UNIVERSIDAD NACIONAL
Facultad de Ciencias Exactas y Naturales
ESCUELA DE INFORMÁTICA

“Facilitar la gestión de revisión de código, pruebas y documentación a los gerentes, mediante un aplicativo basado en un nuevo modelo de metadata de Gerrit”

Para optar al grado de Licenciado en Informática
con énfasis en Sistemas de Información

Nombre Estudiantes
Oscar Alas Escalante
402190360
Télefono: 83029417
oscaralases@gmail.com

Miguel Avilés Jenkins
114140211
Teléfono: 87032673
mavilesj@gmail.com

Tutor:
José Pablo Calvo Suárez

II Ciclo 2021
Heredia, Costa Rica

Agradecimientos y dedicatorias

Queremos agradecer a la Escuela de Informática de la Universidad Nacional de Costa Rica, especialmente a nuestro profesor tutor Máster José Pablo Calvo Suarez, a nuestra lectora interna Mag. Irene Hernández Ruiz y el lector externo MSc. Maximo Coghi Hernández, por su apoyo en el proceso de desarrollo de nuestro proyecto.

Del mismo modo, agradecemos el apoyo brindado por la empresa HPE (Hewlett-Packard Enterprise), específicamente la unidad de Aruba Networks, por brindarnos la oportunidad de desarrollar nuestro aplicativo para esta prestigiosa institución.

Por último, y no menos importante, estamos eternamente agradecidos con nuestras familias por el apoyo brindado a lo largo de los años para poder culminar una etapa más en nuestro desarrollo profesional.

Resumen Ejecutivo

El desarrollo ágil de nuevos productos de software ha incrementado el grado de complejidad en el manejo de distintas versiones de código fuente para un programa. Si bien es cierto existen múltiples herramientas para el desarrollo de código, documentación, manuales, entre otros, muy pocos de ellos ofrecen un sistema de control eficaz y eficiente sobre el estado actual de un proyecto, un módulo, o una función dentro de una librería.

Debido al gran auge del desarrollo de sistemas informáticos, las compañías implementan soluciones innovadoras sistemáticamente con el fin de adquirir ventaja sobre sus competidores. De esta manera, el manejo de versiones es un tema crucial en la implementación de tecnologías de información y comunicación (TIC), ya que estas podrían definir el éxito o el fracaso de un proyecto, así como disminuir el tiempo desde el momento en que se concreta la etapa de diseño hasta su desarrollo y pruebas.

Los sistemas de control de versiones (*Version System Control*) han incluido, a través del tiempo, herramientas para mejorar la experiencia de los usuarios y desarrolladores de aplicaciones y obtener un mejor control sobre las acciones realizadas dentro de un proyecto. Un ejemplo es *Git*, que incluye un módulo llamado *Gerrit*, que permite hacer la revisión de código fuente mediante un navegador web. De esa manera, facilita la colaboración de un grupo grande de desarrolladores, apoyando la aprobación o el rechazo de cambios en el código fuente en un repositorio de desarrollo.

No obstante, *Gerrit* no posee una característica que permita la localización de cambios durante el desarrollo de código fuente de manera sencilla, o la agrupación de elementos dentro de una interfaz amigable con el usuario. Por lo cual, a pesar de ser una herramienta poderosa, presenta algunas limitaciones con respecto a la manera en que se presentan los datos a los gerentes y programadores. Dado que es difícil agrupar información relevante de esta herramienta para la toma de decisiones dentro de una organización, como por ejemplo, el nivel de riesgo ante un cambio, el estado de las revisiones importantes, entre otras, se ha decidido desarrollar un complemento para *Gerrit* que permita la agrupación, búsqueda y control de cambios de manera tal que ayude a simplificar tareas de revisión

sobre los cambios que se han hecho en el código fuente en una línea de tiempo y obtener, de manera eficiente, el estado actual de las revisiones que están abiertas en el sistema.

El método utilizado para validar la mejora en la eficiencia del proceso se realizó mediante pruebas sencillas, en las cuales se cronometraba el tiempo que tarda un usuario o gerente en obtener la información de las revisiones abiertas por un equipo de trabajo, inicialmente sin utilizar el aplicativo. Luego se realizó el mismo flujo de trabajo, pero ejecutando la aplicación propuesta con resultados favorables.

TABLA DE CONTENIDOS

Índice de Tablas

Índice de Figuras

CAPÍTULO I
INTRODUCCIÓN

CAPÍTULO I: INTRODUCCIÓN

1. Antecedentes

Actualmente *Git* es una de las herramientas de control de versiones más utilizadas tanto en empresas públicas como privadas, con el fin de agilizar el desarrollo de productos de software. (Martí, 2009, p.1).

Dado el impacto que tiene la gestión eficiente de código fuente, muchas compañías dependen de herramientas que permitan controlar los cambios en el código fuente para un determinado proyecto. De lo contrario, la planificación y el tiempo de desarrollo se podrían ver afectados. Por ello, muchas de las empresas más importantes en el área de desarrollo de software utilizan *Gerrit*, tal y como se menciona en el evento *GitTogether*:

> ***Gerrit, un sistema basado en Git para la administración de revisiones de código, está ayudando a extender el popular sistema de control de revisiones en compañías que utilizan Android, muchas de las cuales cuentan con altos niveles de aseguramiento de calidad, administración y procesos legales alrededor del software. HTC, Qualcomm, TI, Sony Ericsson y Android, originado por Google, dijo el director del proyecto Shawn Pearce en una charla en el evento GitTogether en Octubre del 2009, fecha en la cual se realizó en Google Mountain View*** (Martí, 2009, p.1).

Debido a los grandes beneficios que esta herramienta trae para organizaciones orientadas a la tecnología como lo es HPE (Hewlett Packard Enterprise), la unidad de negocio de Aruba utiliza *Gerrit* como un medio de control de código para evitar errores ingresados por un ingeniero de desarrollo, analista de calidad o bien un gerente, mediante las revisiones obligatorias en caso de querer realizar algún cambio al código fuente durante el desarrollo de un proyecto. Con lo cual, se establecieron ciertos mecanismos de control como los siguientes:

- Se implementaron herramientas automatizadas que realizan pruebas de sintaxis, estándar de código, entre otros.
- Revisión por parte de ingenieros (desarrolladores, analistas de calidad, arquitectos, gerentes, entre otros)
- Revisión de regresiones (se ejecuta un conjunto de pruebas según el cambio efectuado con el fin de validar que el producto no falla debido al cambio introducido)

A pesar de ser una herramienta que permite mejorar el proceso de revisión de código e inclusión de este a la rama principal de un proyecto, presenta algunos problemas de proceso a la hora de ser implementada en proyectos de gran envergadura. Por ejemplo, *Gerrit* envía múltiples correos electrónicos para cada cambio realizado que se pretende incluir en la rama principal. Esto provoca descontento entre los usuarios que, por lo general, son programadores, analistas de calidad y gerentes.

La abrumadora cantidad de correos enviados por la herramienta hace que sus consumidores, tanto el creador del cambio como los revisores, pierdan el hilo de las revisiones abiertas. Como se comenta en foros de discusión sobre *Gerrit*, como por ejemplo Ashwini T R (Febrero 2017), "***Gerrit envía demasiados correos por revisión***". Los mismos clientes de la herramienta proveen retroalimentación sobre el uso desproporcionado del correo para enviar notificaciones. Lo anterior provoca que muchos ingenieros creen una regla para enviar los correos de *Gerrit* a la carpeta de correo no deseado, o bien, a una carpeta donde agrupan todos los correos referentes a las revisiones (chromium.org, 2016), con los cual muchas veces se olvidan de revisar dichos correos. Debido a esto, se hace sumamente complicado para los gerentes de proyectos analizar y tener un control de los cambios realizados durante el período de desarrollo, con el fin de administrar y organizar los recursos del proyecto de manera adecuada.

El cuello de botella provocado por *Gerrit*, debido a la cantidad de revisiones sin completar puede llegar a tener dimensiones incontrolables. Actualmente, no se posee un control estricto sobre las revisiones y los revisores encargados de ejecutar las mismas. A grandes rasgos, si no se controla la cantidad de revisiones abiertas, estas crecerían

rápidamente e impedir que el código llegue a la rama principal del proyecto y, por ende, se dan retrasos en los entregables. Debido a que algunas veces las revisiones no son ejecutadas a tiempo, estas quedan en el olvido, provocando retrasos, un uso inadecuado de recursos y un impacto financiero para la organización.

Si bien es cierto *Gerrit* brinda una interfaz web que permite consultar las revisiones abiertas, no hay una forma sencilla para los gerentes del proyecto de rastrear cambios importantes de código o que posean gran cantidad de tiempo en espera de revisión, o bien, cambios que han sido creados por sus equipos de trabajo.

En la siguiente tabla se podrá encontrar un ejemplo de una lista de revisiones tomadas de un equipo de trabajo en el año 2017, donde se muestra la cantidad de tiempo que cada uno de los cambios tomó, desde la fecha en la cual se abrió su revisión y la fecha en la cual fue fusionado a la rama principal del proyecto. Como se aprecia en la tabla, la gran mayoría de las revisiones superan los 15 días, lo cual demuestra lo lento que llega a ser el proceso de revisión de cambios de código, impactando la capacidad de desarrollo de nuevos sistemas e incluso la pérdida de clientes debido a atrasos en las entregas de las versiones de software.

Esta tabla posee los siguientes datos de interés:

Created On: hace referencia al día en el cual el desarrollador abrió la revisión para agregar el cambio a la rama principal.

Closed On: indica el día en el que la revisión fue aprobada y culminó el proceso de fusión de la revisión hacia la rama principal.

Delta: representa el tiempo que le tomó a cada una de las revisiones listadas, en completar el proceso de revisión y fusión a la rama principal, desde que se creó hasta que finalmente fue agregado el cambio al proyecto.

Owner: se refiere a la persona que creó la revisión

Subject: brinda una breve descripción de cuál es el cambio que se está efectuando, de modo que los revisores tengan una idea de los cambios que se están introduciendo.

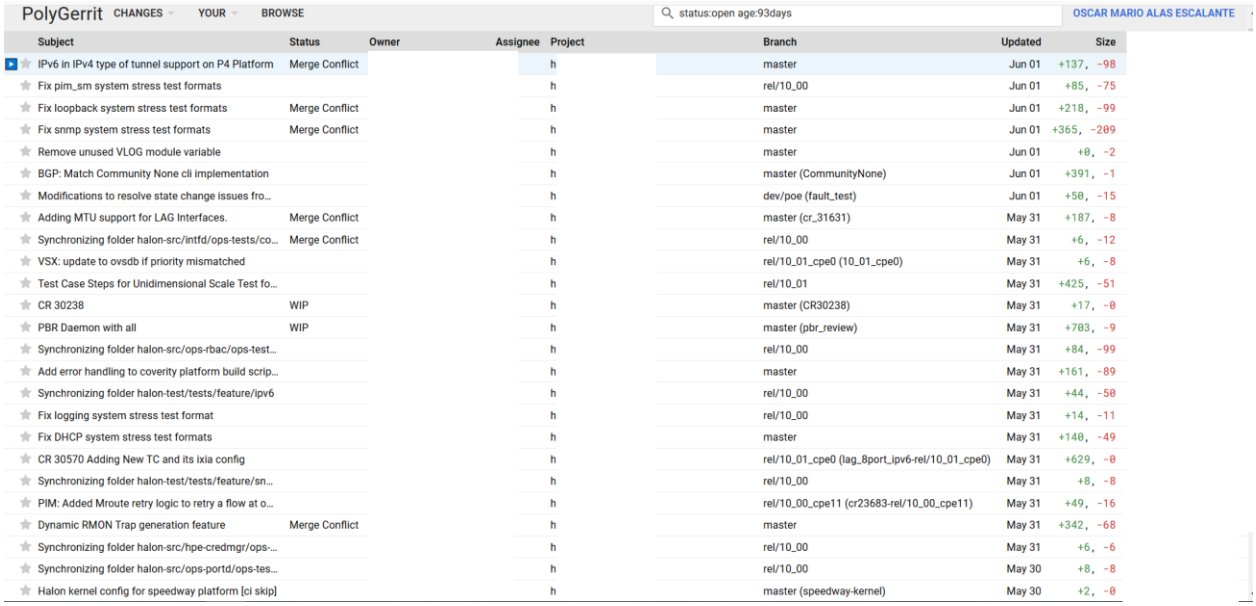
Created On	Closed On	Delta	Owner	Subject

