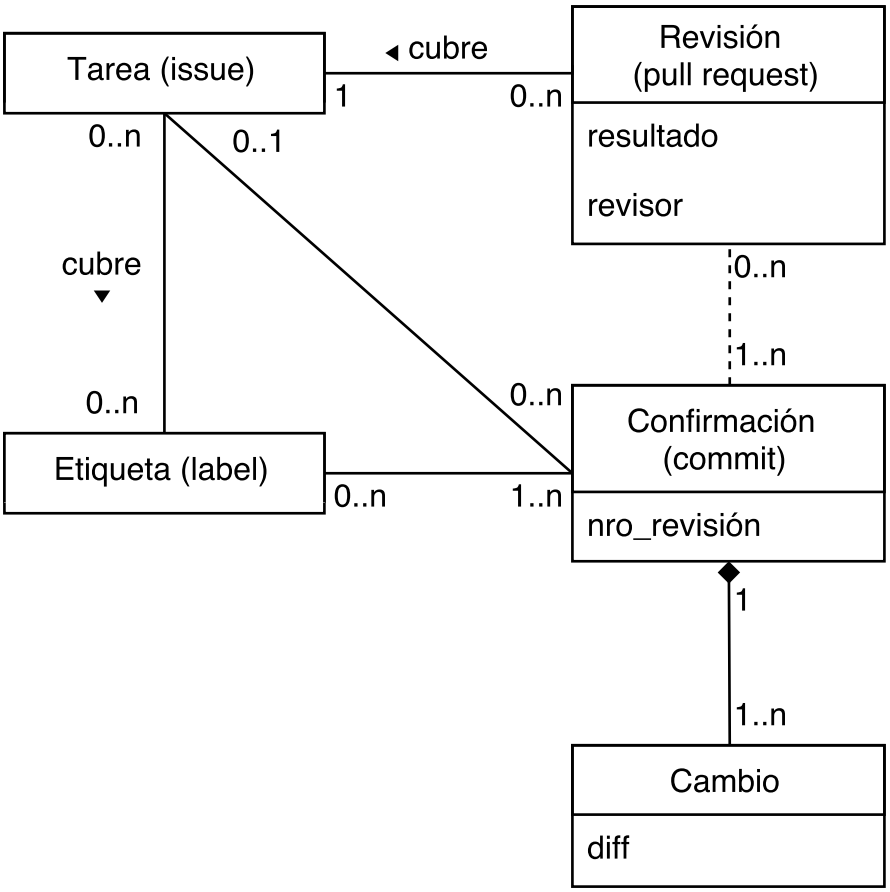

Figura 3.2: Revisiones de código en git.

Repositorio

El repositorio es la parte central de un sistema de control de versiones, es el lugar donde se almacenan los datos del sistema, normalmente estructurados en forma de árbol de ficheros [13].

La característica principal de un repositorio es que mantiene todas las versiones de cada fichero, permitiendo al cliente recuperar estados previos de

los mismos.

Cambio (diff)

Un cambio es una modificación concreta de un fichero alojado en en un repositorio de un control de versiones [15].

Resulta útil para comparar dos versiones del mismo archivo.

Rama (branch)

Una rama es un apuntador a una confirmación. Por cada confirmación realizada, la rama avanza hasta la última confirmación. Por defecto existe una rama principal [5].

Crear otras ramas sirve para abrir nuevas líneas de desarrollo, por ejemplo una rama de pruebas, evitando así desordenar la principal.

Integración (merge)

Se entiende por integración a la unión de los cambios de dos ramas en una. Existen dos patrones de uso de esta operación [6]:

- Añadir los cambios realizados en una rama donde se están implementando nuevas características a la rama principal.
- Añadir los cambios de la rama principal en una rama donde se están implementando nuevas características con el fin de mantener la rama actualizada con los últimos parches y características. Esta práctica permite reducir el riesgo de conflictos[1] al unir la rama de desarrollo con la principal.