8-jun-17	20-jun-17	12	Jose Gabriel Lopez Villalobos	Fix ct test cases for 8400
8-jun-17	**16-jul-17**	**38**	**Jorge Enrique Salazar Alvarado**	**Add skip mark for physical environments**
7-jun-17	30-jun-17	23	Francisco Jose Porras Velasquez	Fix header to exclude 8400 as supported platform
8-jun-17	**11-jul-17**	**33**	**Jorge Enrique Salazar Alvarado**	**Add skip mark for physical environments**
8-jun-17	28-jun-17	20	Pablo David Araya Martinez	CIT-QA: Adding restrictions on topology for single MM
8-jun-17	**23-jun-17**	**15**	**Jose Gabriel Lopez Villalobos**	**Fix platform supported by test case**
8-jun-17	3-jul-17	25	Francisco Jose Porras Velasquez	Fix hpe-mcast-mtm TCs to skip physical topologies
5-jun-17	**14-jul-17**	**39**	**Randall Loaiza Abarca**	**[Test] SSH system enable TC**
2-jun-17	17-jun-17	15	Francisco Jose Porras Velasquez	Fix on ft_LinkAggregation_DynamicToStaticConversion for stability in 8400
26-may-17	**19-jun-17**	**24**	**Francisco Jose Porras Velasquez**	**Fix guidelines and stability in test_Jumboframe_MaxVlanSimulateTraffic.py**
8-jun-17	26-jun-17	18	Francisco Jose Porras Velasquez	Fix hpe-mcast-mgmd TCs to skip physical topologies
8-jun-17	**15-jul-17**	**37**	**Daniel Ramirez Chaverri**	**[test] Fixing test so it can rn on ridley**
8-mar-17	12-abr-17	35	Francisco Jose Porras Velasquez	[test] Fix guidelines on link aggregation FT TCs
11-may-17	**11-jun-17**	**31**	**Francisco Jose Porras Velasquez**	**Fix guidelines and port rename in proxy-arp TCs**
6-jun-17	29-jun-17	23	Francisco Jose Porras Velasquez	Fix hpe-mcast-mtm tests to support 8400
23-may-17	**31-may-17**	**88**	**Francisco Jose Porras Velasquez**	**Fix for NTP system test cases**
7-jun-17	15-jul-17	38	Francisco Jose Porras Velasquez	Fix oobm test case to run in 8400
30-may-17	**15-jun-17**	**16**	**Francisco Jose Porras Velasquez**	**Fix guidelines and stability of test_ft_UDPForwarder_PostReboot.py**
6-jun-17	29-jun-17	23	Randall Loaiza Abarca	[Test] Include debug information on the review
3-jun-17	**18-jun-17**	**15**	**Pablo David Araya Martinez**	**CIT-QA Fix on LLDP tests for Ridley stability**
4-may-17	2-jun-17	29	Jorge Enrique Salazar Alvarado	Port renaming and guidelines fixes
6-jun-17	**25-jun-17**	**19**	**Francisco Jose Porras Velasquez**	**Fix platforms supported by the test case.**
25-may-17	30-jun-17	36	Francisco Jose Porras Velasquez	Fix physical interfaces MTU TC for guidelines and stability
19-may-17	**30-jun-17**	**42**	**Daniel Ramirez Chaverri**	**[test] Fixing st MACTables tests**
20-mar-17	12-abr-17	23	Jose Pablo Hernandez Alvarado	[test] Fix guidelines for MSTP LAG stress test
19-may-17	**2-jul-17**	**44**	**Pablo David Araya Martinez**	**CIT-QA guidelines fixing for MCLAG component tests**
16-may-17	18-jun-17	33	Daniel Ramirez Chaverri	[test/lib] Fixing guidelines for BGP
23-may-17	**18-jun-17**	**26**	**Pablo David Araya Martinez**	**CIT-QA guielines fixing for SNMP**
2-may-17	21-may-17	19	Jose Gabriel Lopez Villalobos	[test] Fix header of TC VrfConfiAndVlanMapping

Tabla 1. Tabla delta de revisiones

Fuente: elaboración propia

En la siguiente imagen se aprecia un claro ejemplo de 25 revisiones, cuya última actualización fue entre el 31 de mayo y 1 de junio de 2017, las cuales aún poseen un estado de abierto y que muy posiblemente ya fueron olvidados tanto por sus autores como por sus revisores.



Subject	Status	Owner	Assignee	Project	Branch	Updated	Size
IPv6 in IPv4 type of tunnel support on P4 Platform	Merge Conflict			h	master	Jun 01	+137, -98
Fix pim_sm system stress test formats				h	rel/10_00	Jun 01	+85, -75
Fix loopback system stress test formats	Merge Conflict			h	master	Jun 01	+218, -99
Fix snmp system stress test formats	Merge Conflict			h	master	Jun 01	+365, -209
Remove unused VLOG module variable				h	master	Jun 01	+0, -2
BGP: Match Community None cli implementation				h	master (CommunityNone)	Jun 01	+391, -1
Modifications to resolve state change issues fro…				h	dev/poe (fault_test)	Jun 01	+50, -15
Adding MTU support for LAG Interfaces.	Merge Conflict			h	master (cr_31631)	May 31	+187, -8
Synchronizing folder halon-src/intfd/ops-tests/co…	Merge Conflict			h	rel/10_00	May 31	+6, -12
VSX: update to ovsdb if priority mismatched				h	rel/10_01_cpe0 (10_01_cpe0)	May 31	+6, -8
Test Case Steps for Unidimensional Scale Test fo…				h	rel/10_01	May 31	+425, -51
CR 30238	WIP			h	master (CR30238)	May 31	+17, -0
PBR Daemon with all	WIP			h	master (pbr_review)	May 31	+703, -9
Synchronizing folder halon-src/ops-rbac/ops-test…				h	rel/10_00	May 31	+84, -99
Add error handling to coverity platform build scrip…				h	master	May 31	+161, -89
Synchronizing folder halon-test/tests/feature/ipv6				h	rel/10_00	May 31	+44, -50
Fix logging system stress test format				h	rel/10_00	May 31	+14, -11
Fix DHCP system stress test formats				h	master	May 31	+140, -49
CR 30570 Adding New TC and its ixia config				h	rel/10_01_cpe0 (lag_8port_ipv6-rel/10_01_cpe0)	May 31	+629, -0
Synchronizing folder halon-test/tests/feature/sn…				h	rel/10_00	May 31	+8, -8
PIM: Added Mroute retry logic to retry a flow at o…				h	rel/10_00_cpe11 (cr23683-rel/10_00_cpe11)	May 31	+49, -16
Dynamic RMON Trap generation feature	Merge Conflict			h	master	May 31	+342, -68
Synchronizing folder halon-src/hpe-credmgr/ops-…				h	rel/10_00	May 31	+6, -6
Synchronizing folder halon-src/ops-portd/ops-tes…				h	rel/10_00	May 30	+8, -8
Halon kernel config for speedway platform [ci skip]				h	master (speedway-kernel)	May 30	+2, -0

Figura 1. Revisiones abiertas y olvidadas.
Fuente: elaboración propia

2. Planteamiento del problema

Hoy en día para las empresas es de suma importancia contar con herramientas que les permitan agilizar los procesos y hacer un uso eficiente del tiempo y los datos acerca de las revisiones, con el fin de generar productos de software de alta calidad en el menor tiempo posible. Herramientas como *Gerrit* permiten realizar lo anterior, pero sin un uso adecuado y control necesario, pueden convertirse en un proceso lento y tedioso que, en lugar de beneficiar, afectan los procesos ágiles. Por este motivo, se necesita contar con una herramienta que permita llevar ese control sin afectar el flujo de trabajo de los proyectos.

Uno de los grandes retos al utilizar *Gerrit* es la gran cantidad de metadatos que genera, por lo que se torna muy complejo manejar el API (interfaz de programación de aplicaciones) que implementa. Se consume mucho tiempo y recursos para obtener los datos que se

requieren para la toma de decisiones, lo que hace que muchas de las revisiones queden relegadas y se mantengan en una lista de pendientes que en muchas ocasiones no se les llega a dar seguimiento. Esto definitivamente afecta el proceso de aseguramiento de la calidad de software.

Además de la complejidad del API y la estructura de los metadatos, no existe ninguna manera de dar trazabilidad a las revisiones y notificaciones sobre los cambios en el código fuente más que por correo electrónico, lo cual es molesto para los colaboradores de los grupos de trabajo y los gerentes, ya que se reciben decenas de correos electrónicos, inclusive de proyectos no relacionados. Asimismo, falta un mecanismo centralizado que permita visualizar e identificar de manera sencilla las diferentes revisiones, agrupadas por diversos parámetros, y que permita priorizar las tareas de acuerdo con las necesidades del proyecto.

Dado que la organización HPE (*Hewlett-Packard Enterpise*) adolece de una herramienta que solvente los problemas descritos anteriormente, nace la necesidad de diseñar e implementar una solución que logre aprovechar la información de *Gerrit* sobre la revisión de cambios en el código fuente. Esta aplicación deberá permitir realizar las siguientes tareas:

- Mejor control sobre los cambios en el código fuente, de manera que se puedan consultar de manera eficiente las modificaciones realizadas.

- Presentar la información sobre las revisiones en un formato amigable que permita facilitar la toma de decisiones por parte de los gerentes de proyectos. Esto ayudaría en el análisis sobre el impacto que puedan tener los cambios hechos en el código. Por ejemplo, determinar la factibilidad de agregar características nuevas o de eliminar cierta funcionalidad, de acuerdo con la estrategia de mercado y de las características del producto que se quiere lanzar.

- Acceso a la información actualizada. Debido a que actualmente las consultas de información en *Gerrit* son poco eficientes, los datos de las revisiones normalmente se obtienen de manera tardía. La aplicación se mantendría sincronizada con las

nuevas revisiones de manera automática, por lo que la información siempre se mantendrá actualizada.

3. Justificación

Luego de realizar el análisis del problema e investigar si existe alguna aplicación o *plug-in* que cubriera las necesidades presentadas por la organización, se determinó que es necesario desarrollar una herramienta que permita obtener la metadata que proporciona *Gerrit*, identificar y procesar los datos relevantes, y mostrarlos de manera amigable a los usuarios. Gerrit, al estar basado en software libre, provee distintas maneras de acceder a su información. La más común es utilizar su interfaz gráfica, pero también implementa un API REST que puede ser utilizado para extraer información específica de acuerdo con los requerimientos dados.

Actualmente, el proceso para la inclusión de código en un proyecto que utiliza *Git* como sistema de control de versiones, y *Gerrit* como sistema de revisión de código dentro de HPE, es el siguiente:

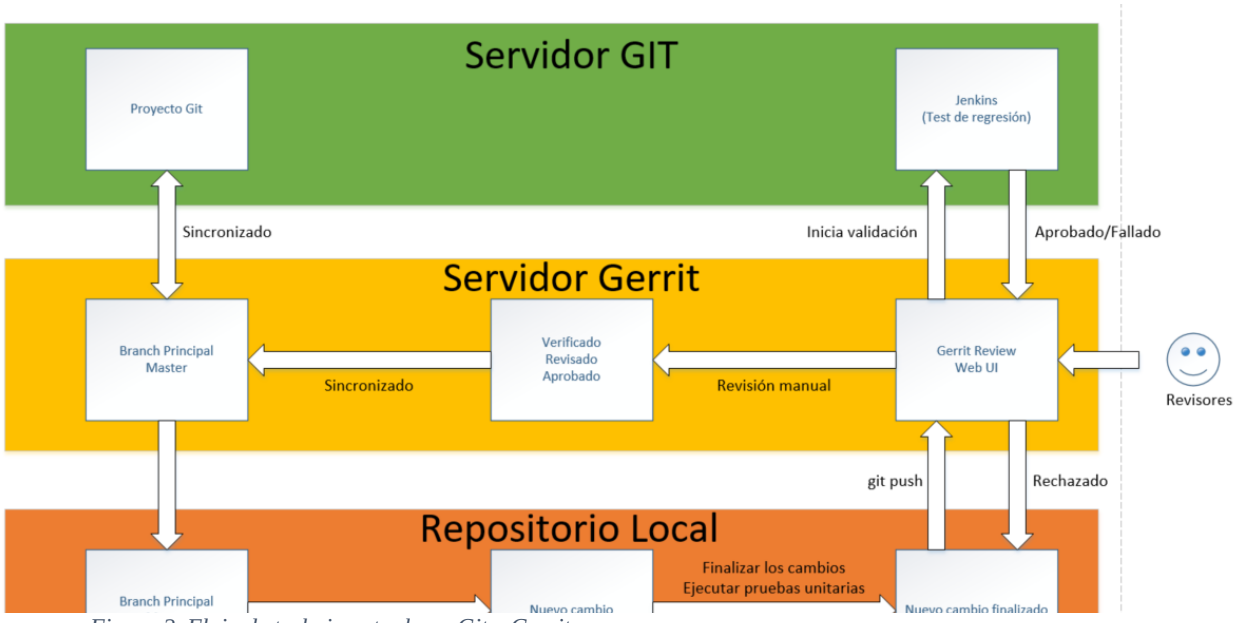


Figura 2. Flujo de trabajo actual con Git y Gerrit
Fuente: Elaboración propia

El servidor *Git* posee la última versión de código. Este es llamado la “rama principal”. Por otra parte, tenemos el servidor de *Gerrit*, el cual posee una copia actualizada o sincronizada con la rama principal que se encuentra en *Git*.

Para hacer algún cambio, ya sea agregar código nuevo, modificar código existente o bien eliminar algún segmento de código, se debe hacer realizar una copia del código en el repositorio local. Una vez que se tiene la última versión de código en el repositorio local, se crea una nueva rama de trabajo. En esta se ejecuta cualquier cambio necesario en el código. Al finalizar los cambios, se realizan pruebas unitarias para garantizar que estos no afecten la integridad del código principal. Cuando se garantiza esta integridad, se realiza una fusión del código en la rama que posee el servidor de *Gerrit*.

En este punto se inicia la compilación del código gracias a Jenkins, un servidor utilizado para la integración continua. Esto quiere decir que a medida que se agregan cambios en el código, Jenkins trabaja independiente de *Git* o *Gerrit* ejecutando pruebas de regresión después de cada compilación con el fin de garantizar la calidad. Si alguna prueba falla, la herramienta retornará un -1 en la aplicación web, de lo contrario proveerá un +1.

Usualmente, una vez que Jenkins brinda su veredicto, entran en juego los revisores del código. Estos son asignados gracias a una lista de personas encargadas de revisar el código entrante a cada uno de los módulos, con el fin de proveer retroalimentación, o bien, solicitar cambios en el código en caso de ser necesarios antes de ser fusionado con la rama principal.

Los revisores de los módulos tienen la posibilidad de brindar un +2 o +1 dependiendo de la responsabilidad que se le fue otorgada para permitir el ingreso de código nuevo a cada módulo. Por lo tanto, para que un código sea fusionado a la rama principal, requiere +1 por parte de Jenkins (ejecutando pruebas automatizadas), +1 de un revisor (este es opcional) y, finalmente, +2 de uno de los revisores principales. Por lo tanto, cuando un cambio en el código logró obtener +3, el código es fusionado a la rama principal de *Gerrit*, y continuamente a la rama principal de Git.

Debido a la gran cantidad de cambios de código existentes, la lista de revisiones pendientes crece, lo cual genera una gran brecha de tiempo desde el momento en que las revisiones son abiertas, hasta que los responsables realizan la inspección y aprobar los cambios. Basado en lo anterior, muchos cambios de código no llegan a la rama destinada, ya que no son revisados a tiempo, o en otros casos, podrían entrar a la rama principal versiones de código ya desactualizadas. A esto se suma que, para cada liberación del producto, se crean ramas que poseen arreglos en el código, casos nuevos, actualización de la documentación y otros insumos que hacen que el problema crezca de forma desproporcionada.

Existe entonces un contraste con la tendencia de uso de metodologías ágiles utilizada en la actualidad, en la que es posible que cambien las prioridades rápidamente. Hay casos en los cuales los ingenieros de HPE trabajan sobre una prueba automatizada como prioridad número uno. Luego, proceden a hacer el cambio en el código, esperando a que la persona responsable de la revisión la lleve a cabo. Sin embargo, el encargado de la revisión no la realiza a tiempo y el ingeniero que hizo la actualización en el código es asignado a otras tareas debido al cambio en la prioridad. Después de este tiempo, el responsable de examinar el código hace los comentarios necesarios, pero el ingeniero a cargo ya no tiene esa tarea como prioridad número uno, lo que conlleva a que se pierdan horas de trabajo o que el avance sea más lento de lo esperado en algunas tareas en particular.

A la hora de dar seguimiento a estos cambios de código por parte de los gerentes del proyecto, un factor muy importante es la productividad, para el que, en este momento, no existe una manera sencilla de obtener los datos para medirla. Esta productividad se estima con el tiempo que toma una revisión, desde que se abrió hasta el momento que fue fusionada con el repositorio de destino. Como se ha mencionado actualmente, no existe una manera automatizada de realizar esta tarea y los gerentes son los encargados de realizar el proceso manualmente, lo cual consume mucho tiempo y los resultados podrían ser imprecisos.

Con la creación de una herramienta que abstraiga la información más relevante de la base de datos de *Gerrit* y que mantenga la información actualizada, el flujo de trabajo mejoraría la productividad y agilizaría el tiempo de respuesta de los ingenieros ante la

necesidad de incorporar los cambios sugeridos. También, permitiría que los gerentes del proyecto puedan tener la información centralizada y agruparda de una manera conveniente, para tomar las decisiones pertinentes de acuerdo con el estado del proyecto, las necesidades de los clientes y la estrategia de la organización.

4. Objetivos del Proyecto

4.1 Objetivo general

Desarrollar un aplicativo web basado en un nuevo modelo de metadatos que complemente el flujo de trabajo de *Gerrit* para facilitar la gestión de revisión de código, pruebas y documentación para los usuarios finales.

4.2 Objetivos específicos

1. Realizar un análisis de la herramienta *Gerrit* para comprender el flujo de trabajo y extraer datos específicos de la metadata mediante las interfaces disponibles.
2. Analizar las necesidades de la compañía con el fin de proponer un nuevo modelo de metadatos que permita obtener la información necesaria para mejorar el tiempo de respuesta durante la revisión de código.
3. Implementar un aplicativo web amigable con el usuario que utilice la metadata propuesta para satisfacer las necesidades planteadas por los usuarios.
4. Evaluar el aplicativo en un ambiente de trabajo con el fin de validar si el mismo incide en la mejora de los tiempos de revisión de código y la productividad de los usuarios.

4.3 Tabla de objetivos específicos

Objetivo	Producto	Indicador logro
Realizar un análisis profundo de la herramienta *Gerrit*, como motor de	Análisis del flujo de trabajo actual, así como la lista de necesidades y metadata.	Se obtuvo un documento con la situación actual, metadatos, deficiencias y requisitos para ejecutar el proyecto cubriendo las limitaciones de la herramienta.

búsqueda para entender el flujo de su metadata.		
Analizar las necesidades de la compañía con el fin de proponer un nuevo modelo de metadatos retornando únicamente la información necesaria mejorando la productividad.	Modelo de metadatos para Gerrit	Se generó un nuevo modelo de metadatos basado en las necesidades de los usuarios que interopera con *Gerrit* con el objetivo de facilitar la información necesaria para mejorarla productividad.
Implementar un aplicativo web amigable con el usuario que utilice la metadata propuesta para satisfacer las necesidades planteadas por los usuarios.	Software	Se creó el software completo, desarrollado con los lenguajes Django, Python, HTML, JavaScript, jQuery, CSS, Bootstrap, REST y JSON. El cual cuenta con un modelo de control estadístico para la toma de decisiones, indicadores de productividad, módulo de exportación de archivos.
Evaluar el aplicativo en un ambiente de trabajo con el fin de validar si el mismo incide en la mejora de la productividad de los usuarios.	Software evaluado	La aplicación desarrollada solventó las necesidades expuestas en la etapa de análisis además de garantizar el cumplimiento de los objetivos. Conclusiones y recomendaciones

Tabla 2. Objetivos específicos

CAPÍTULO II
MARCO TEÓRICO

CAPÍTULO II: MARCO TEÓRICO

Git

¿Qué es Git?

Git es un "VCS *(Version control system)* por sus siglas en inglés", lo cual traducido al español significa sistema de control de versiones. (Chacon, 2014, p.9)

A pesar de que hay muchas semejanzas entre *Git* y otros VCS como Subversion, CVS, Perforce, entre otros, la forma de modelar los datos es completamente diferente. Algunos sistemas de control de versiones modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo. (Chacon. 2014, p.10)

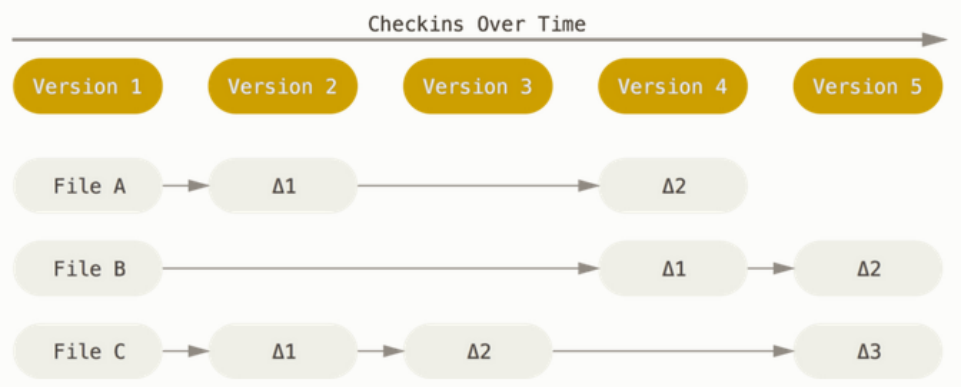


Figura 3. Otros sistemas almacenan los datos como cambios en cada archivo tomando como punto

Nota. Adaptado de ProGit (p. 14), por S. Chacon, 2014, Apress

En su lugar, *Git* modela sus datos como un conjunto de fotografías o instantáneas de un sistema de archivos. Una vez que se haya realizado un cambio, o se guarda el proyecto en *Git*, básicamente se hace una instantánea (snapshot) del estado actual de todos los archivos en ese momento, guardando una referencia a esa instancia. Para mejorar la eficiencia, si un archivo no fue modificado, *Git* no almacena el archivo de nuevo, sino que guarda un enlace o referencia al archivo anterior idéntico que ya se tiene almacenado. (Chacon, 2014, p.14)

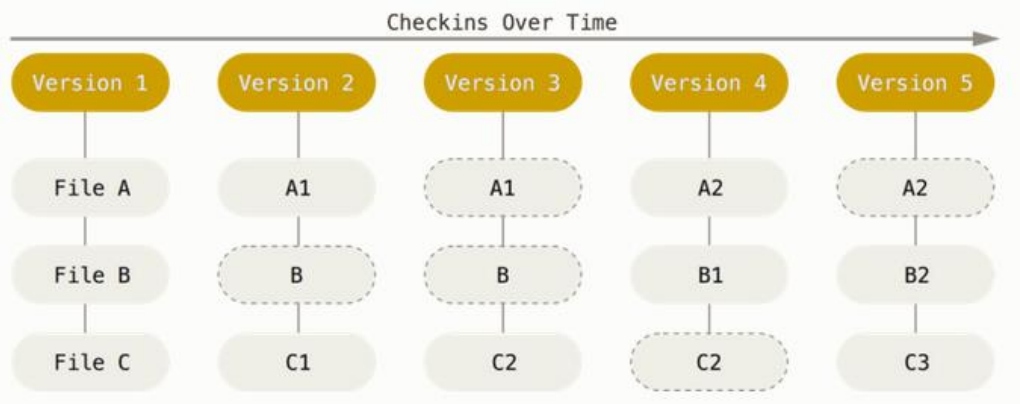


Figura 4. Git guarda la información como snapshots (instantáneas)

Nota. Adaptado de ProGit (p. 14), por S. Chacon, 2014, Apress

Para entender un poco más cuál es la problemática que deseamos solventar, es necesario que se tenga claro o al menos una noción de cómo funciona *Git*.

Git posee tres estados principales en los que para los archivos: confirmado (*committed*), modificado (*modified*), y preparado (*staged*).

Confirmado quiere decir que los datos están almacenados de manera segura en la base de datos local. *Modified* quiere decir que un archivo ha sido modificado, pero no está almacenado de manera segura en la base de datos local. Preparado quiere decir que hemos marcado un archivo modificado para que entre a la base de datos en el próximo commit o confirmación. (Chacon, 2014, p.16)

Secciones principales de un proyecto Git

Directorio de Git (Git directory)

El directorio de *Git* es donde se almacenan los metadatos y la base de datos de objetos para un proyecto. Esta es la parte más importante de *Git*, y es lo que se copia cuando se clona un repositorio desde una computadora. (Chacon, 2014, p.17)

Directorio de trabajo (Working Directory)

El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos del directorio de *Git* (*Git directory*), y se clonan en disco para que puedan ser utilizados o modificados. (Chacon, 2014, p.17)

Área de preparación (Staging area)

El área de preparación es un archivo, usualmente está contenido en el directorio actual, este archivo contiene información acerca de lo que va a ir en el próximo commit o confirmación, en otras palabras, se guardan los cambios hechos en el área de preparación junto con una breve descripción de la permuta, al utilizar el comando "***git commit***" se espera agregar las actualizaciones a la rama principal de trabajo. (Chacon, 2014, p.17)