Confirmación (commit)

Una confirmación (o commit) supone el almacenamiento en el repositorio de una nueva versión de los ficheros que contiene, en otras palabras, sirve para guardar los cambios [5].

Cada confirmación lleva asociados uno o varios ficheros modificados, un identificador denominado revisión y un mensaje donde se describen los cambios realizados.

[1] Existe un conflicto cuando al integrar dos ramas, un mismo fichero ha sido modificado en ambas y por ello es necesario decidir que cambios deben mantenerse en la versión final.

Tag

La finalidad del uso de tags es dar nombre a alguna versión para que pueda ser localizada facilmente en el futuro, es decir, permite identificar de forma sencilla revisiones importantes del proyecto [15].

Issue (bug tracker)

Las "Issues" son una funcionalidad que ofrece la herramienta Github que permiten mantener la trazabilidad de las diferentes tareas, mejoras o fallos de los proyectos [11].

Pull request

Las "Pull Requests" son una característica concreta de la herramienta Github. Sirven para mantener conversaciones sobre los cambios donde se pueden exponer ideas, asignar tareas, tratar detalles y hacer revisiones de código [10].

Etiqueta (label)

Las etiquetas (labels) de Github se pueden aplicar a las "issues" o "pull requests" como método para indicar prioridad, categoría, a que requerimiento pertenece o cualquier otra información que pueda ser de utilidad .

Por defecto Github tiene siete etiquetas: "bug", "duplicate", "enhancement", "help wanted", "invalid", "question" y "wontfix".

3.2. Conceptos relacionados con buenas prácticas ágiles

Esta parte de conceptos teóricos está dedicada a buenas prácticas ágiles que se van a utilizar a lo largo del desarrollo del presente proyecto.

Para la definición de los siguientes conceptos se ha seguido el glosario de términos de Agile Alliance [2].

Integración continua

La integración continua es una práctica de desarrollo software donde los miembros de un equipo integran su trabajo frecuentemente, normalmente como mínimo una integración diaria [9].

Tiene dos objetivos principales:

- Minimizar la duración y el esfuerzo requerido por cada tarea de integración.

- Permitir la creación de una versión del producto candidata a ser publicada en cualquier momento.

Para la consecución de estos objetivos se requiere un proceso de integración reproducible y altamente automatizado. Para tal fin se pueden utilizar herramientas de control de versiones, diversas políticas y convenciones, así como las propias herramientas de integración contínua.

Despliegue continuo

El despliegue contínuo se puede entender como una parte de la integración contínua que busca minimizar el tiempo que va desde el inicio del desarrollo de una característica, hasta que dicha característica es utilizada por el usuario final en un entorno de producción.

Construcción automática

Por construcción se entiende al proceso de convertir ficheros y otros elementos en el producto final. La construcción puede incluir:

- Compilar ficheros fuente.
- Empaquetar ficheros compilados.
- Crear de instaladores.
- Crear o actualizar el esquema o los datos de la base de datos.

Diagrama "Burn Down"

Un diagrama "Burn Down" permite relacionar la cantidad de trabajo restante (en el eje vertical) con el tiempo transcurrido desde el inicio del proyecto —burn down del producto— o un hito —burn down del sprint— (en el eje horizontal).

Desarrollo iterativo e incremental

Se dice que un proyecto ágil sigue un desarrollo iterativo cuando permite la repetición de actividades relacionadas con el desarrollo de software, así como la revisión del trabajo ya realizado mediante refactorizaciones por ejemplo.

El desarrollo iterativo también se puede caracterizar por contar con una serie de iteraciones con una duración prefijada, aunque el uso de ciertas técnicas de planificación como Kanban no requieren que esta parte se cumpla.

Actualmente, gran parte de los proyectos ágiles además de seguir técnicas de desarrollo iterativo, utilizan técnicas de desarrollo incremental, lo cual supone que cada versión sucesiva del producto debe ser usable y añadir nuevas funcionalidades visibles para el usuario (incrementos verticales).