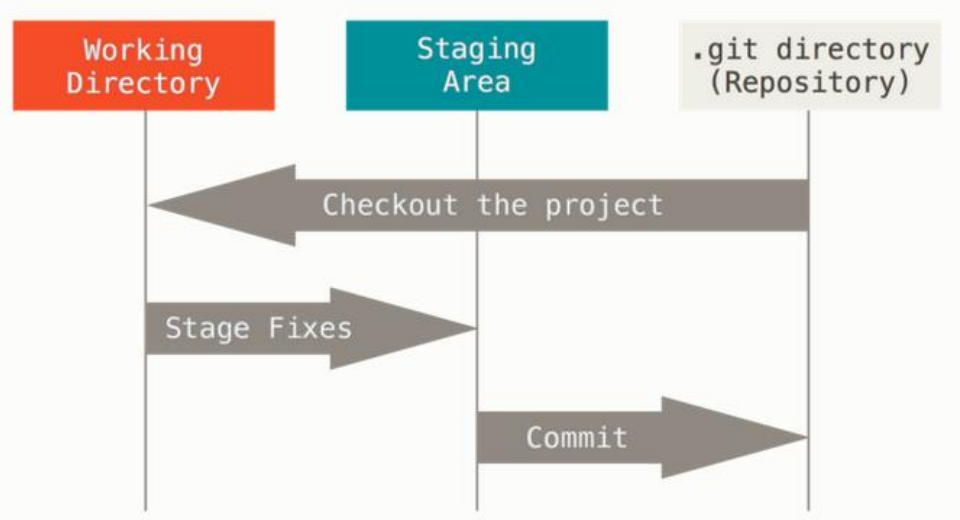


Figura 5. Secciones de un proyecto Git

Nota. Adaptado de ProGit (p. 16), por S. Chacon, 2014, Apress

Ramificaciones en Git

Actualmente todos los sistemas de control de versiones cuentan con diferentes ramas para su funcionamiento. Ramificaciones en *Git* significa que se ha tomado todo lo que se

encuentra en la rama principal (master) y se realiza una copia, siendo este un trabajo aparte y que no se vería reflejado en la rama principal, por el momento. (Chacon, 2014, p.59)

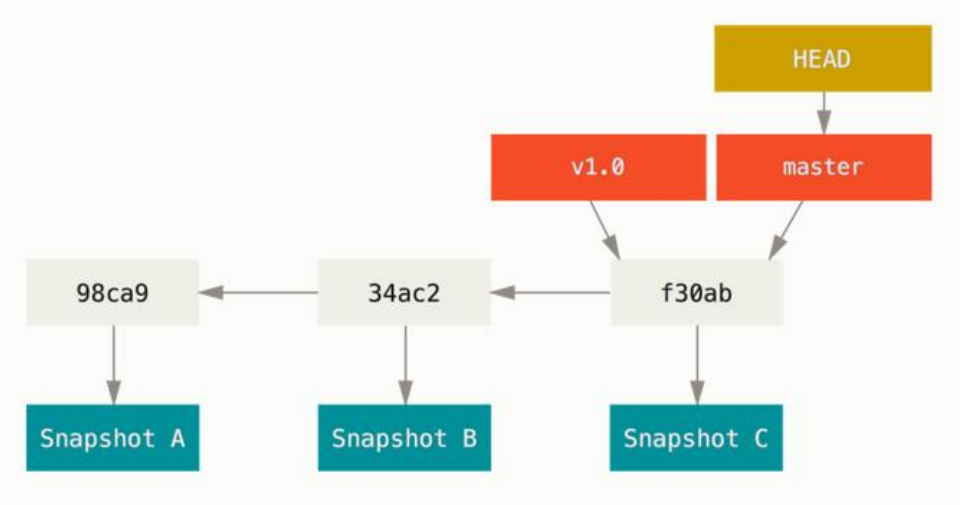


Figura 6. Ramificaciones de Git

Nota. Adaptado de ProGit (p. 61), por S. Chacon, 2014, Apress

Cada vez que se realice una confirmación (commit), *Git* crea un punto de control en el cual conserva una copia del archivo, así como metadatos como el nombre del autor, mensajes explicando los cambios realizados, el padre directo de esta rama, entre otros. (Chacon, 2014, p.59)

Flujo de trabajo

Existen varias maneras de utilizar las ramas de *Git*, lo más común es que los desarrolladores utilizan la rama principal o master para mantener ahí únicamente el código estable y limpio (el que se va a utilizar finalmente en el sistema en producción). También suelen utilizar al menos 2 ramas más: la de código de pruebas y código futuro. (Chacon, 2014, p.156)

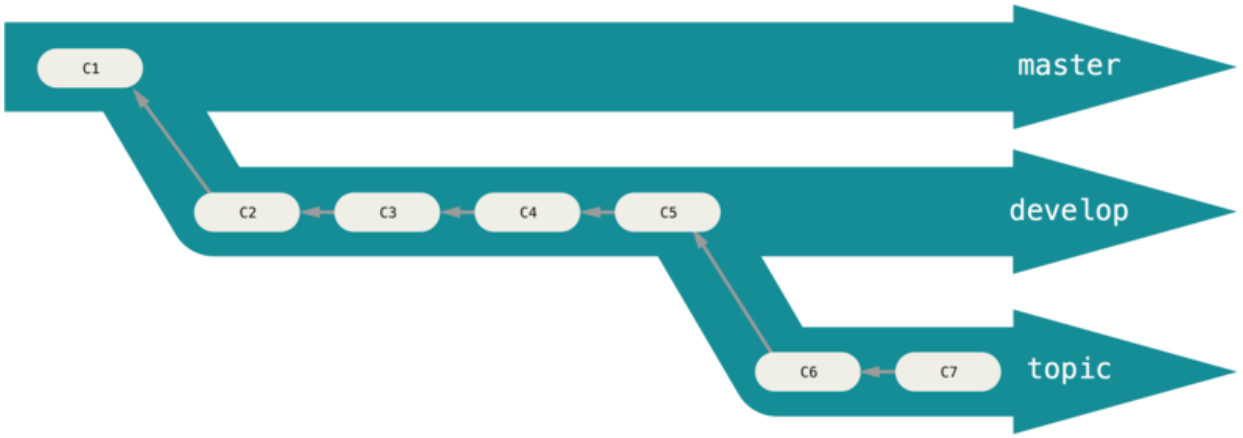


Figura 7. Flujo de trabajo Git

Nota. Adaptado de ProGit (p. 77), por S. Chacon, 2014, Apress

Gerrit

Como se ha mencionado anteriormente, lo más común es mantener un código limpio en máster. Existe un complemento para *Git*, llamado *Gerrit*, que facilita la administración de todo el código entrante a máster o cualquier rama del repositorio. (Marti. 2009)

¿Qué es *Gerrit*?

Gerrit es un sistema basado en *Git* para la gestión de revisión de código, su popularidad comenzó dentro de las compañías que utilizan Android, bebido a que las mismas tienen que brindar una garantía de calidad, gestión y procesos legales relacionados con la creación de software. (Marti. 2009). La herramienta comenzó debido a un escape progresivo de una herramienta interna de Google llamada “Mondrian”, la cual permite a los desarrolladores ver los cambios propuestos uno al lado del otro, además de agregar comentarios en cualquier parte del código bajo revisión. (Marti. 2009)

A pesar de que la herramienta era un éxito enorme dentro de Google, tenía una alta dependencia con la infraestructura interna de Google, por lo cual no podía ser utilizada fuera de la compañía. Debido a esto, Google decidió crear una versión del “Mondrian” para el público en general, la llamo *Gerrit* Rietveld, en honor a un arquitecto moderno. (Marti.

2009). El resultado de este esfuerzo es la aplicación que se utiliza hoy en día a nivel mundial para realizar revisiones de código, en compañías con altos estándares de calidad.

Para proponer un cambio de código por medio de *Gerrit*, el desarrollador debe crear una nueva rama (*branch*) *Git* para su cambio. Cada cambio, o cada arreglo dentro de un cambio, se convierten en una rama nueva. Con el fin de preservar la información entre las distintas versiones del mismo trabajo, *Gerrit* incluye un gancho (*hook*) el cual tiene un "Change-Id" para enviar los mensajes relacionados al cambio (Marti. 2009). Una vez que el cambio de código fue enviado por el desarrollador, *Gerrit* se encarga de enviar un correo electrónico a los revisores, brindando un enlace para que el revisor brinde su aporte.

En la siguiente imagen se visualiza como se despliega una revisión que está en proceso. En el lado izquierdo de la pantalla en color rojo, se aprecia el estado original de la revisión, mientras que en el lado derecho de la escena, en color verde, se observa como luce una revisión luego de haber recibido cambios con base en la retroalimentación de los revisores.

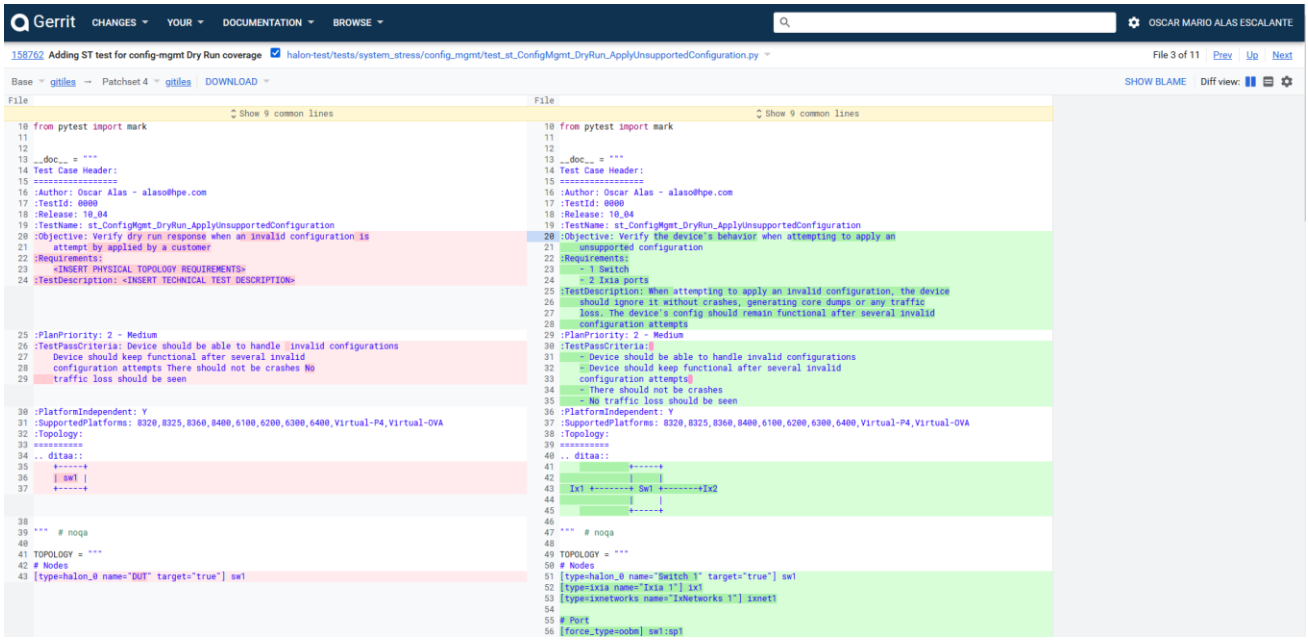


Figura 8 Código original y cambio efectuado en la revisión

Fuente: Elaboración propia

Para que un cambio sea fusionado con la rama principal, debe pasar por un proceso de aprobación de varios niveles. Por lo general, un desarrollador podría aplicar un -1 o +1,

que quiere decir “prefiero que no se envié esto” y “me parece bien”, respectivamente. Otros usuarios de mayor jerarquía tienen la opción de brindar un -2, “no enviar”, y un +2, “aprobado”. En la siguiente figura se aprecia el proceso descrito anteriormente. (Gerrit, 2021)

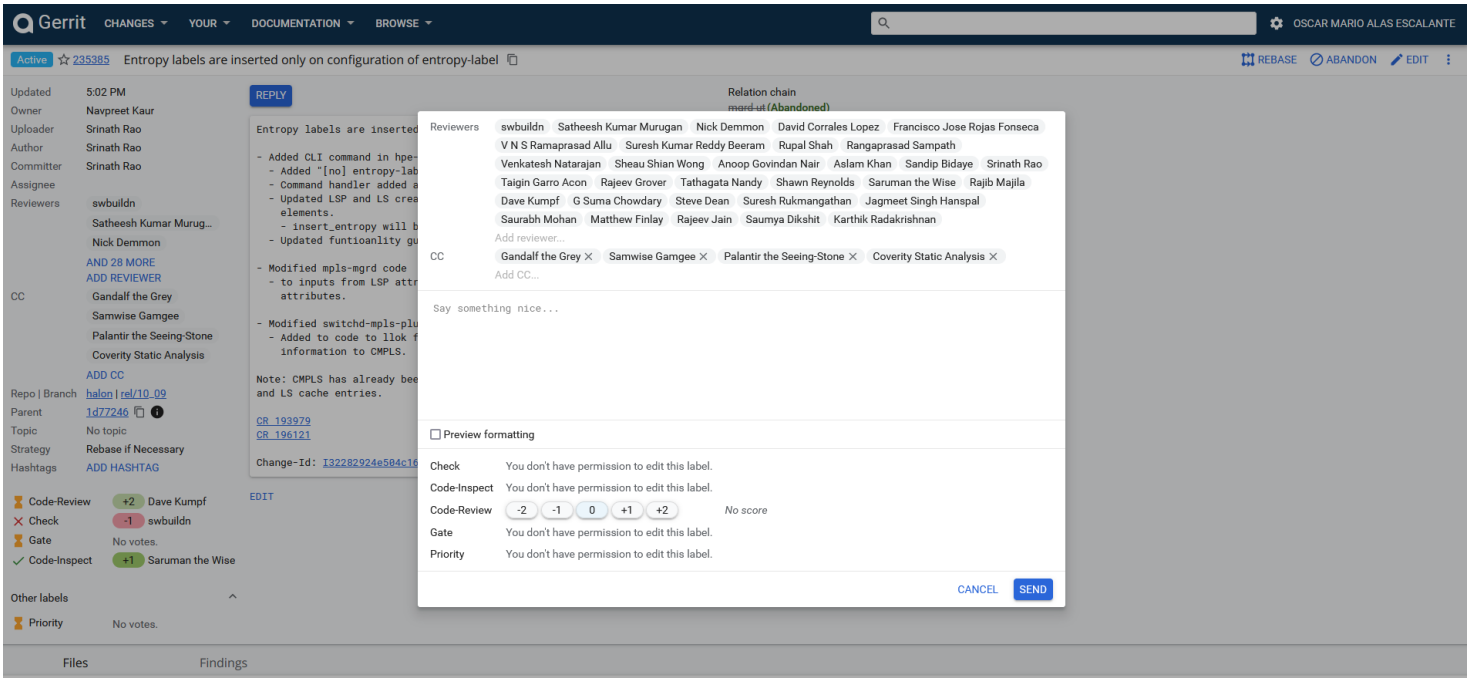


Figura 9 Proceso de revisión

Fuente: Elaboración propia

Un cambio que haya sido rechazado y que necesite retrabajo, mantiene su “Change-ID” con el fin de preservar los metadatos en *Gerrit*. El revisor, de esta manera, puede ver sus comentarios originales y las respuestas, mediante un historial que queda ligado a cada uno de los cambios. (Gerrit, 2021)

En la siguiente imagen se aprecia la lista de cambios efectuados por un usuario en específico:

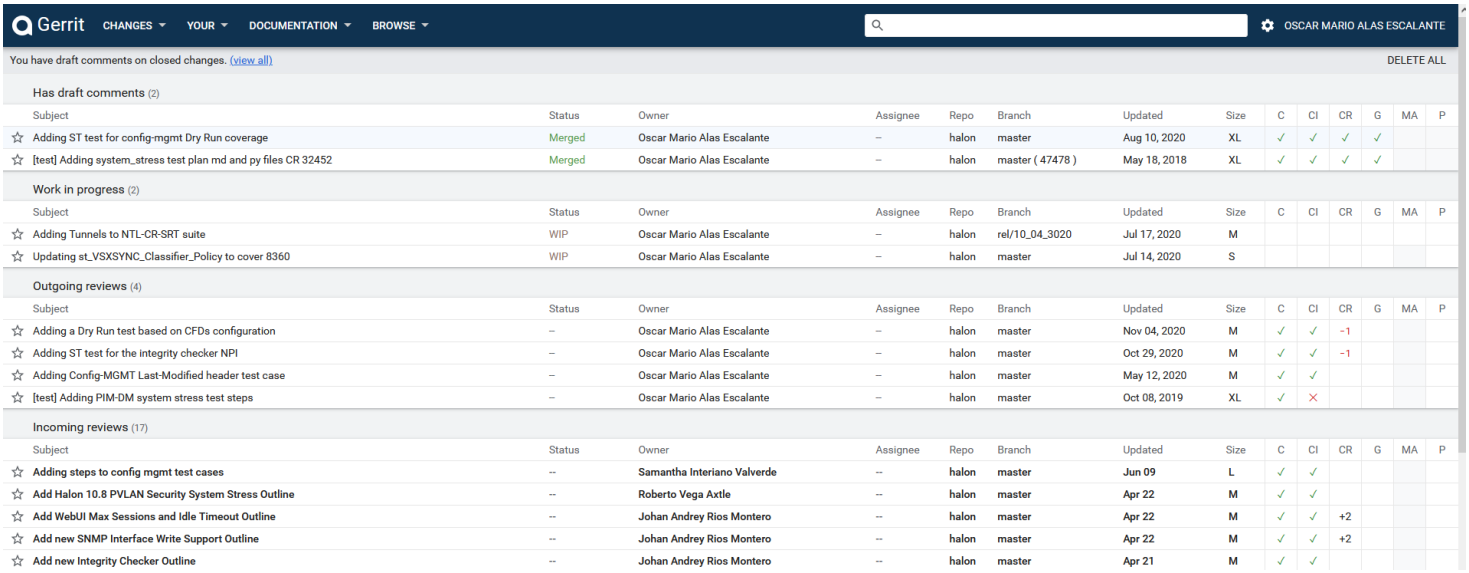


Figura 10 Lista de cambios

Fuente: Elaboración propia

En resumen, *Gerrit* da soporte para que cada cambio en el código (commit) se cree una revisión (review), en el cual una lista de personas (reviewers) realicen comentarios, y soliciten cambios o sugerencias, antes de aprobar que el código sea fusionado en su rama de destino. (Gerrit, 2021)

En la siguiente figura se aprecia la interacción que permite la herramienta de *Gerrit*, en este escenario hay 2 desarrolladores que hacen una copia del repositorio en su área de trabajo, realizan cambios y los envían a revisión. Al estar en revisión, reciben retroalimentación por parte del revisor, este ciclo se ejecuta para cada una de las revisiones que se desean agregar al repositorio. Se deben cumplir todos los requerimientos solicitados por el revisor para poder llegar al repositorio, de lo contrario, los cambios no van a ingresar sin su aprobación. (Gerrit, 2021)

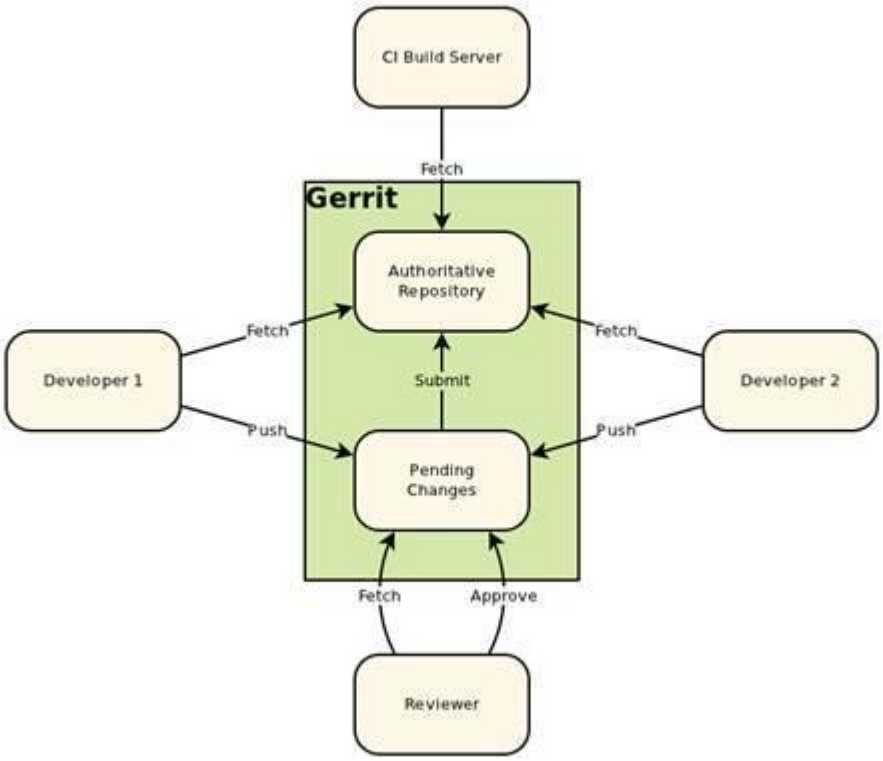


Figura 11. Flujo de trabajo Gerrit

Nota. Adaptado de Gerrit Code Review el 23 octubre 2021.

Metadata *Gerrit*

NoteDB es el backend de almacenamiento para metadatos de revisión de código *Gerrit*. Este se basa en *Git*, por consiguiente, las revisiones de código se almacenan junto al código que se está modificando. El NoteDB reemplazo el backend SQL tradicional para los cambios, cuentas y grupo de metadata usado en la versión 2.x de *Gerrit*. (GoogleSource.com, 2021)

Ventajas:

Simplicidad: Toda la información se almacena en una única ubicación dentro del directorio del sitio, en lugar de estar dividida entre el directorio del *server* y una base de datos externa.

Consistencia: La replicación y los backups utilizan instantáneas de las referencias de repositorio de *Git*.

Auditabilidad: Las modificaciones a los cambios se almacenan como una secuencia de confirmaciones (commits) de *Git*.

Extensibilidad: Desarrolladores de complementos (plugins) agregan nuevos campos a los metadatos sin que el esquema de la base de datos central tenga que estar al tanto de ellos.

Formato:

Cada revisión, o "cambio", en *Gerrit*, recibe un número de identificación único. Las diferentes revisiones, denominadas "patchsets", para un número de identificador cualquiera, están almacenadas en un directorio específico. En la siguiente imagen, se observa el número de identificador para una revisión y la cantidad de "patchsets" o cambios que esa revisión sufrió antes de lograr ser agregada a la rama master.

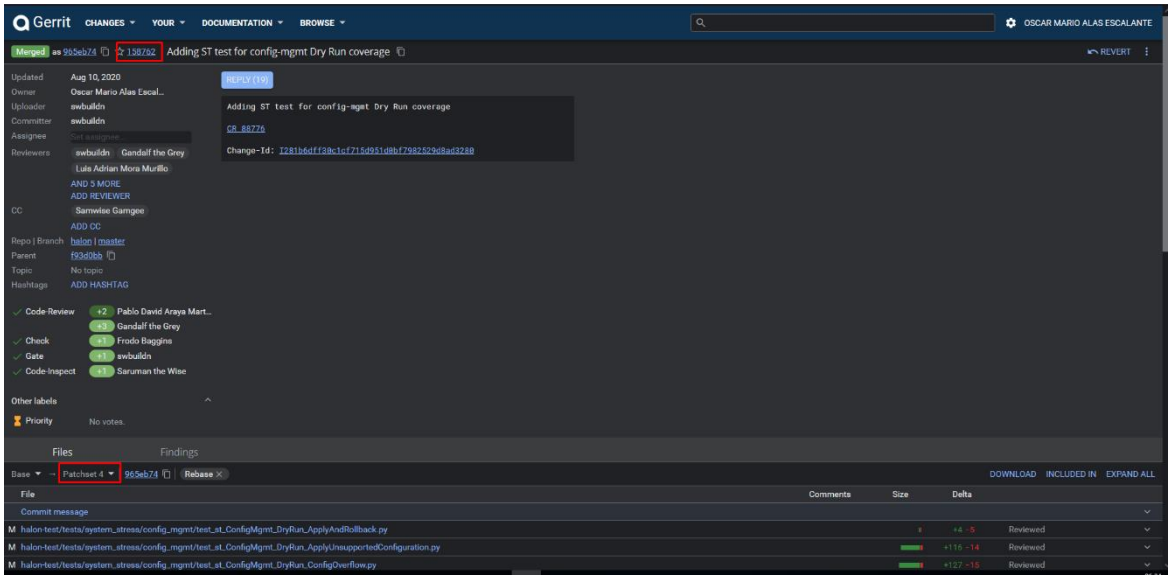


Figura 12 Número de identificación para una revisión y el patchset

Fuente: elaboración propia

Los metadatos son una rama de notas. Los mensajes de confirmación en una rama contienen modificaciones a los datos globales del cambio (votos o comentarios globales). Los comentarios en el código se almacenan en un NoteMap, donde la llave es el SHA-1 de la revisión o el conjunto de parches al que pertenece el comentario y el valor es una estructura de datos JSON. (GoogleSource.com, 2021)

La siguiente imagen muestra el ejemplo de cómo se ve el JSON de una revisión.