Retrospectiva

La retrospectiva requiere que el equipo tenga reuniones regulares, normalmente dichas reuniones siguen el ritmo de las iteraciones. En ellas se deben reflejar los cambios más destacados tomando como referencia la anterior reunión, sirviendo como ayuda en la toma de decisiones para

Tablero de tareas

El tablero de tareas es una herramienta donde éste se divide en varias columnas, normalmente tres, cuyo fin es categorizar las diferentes tareas, por ejemplo en "pendientes", "en progreso", y "finalizadas".

El tablero de tareas se debe actualizar con frecuencia para mantener cada tarea en su estado real. Normalmente al inicio de cada iteración el tablero se vuelve a poner en un estado inicial para reflejar la planificación de la iteración.

Pruebas unitarias

Una prueba unitaria es un pequeño fragmento de programa escrito y mantenido por el equipo de desarrollo, su misión es ejecutar una parte del código y comparar el resultado de la ejecución con el resultado esperado. La salida de una prueba unitaria es binaria, indicando que la prueba ha pasado si cumple las expectativas, o que ha fallado en caso contrario. Comúnmente se debe desarrollar un número de pruebas unitarias acorde al tamaño del código que va a ser probado, dichas pruebas son agrupadas en lo que se denomina conjunto de pruebas unitarias o "test suite".

Técnicas y herramientas

En este apartado se definen las diferentes técnicas y herramientas utilizadas a lo largo del desarrollo del trabajo.

4.1. Metodologías ágiles

Frente a las metodologías tradicionales, se ha optado por el uso de metodologías ágiles, las cuales priman [3]:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcional sobre documentación.
- Colaboración del cliente sobre negociación de contratos.
- Respuesta ante cambios sobre seguimiento de un plan.

Scrum

Concretamente se ha optado por el uso de Scrum como metodología ágil.

Scrum propone seguir un proceso de desarrollo iterativo e incremental a través de iteraciones denominadas sprints y de revisiones. Cada iteración debe finalizar con la entrega de una parte funcional del producto [12].

4.2. Git

Git es un sistema de control de versiones distribuido diseñado para manejar proyectos de cualquier tamaño con velocidad y eficiencia.

- https://www.git-scm.com

4.3. Github

Github es una plataforma que permite alojar proyectos en repositorios de código que utilizan el sistema de control de versiones Git. También cuenta con funcionalidades para realizar revisiones de código. En este trabajo se va a utilizar esta herramienta como repositorio de código y como fuente de datos sobre revisiones de código en diferentes repositorios.

- https://www.github.com

Alternativas estudiadas

Bitbucket es una alternativa a Github como herramienta de repositorio de código.

- https://bitbucket.org

En el ámbito de herramienta de revisión de código se estudiaron Gerrit Code Review y Review Board.

- Gerrit Code Review: https://www.gerritcodereview.com/
- Review Board: https://www.reviewboard.org/

4.4. Node.js

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript v8 de Chrome. En este trabajo se utiliza para ejecutar código JavaScript en el lado del servidor.

- https://nodejs.org

npm

Node.js cuenta con un npm, un gestor de paquetes con un amplio número de librerías registradas.

- https://www.npmjs.com/

4.5. TypeScript

TypeScript es un superconjunto tipado de JavaScript que es compilado a JavaScript plano. Está desarrollado por Microsoft, y su uso puede mejorar la legibilidad y el entendimiento del código con respecto a JavaScript.

- https://www.typescriptlang.org/

4.6. Visual Studio Code

Visual Studio Code es un entorno de desarrollo integrado (IDE) multiplataforma desarrollado por Microsoft, cuenta con una herramienta integrada para el uso de Git, así como un elevado número de extensiones que permiten personalizar el editor para las necesidades concretas de cada proyecto.