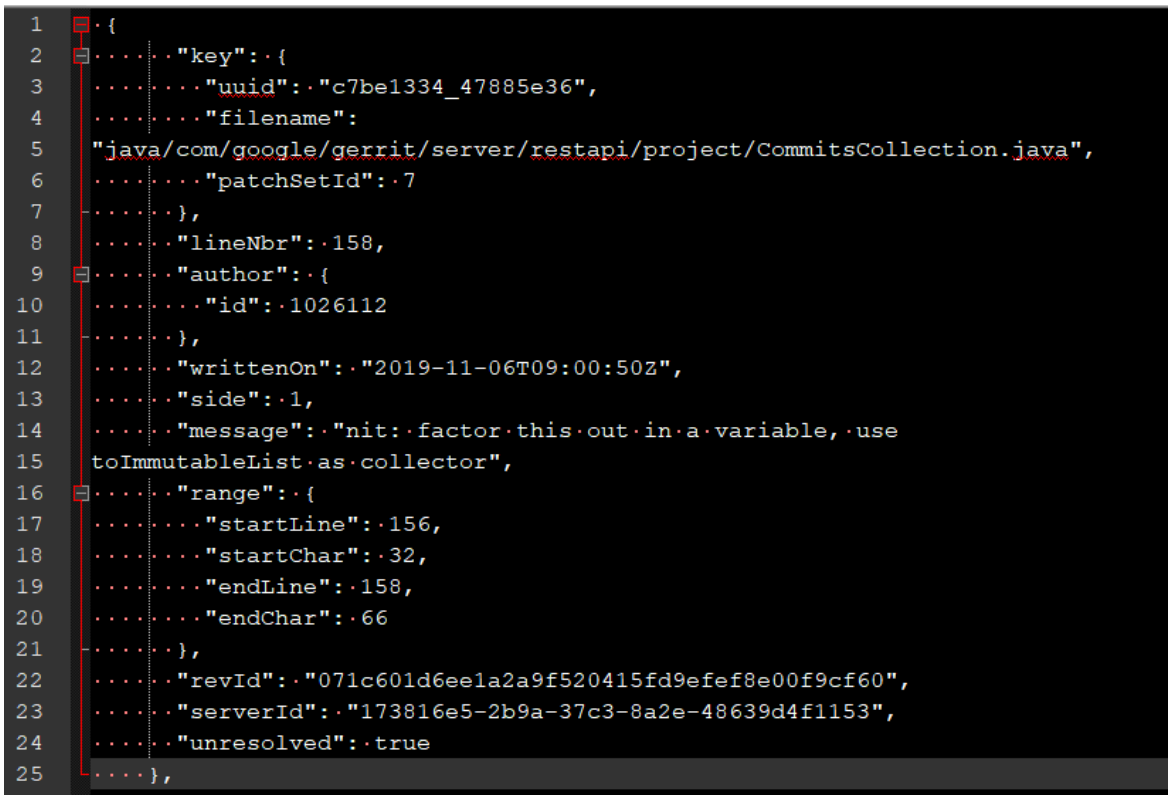


Figura 13 JSON de una revisión

Fuente: Elaboración propia

Modelo de datos REST

¿Qué es REST?

REST es una interfaz que permite conectar diversos sistemas basados en protocolo HTTP el cual sirve para generar datos y operaciones, retornando la información en formato XML o JSON. (Rosa, J. 2018)

En la actualidad, el formato JSON es el más utilizado debido a que es más legible y ligero si lo comparamos con XML (Orbis Sapienta, 2021). REST se apoya en HTTP, y los parámetros que utiliza son los mismos, con los cuales se permite realizar funciones como GET, POST, PUT y DELETE.

Un API REST, es un servicio en el cual vamos a almacenar la lógica de negocio y permite brindar los datos como una serie de recursos URL con límites definidos por el desarrollador.

¿Por qué se recomienda usar REST?

REST permite crear una petición HTTP, el cual contiene toda la información necesaria y solo espera una respuesta. De esa manera, mediante los distintos métodos utilizados por HTTP, es posible crear, modificar o borrar recursos en la aplicación.

POST: Crear/agregar recursos.

GET: Obtener un recurso/consulta específica.

PUT: Modificar un valor o variable.

PATCH: Modificar un recurso (no es un dato).

DELETE; Permite borrar un recurso.

Todos los objetos se manipulan o manejan por medio de un URI (Identificador Uniforme de Recurso), lo cual brinda trazabilidad de cuál usuario realizó el cambio dentro de la página web o sistema por medio de REST.

Ventajas de REST:

- Permite una mejor separación del modelo cliente/servidor.
- Posee gran interoperabilidad con proyectos de *Git*
- Es totalmente independiente de la plataforma, es decir es compatible con Windows, Linux, Mac o cualquier otro sistema operativo.
- Es sumamente escalable debido a la separación de cliente y servidor

JSON

¿Qué es *JSON*?

JSON (*JavaScript Object Notation*), por sus siglas en inglés, es un formato de texto sencillo para el intercambio de datos entre aplicaciones, fue especificado por Duglas

Crockford y se ha convertido en el formato de datos legible por humanos para almacenar estructuras de datos. (jason.org, 2021)

Una de las principales razones por las cuales se popularizó el uso de este formato de texto para el manejo de datos, es la forma tan sencilla de crear un analizador de sintaxis para el mismo. Además de esto, como se trata de un subconjunto de notación literal de objetos JavaScript, el cual es soportado casi en cualquier navegador, facilitó su aceptación por parte de los desarrolladores. Debido a la gran acogida y el crecimiento del uso del formato JSON, en el año 2019 se consolido como un formato independiente del lenguaje.

Por lo general, JSON se emplea en escenarios donde el tamaño de flujo de información entre uno o varios clientes y un servidor es bastante considerable. Por este motivo, compañías como Yahoo!, Google, Amazon, entre otras, han optado por implementar sus principales aplicaciones o servicios basados en JSON para el manejo de datos.

Ventajas:

Con base en la popularización de este lenguaje, sumado al gran número de grandes compañías que lo han adoptado, no es de extrañar que muchas aplicaciones de software libre se volcaran a la utilización del mismo, debido a que posee ciertas ventajas sobre XML (*Extensible Markup Lenguage*), por sus siglas en inglés, como como se describe en el artículo “JSON: Que es y para qué sirve” (Barrera, 2021)

- Es fácil de entender y es auto descriptivo
- Es más rápido que XML
- Es más sencillo de leer que un archivo en formato XML
- Es más ligero, lo cual facilita su transmisión
- El procesamiento es más veloz que XML
- Es fácilmente entendido en forma nativa por analizadores de JavaScript

JSON API

Al ser un formato que se ha extendido y popularizado cuenta, con un API (*Application Programming Interface*) en diversos lenguajes de programación, tales como:

C, C++, Java, Perl, PHP, Python, Ruby, entre otros, lo que permite analizar, procesar, transformar y generar datos en este formato. Lo anterior lo hace idóneo como método implementado para la transición de un lenguaje a otro. Es decir, una aplicación web que utiliza XML puede coexistir fácilmente con JSON sin mayor dificultad. En caso de ser necesario, el formato JSON permite ser exportado e importado en diversos lenguajes con facilidad, lo cual lo convierte en un ente versátil que permite el desarrollo sencillo de nuevas tecnologías sin aumentar el uso de recursos.

Investigaciones Similares y Mejores prácticas

Una de las principales necesidades que se pretende resolver con esta investigación es la falta de métricas del proceso de revisión de código. “El uso de métricas en el desarrollo de software está motivado por varias razones, siendo la mejora del proceso la más popular. Otros usos comunes para métricas son soporte de comunicación y ayuda con la toma de decisiones.” (Kupiainen et al., 2015).

El problema de *Gerrit* con las métricas es que no son sencillas de obtener, principalmente para usuarios sin conocimientos en lenguajes de programación. Como se mencionó anteriormente, otro de los problemas de *Gerrit* es el excesivo envío de correos electrónicos que son ignorados por los desarrolladores, por lo que es necesario reducir o eliminar esta característica. También existen problemas por falta de notificaciones en ciertas situaciones, por ejemplo, cuando un revisor agrega un comentario sin darle alguna puntuación a la revisión.

Otro problema que también está relacionado con la forma en que *Gerrit* notifica a los desarrolladores es el siguiente: cuando el revisor deja un comentario en revisión, el autor no recibirá ninguna notificación a menos que el revisor también asigne un -1 a la calificación de revisión. Estos comentarios tienen la posibilidad de permanecer ocultos por otros revisores, y eso limita la efectividad de la transferencia de conocimiento cuando se compara, por ejemplo, con las listas de correo, donde los comentarios se hacen más visibles. La interfaz de usuario de *Gerrit* también oculta los mensajes de la vista a menos que el usuario seleccione manualmente abrirlos, Lehtonen, S. (2015).

CAPÍTULO III
METODOLOGÍA

CAPÍTULO III: METODOLOGÍA

1. Tipo de investigación

El objetivo del presente proyecto es ofrecer una solución a problemas puntuales, que pueda facilitar la gestión de revisión de código, pruebas y documentación a los gerentes, mediante un aplicativo basado en un nuevo modelo de metadata de *Gerrit*, por lo tanto, el tipo de investigación desarrollada es "investigación tecnológica aplicada".

2. Población y muestra

El proyecto se realizó en la compañía HPE (Hewlett-Packard Enterprise), específicamente en una subdivisión de la compañía llamada Aruba an HPE Company.

HPE cuenta con alrededor de 59.400 empleados alrededor del mundo, (Macrotrends.net, 2021) de los cuales unos 7000 pertenecen a la organización de Aruba Networks según su sitio oficial Arubanetworks.com, 2021, la cual es la encargada del desarrollo de la nueva generación de dispositivos de redes AOS-CX, los cuales se encuentra desarrollados por medio de uso de *Git* como controlador de versiones y *Gerrit* para la revisión de los cambios efectuados. Esto quiere decir que la población total es de alrededor de 7000 empleados además de las herramientas automatizadas que interoperan con *Gerrit*.

En Aruba Networks existen diversos actores que participan en el proceso de revisión de código. Debido a que es un entorno mayormente tecnológico, hay diversos usuarios como lo son gerentes, desarrolladores, analistas de calidad y se utilizan ampliamente herramientas automatizadas. Todos los anteriores se enfocan en un único objetivo: generar un producto de calidad y garantizar la fiabilidad al usuario final.

En esta sección se habla de los diversos usuarios y su rol dentro del proceso.

Gerentes:

Por lo general los gerentes, no realizan ningún cambio de código o documentación. No obstante, están encargados de asegurarse de que los cambios para los distintos módulos

que posee su equipo de trabajo sean introducidos adecuadamente en las diferentes ramas (de ser necesario), en los tiempos pertinentes y con la prioridad adecuada.

Desarrolladores:

Los desarrolladores están divididos en grupos de trabajo para atender las necesidades de los distintos módulos del proyecto. Ellos están encargados de crear nuevas funcionalidades o arreglar defectos en un tiempo determinado, así como también desarrollar pruebas de funcionalidad para sus módulos, con el fin de garantizar el buen funcionamiento de la aplicación y evitar introducir errores que no existían anteriormente, llamados comúnmente regresiones.

Analistas de calidad:

Los analistas de calidad o QA (*Quality Assurance* por sus siglas en inglés) están encargados de desarrollar pruebas automatizadas a nivel de sistema, realizando interoperabilidad entre módulos, productos, y escalando el sistema como lo haría un usuario final para las nuevas funcionalidades introducidas y realizar validaciones de los arreglos introducidos durante el desarrollo.

Herramientas automatizadas:

- **Boromir:**
  Es el gran guardián de todas las cosas de Jenkins y es el protector apasionado del gran código. Boromir maneja los trabajos de compilación y "gate" de Jenkins.
- **Sauron:**

  A Sauron le gusta facilitar revisiones de código eficientes, por lo que agrega los revisores apropiados a las revisiones. Si la revisión del código cambia, se agregarán revisores adicionales. Además, Sauron intentará preservar los votos CR + 2 existentes en la carga de un nuevo conjunto de parches.

- **Saruman:**
  Saruman analiza todo el código modificado para hacer cumplir los estándares de codificación.

- **Gandalf:**

  Una vez que un solo revisor de cada módulo modificado haya otorgado un CR +2, Gandalf otorgará el CR +3, lo que permitirá que la revisión comience los trabajos de “gating” y, finalmente, fusionar los cambios.
- **Legolas:**

  Hace un mapeo de las pruebas necesarias para validar los cambios introducidos, con el fin de permitir a Jenkins ejecutar un subconjunto de pruebas.
- **Smeagol:**

  Cuando Boromir informa una falla o la terminación de un trabajo de Jenkins por un cambio de *Gerrit*, Smeagol intentará buscar en los registros de ese trabajo, así como en sus subtrabajos, mensajes de error. Si se encuentran mensajes de error, Smeagol categorizará los errores y enviará notificaciones por correo electrónico, si se encuentran errores de compilación, la notificación se enviará al autor del cambio.
- **Samwise:**

  Es el encargado de tener actualizada la herramienta en la que se lleva el control de los defectos, agregando comentarios a la misma en caso de que el arreglo se haya fusionado o, por el contrario, si falló el proceso de inclusión del cambio.
- **Palantir:**

  Es una herramienta se usa para observar a *Gerrit* y asegurarse de que no haya revisiones atascadas. Algunos ejemplos de reseñas "atascadas" son:
  - Gandalf no se ejecutó
  - En espera de la revisión de Yocto

  Fleur Delacour
- Fleur asiste con el proceso de ramificaciones, realizando análisis de los cambios introducidos, con el fin de identificar si es necesario hacer la elección del cambio a alguna otra rama dentro del proyecto.

La muestra para nuestra investigación consiste en el equipo de NTL (*Network Test Lab*) en Costa Rica, el cual consta de 3 equipos enfocados en el proceso de QA (Quality

Assurance o aseguramiento de la calidad por su traducción al español), con un total de alrededor de 40 colaboradores.

3. Descripción de instrumentos

Para obtener resultados del impacto de la aplicación, se realizó pruebas en un grupo focalizado que consistió de 1 gerente y 2 ingenieros de control de calidad. Se ejecutó de manera manual la búsqueda de las revisiones ejecutadas por un equipo de Costa Rica, es decir, cada participante buscó la lista de personas necesarias y, una vez con dicha lista, realizaron una búsqueda de todas las revisiones abiertas por todos los miembros. Luego, tabularon la información para realizar la toma de decisiones o tener la información actualizada, sobre cuáles fueron las revisiones que aún tenían actividades pendientes.

El siguiente paso, fue realizar el mismo proceso, pero esta vez haciendo uso de la herramienta propuesta. En este caso se encontraron varios resultados durante las pruebas. Los integrantes del grupo focal no percibieron ningún defecto o mal funcionamiento del aplicativo. Las respuestas fueron las correctas y la opción de exportar los resultados a Excel fue muy bien recibida por parte del gerente. Los tiempos de carga fueron los esperados por los integrantes de dicho grupo. Sin embargo, se encontraron muchas posibilidades de mejora, así como posibles nuevos módulos para el sistema. Dentro de las expansiones solicitadas para el sistema se encuentran: la habilidad de encontrar revisiones perdidas o que no fueron incluidos en nuevas ramas e integrar herramientas de inteligencia artificial que ayuden a mejorar la toma de decisiones. En términos generales, el aplicativo fue bien acogido por todos los miembros del grupo y se nota una clara mejoría con respecto al proceso actual, que se describe más delante de manera detallada.

4. Procedimientos para analizar la información del diagnóstico

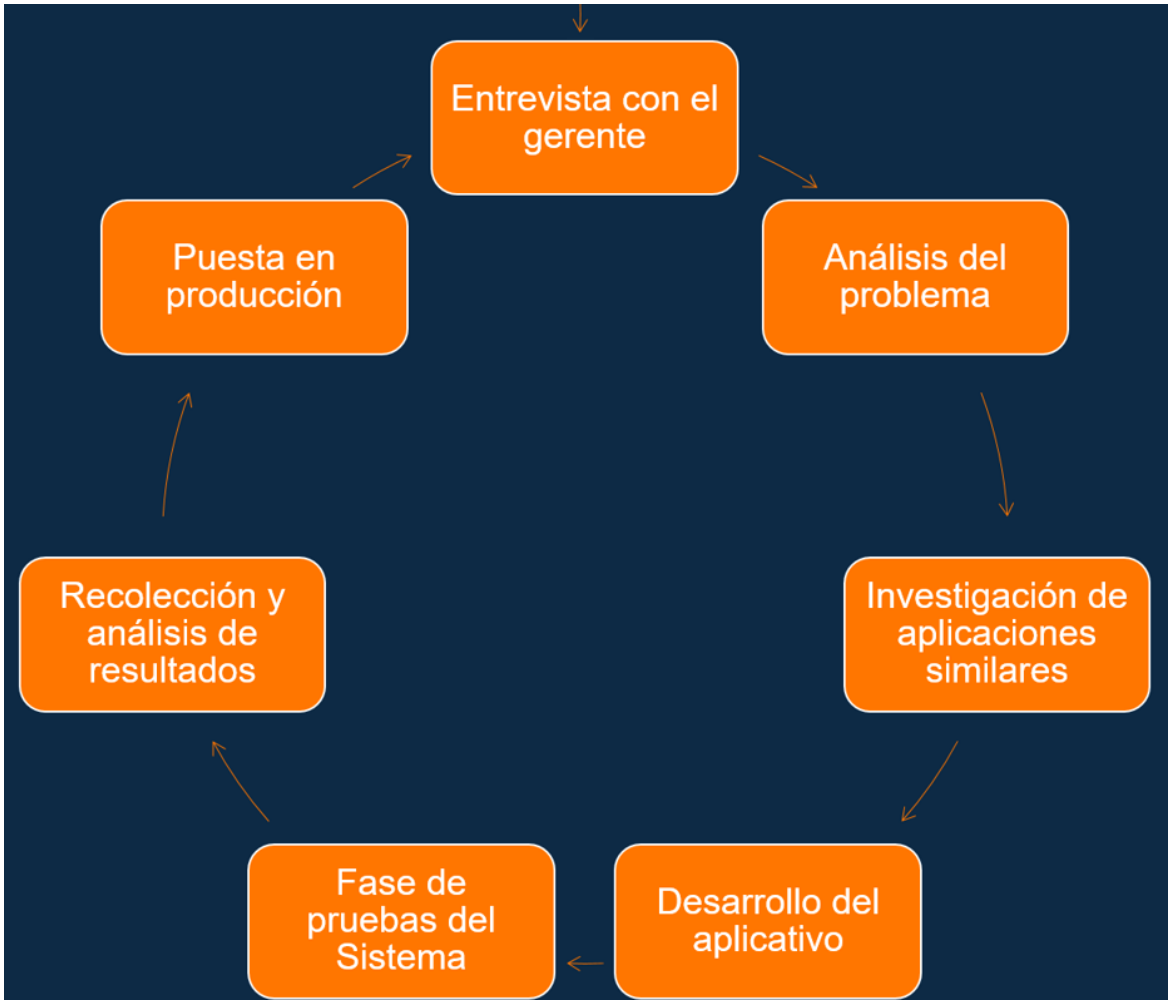


Figura 14. Flujo de trabajo para el desarrollo de la aplicación

Fuente: Elaboración propia

El procedimiento para tener un mejor entendimiento del problema consta en tratar de realizar al menos una iteración del trabajo que debe realizar un gerente o ingeniero que necesite obtener la información de las revisiones realizadas por un grupo de trabajo. Es decir, ¿cómo un gerente o jefe de equipo revisa el progreso de sus ingenieros a cargo? A pesar de que *Gerrit* cuenta con una interfaz gráfica amigable al usuario, el motor de búsqueda o flujo de filtrado de esta no es muy amigable. Si un gerente quiere buscar la

información de su grupo de trabajo, debe hacerlo de uno en uno. No hay una forma sencilla de agrupar la información o exportarla a un archivo u otra herramienta para realizar una toma de decisiones.

Actualmente, un gerente cuenta con alrededor de 12 ingenieros que les reportan directamente. Por lo tanto, si un gerente necesita obtener el estado de las revisiones de cada uno de sus ingenieros a cargo, deberá repetir el siguiente proceso uno a uno. Lo anterior podría crecer en numerosas ocasiones basado en la cantidad de revisiones realizadas por cada ingeniero, multiplicado por la cantidad de ramas con las que cuenta el proyecto. Dependiendo de cuál sea la necesidad o reporte requerido por el gerente, la siguiente lista muestra los diferentes tipos de búsqueda disponibles en *Gerrit*.

1. Buscar por ingeniero

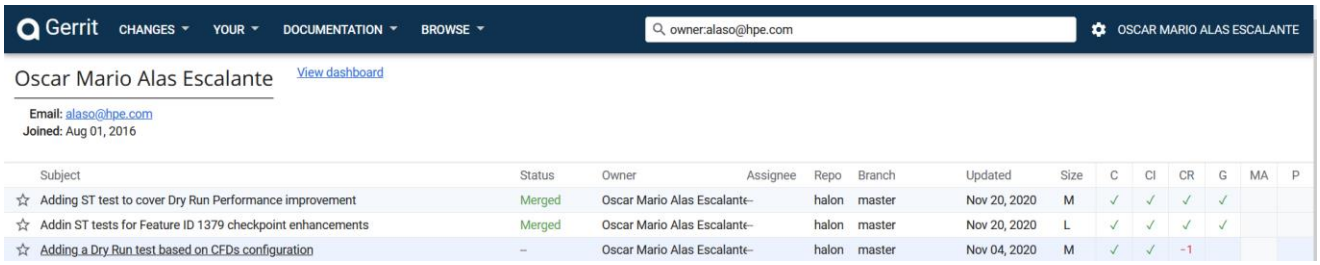


Figura 15. Buscar por ingeniero

Fuente: Elaboración propia

2. Buscar lista de revisión

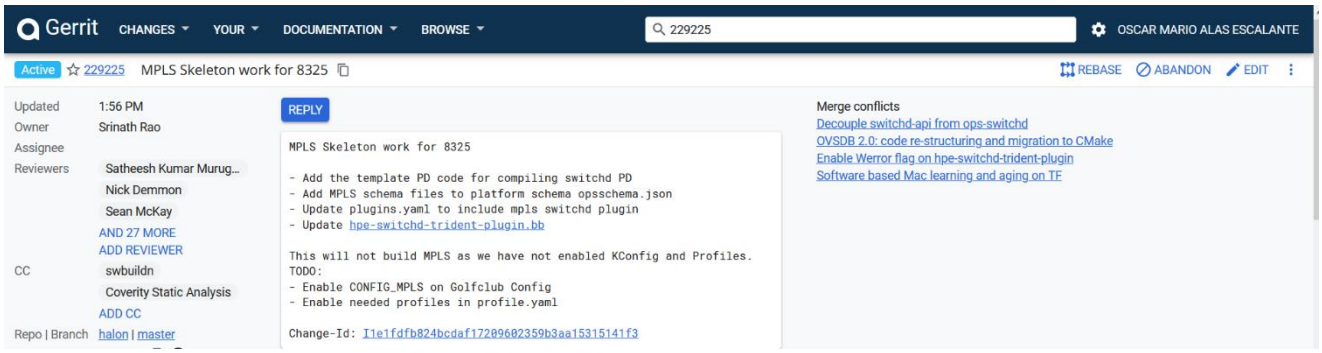


Figura 16. Buscar por revisión

Fuente: Elaboración propia

3. Buscar por rama

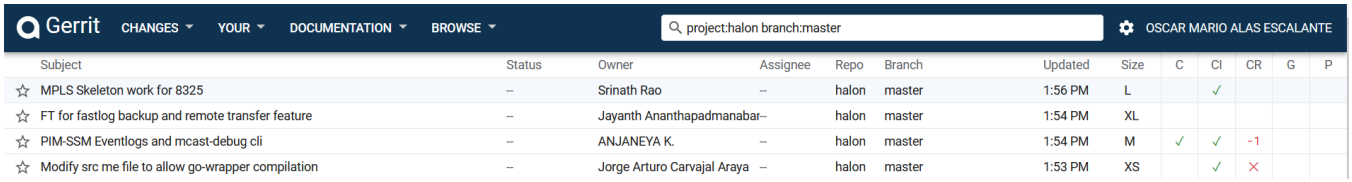


Figura 17. Buscar por branch

Fuente: Elaboración propia

En caso de que el gerente necesite obtener la información de su equipo de trabajo, el proceso debe realizarse para cada uno de los ingenieros sacando la información manualmente, aplicando el filtro con el correo del ingeniero, y copiar y pegar el resultado de la búsqueda en un archivo de Excel en caso de que se quiera tabular datos, o realizar funciones de filtrado de una manera más sencilla desde el archivo de Excel.

El resultado de la búsqueda de aplicaciones similares no tuvo frutos, dado que no se encontró referencia a ninguna aplicación que pudiera satisfacer las necesidades de la organización y que contara con las características de código abierto para modificarla y acoplarla a los sistemas ya existentes dentro de la institución como las herramientas automatizadas.

Dado que no se encontraron aplicaciones similares que pudieran solventar la necesidad, se procedió a analizar las facilidades que brinda la herramienta de *Gerrit.* El siguiente paso, fue estudiar cómo funciona el *Gerrit* API de REST, basado en la documentación existente en el sitio web de la herramienta "***Gerrit* Code Review - REST API**" (*Gerrit Code Review - REST API*, s. f.) esta etapa es crucial ya que podría definir el éxito o el fracaso del proyecto planteado.

> **Una API de transferencia de estado representacional (REST), o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) creada por el informático Roy Fielding, la cual se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful**. (RedHat, ¿Qué es una API de REST?, 2020)

En este punto, se debía tener en cuenta diversas aristas que podía impactar la funcionalidad del aplicativo a desarrollar. Las necesidades eran claras, sin embargo, la implementación de la solución requería una exhaustiva investigación y entendimiento del API que brinda *Gerrit* para definir si se podían cubrir los requerimientos planteados por la organización y acoplarlo a las herramientas ya existentes. Para nuestra fortuna, a pesar de ser un API bastante extenso con muchas áreas no tan bien definidas o documentadas, al realizar este proceso de análisis se logró comprender cómo es que funciona la metadata y como se podría solventar el problema planteado.

El proceso para lograr obtener los datos requeridos de la manera adecuada y de forma optimizada tomó mucho tiempo, dado que la documentación tenía varias áreas grises, en donde no se tenía bien definido o claro el proceso para obtener los datos. Así mismo, el API de REST provee mucha información que no es de utilidad o importancia para nuestros requerimientos o para los usuarios finales. El trabajo de comprender la información, filtrarla y devolverla de un modo adecuado, es lo que podría definir el éxito o fracaso de este proyecto.

Una vez que se lograron obtener los datos adecuadamente, el siguiente reto que se afrontó fue acoplar la herramienta que se estaba implementando con otras herramientas ya existentes en la organización, esto con el fin de reutilizar código y brindar una solución óptima que cualquier usuario pudiera utilizar sin mayor esfuerzo. Una de las principales herramientas con las cuales se necesitaba tener interoperabilidad es llamada "*easyMai*l", la cual provee una lista o grafo con la jerarquía definida en la organización. En la siguiente figura se aprecia un ejemplo de organigrama dentro de HPE.