- https://code.visualstudio.com/

4.7. Google Drive

Google Drive es un servicio de almacenamiento en la nube, ofrece 15 GB de almacenamiento gratuito y se utiliza en este trabajo como sistema de copias de seguridad y como medio para mantener el entorno de desarrollo sincronizado en cualquier máquina.

- https://drive.google.com

4.8. MongoDB

MongoDB es un sistema gestor de base de datos NoSQL. Ofrece escalabilidad, rendimiento y gran disponibilidad. El motivo de uso de este tipo de SGBD sobre uno de tipo SQL en este trabajo es que MongoDB utiliza documentos JSON para almacenar los registros, el mismo formato en que se obtienen los datos que deseamos almacenar desde la API de Github.

- https://www.mongodb.com

4.9. Herramientas de integración continua

Las siguientes herramientas se utilizan en el proceso de integración continua.

Travis CI

Travis CI es un sistema de integración continua. Permite automatizar tareas como la construcción, ejecución de pruebas y despliegue de aplicaciones alojadas en Github.

- https://travis-ci.org/

Gulp

TODO

Alternativas estudiadas

TODO - Grunt

SonarQube o Code Climate

TODO

ZenHub

ZenHub añade funcionalidades que permiten la gestión de proyectos utilizando metodologías ágiles dentro de Github. En este trabajo se utiliza como tablero de tareas y para generar diagramas burndown de cada sprint.

- https://www.zenhub.com

Mocha + Chai

TODO - pruebas unitarias

4.10. LaTeX

LaTeX es un sistema de composición de textos orientado a la producción de documentación técnica y científica. Está formado por un conjunto de macros TeX.

- https://www.latex-project.org/

Texmaker

TODO

Alternativas estudiadas

TODO - LaTeXila

4.11. Skype empresarial

Como herramienta de comunicación para llevar a cabo las reuniones en cada sprint se ha utilizado Skype empresarial, incluida en el paquete de Office 365 ofrecido por la Universidad de Burgos. Permite programar y realizar videoconferencias con funcionalidades añadidas como compartir pantalla o mensajería instantánea.

- https://www.skype.com/es/business/skype-for-business/

4.12. Heroku

TODO - Heroku

Alternativas estudiadas

TODO - Openshift

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski. Software inspections: an effective verification process. *IEEE Software*, 6(3):31–36, May 1989.

[2] Agile Alliance. Agile glossary and terminology. https://www.agilealliance.org/agile101/agile-glossary/. [Internet; accedido 2-abril-2017].

[3] Agile Alliance. Agile manifesto. https://www.agilealliance.org/agile101/the-agile-manifesto/. [Internet; accedido 4-abril-2017].

[4] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 712–721, Piscataway, NJ, USA, 2013. IEEE Press.

[5] Scott Chacon and Ben Straub. *Pro Git*. Apress, Berkely, CA, USA, 2nd edition, 2014.

[6] Charles Duan. Understanding Git: merging. https://www.sbf5.com/~cduan/technical/git/git-3.shtml. [Internet; accedido 14-marzo-2017].

[7] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.

[8] Michael E. Fagan. Advances in software inspections. *IEEE Trans. Softw. Eng.*, 12(1):744–751, January 1986.

[9] Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, page 122, 2006.

[10] Github. Github features. https://github.com/features. [Internet; accedido 15-marzo-2017].

[11] Github. Mastering issues. https://guides.github.com/features/issues. [Internet; accedido 2-abril-2017].

[12] J. Palacio and C. Ruata. Scrum manager. https://http://www.scrummanager.net/. [Internet; accedido 4-abril-2017].

[13] C Pilato, Ben Collins-Sussman, and Brian Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, Inc., 2 edition, 2008.

[14] Wikipedia. Revisión automática de código — wikipedia, la enciclopedia libre, 2014. [Internet; accedido 9-marzo-2017].

[15] Wikipedia. Control de versiones — wikipedia, la enciclopedia libre, 2017. [Internet; accedido 10-marzo-2017].