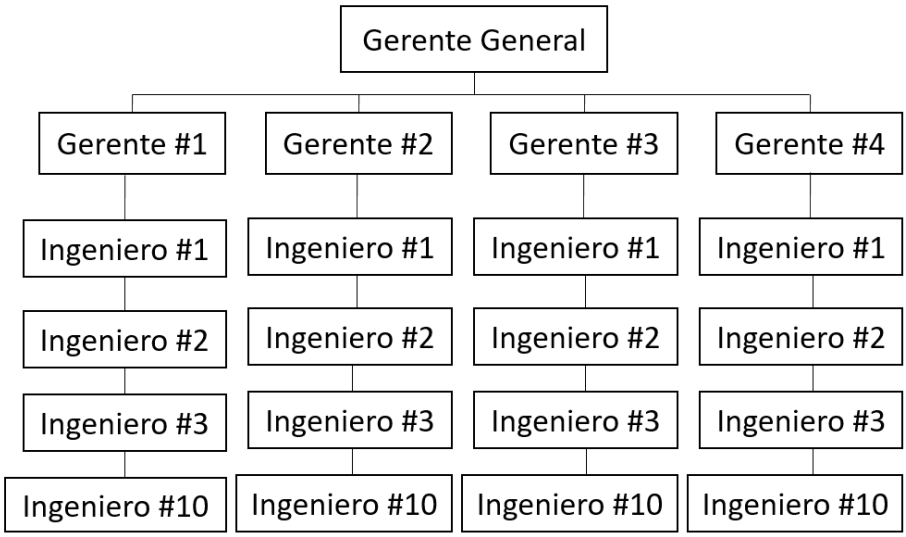


Figura 18. Diagrama organizacional

Fuente: Elaboración propia

Por ejemplo, la herramienta easyMail, facilita la lista de personas que le reporta a un gerente en específico. En este caso, si el "Gerente #1" quiere enviar un correo a todos los empleados que le reportan al "Gerente General", puede hacerlo mediante el uso de esta herramienta.

Ahora, la pregunta sería, ¿qué beneficio tiene una aplicación que genera correos para solventar las necesidades de las revisiones de *Gerrit*? El simple hecho de obtener la jerarquía de la organización brinda la facilidad de realizar búsquedas agrupadas sin tener que ingresar uno a uno los ingenieros que se desea consultar, esto debido a que la información de cuál empleado le reporta a cuál gerente no siempre es tan sencilla de obtener, y la mejor manera de implementarlo de un modo transparente en nuestro aplicativo es usando un método ya existente.

CAPÍTULO IV
PROPUESTA DE SOLUCIÓN

CAPÍTULO IV: PROPUESTA DE SOLUCIÓN

1. Propuesta de solución

Mediante la consulta a expertos se creó una propuesta de un nuevo modelo de metada que va a ser consumido mediante un aplicativo. La consulta se realizó siguiendo ejemplos de modelos utilizados anteriormente de manera manual por parte de los usuarios, con el fin de abstraer los principales filtros que podrán ser utilizados en todos los proyectos de manera rápida y sencilla.

Para la implementación de un tablero de control mediante el cual se consulte el modelo propuesto, fue necesario analizar las últimas herramientas de programación web que cumplan con los estándares requeridos y que permitan la construcción de una interfaz que sea intuitiva y amigable con el usuario. Además, debería proporcionar mucha flexibilidad y que preferiblemente sea multiplataforma.

2. Validación de la propuesta

El aplicativo se evaluará en un ambiente de trabajo, en la empresa Hewlett Packard Enterprise, ubicada en la Zona Franca América, Flores, Heredia. Para medir el rendimiento del proyecto, se comparan resultados antes y después. También se creará una tabla comparativa con la información brindada por la herramienta y la que se podría obtener de manera manual con Gerrit. Además, como se ha mencionado anteriormente, el tiempo es el factor principal para evaluar el rendimiento de los flujos de trabajo. Una vez que se hayan generado los datos consultados, se creará una tabla para comparar los resultados de las consultas.

Para evaluar el aplicativo, se realizó la misma consulta que consta de realizar una búsqueda utilizando el mismo gerente, obtener sus empleados directos y luego obtener el estado de revisión por empleado. Para hacer la comparación, se utilizó el método actual contra el aplicativo propuesto.

El método actual consta de varios pasos los cuales son:

- Ingresar a la herramienta "*easymail*"
    - Realizar la consulta de los gerentes y sus empleados directos.

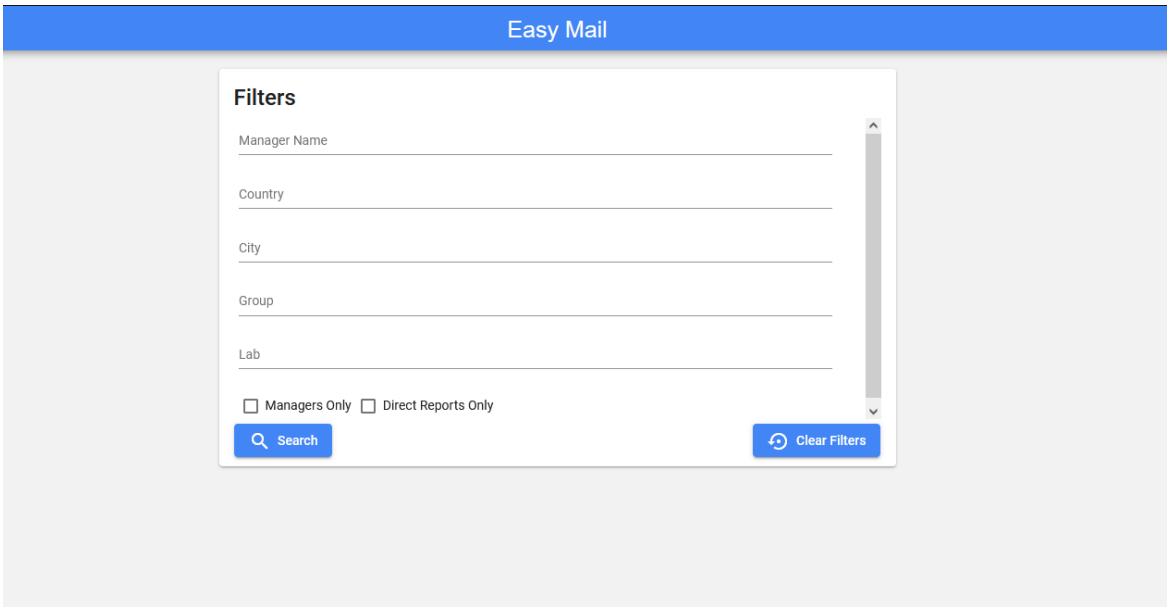


Figura 19. Interfaz de la aplicación Easy Mail

Fuente: Elaboración propia

- Ingresar a "*Gerrit*"
- Realizar una consulta con el primer empleado de lista anterior

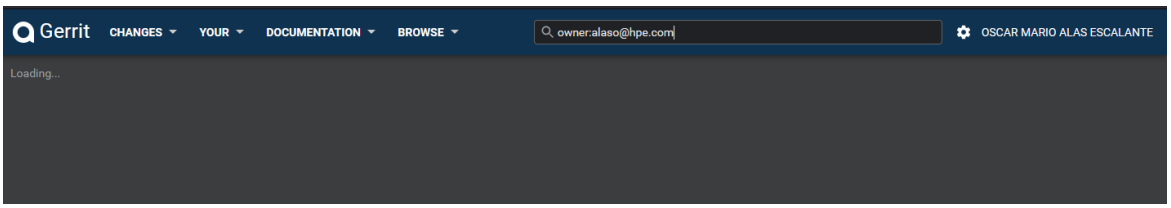


Figura 20 Búsqueda por empleado

Fuente: Elaboración propia

El tiempo con el método actual por una búsqueda sencilla se estima en 45 segundos.

- Repetir el paso anterior por cada empleado de la lista

El aplicativo realiza las tareas anteriores en un solo paso:

- Ingresar al aplicativo y filtrar por "*manager*" y "*status*". Se usó el mismo mánager de la prueba anterior con sus 11 empleados

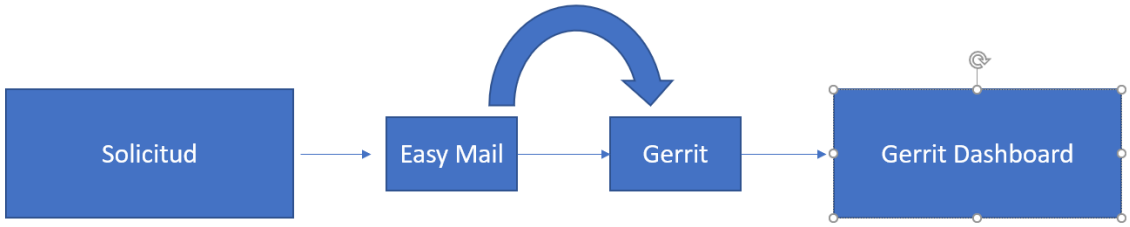


Figura 21 Flujo de trabajo

Fuente: Elaboración propia

- El aplicativo realizará una solicitud a "easy mail" para luego iterar en "*Gerrit*" por cada una de las respuestas obtenidas.
- Para el usuario final el resultado es obtenido de manera más sencilla y rápida que utilizar varias herramientas a la vez.

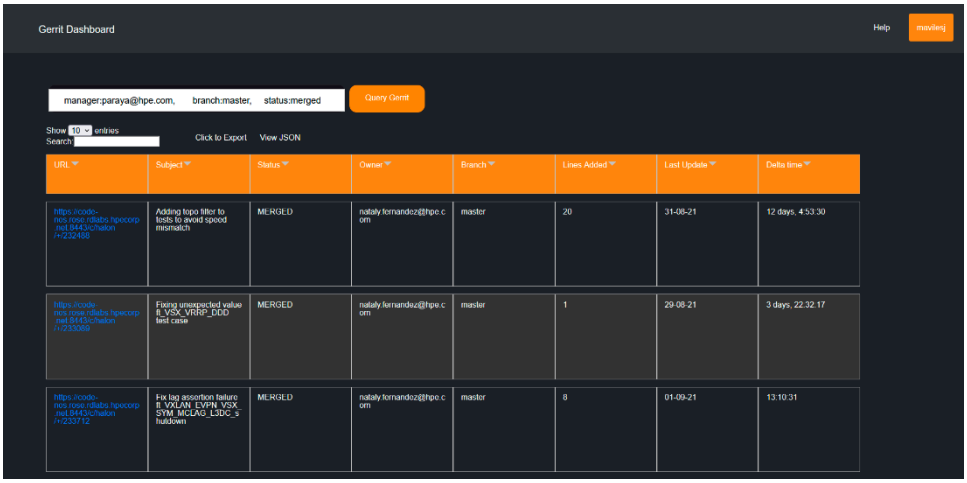


Figura 22 Interfaz del aplicativo desarrollado

Fuente: Elaboración propia

Comparando el tiempo total de ambas pruebas se obtiene el siguiente resultado, en la figura 23 se observa que se realizaron 3 iteraciones de la validación propuesta, tomando un tiempo promedio de 579 segundos.

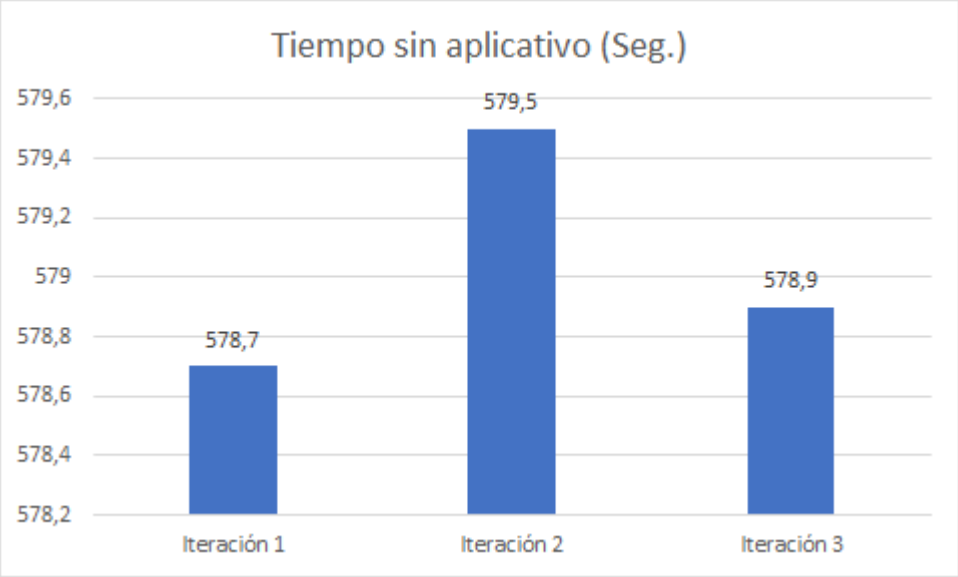


Figura 23. Resultado de pruebas realizadas sin utilizar el aplicativo el aplicativo

Fuente: Elaboración propia

En la figura 24, de la misma manera se hicieron 3 iteraciones utilizando la nueva herramienta, el tiempo utilizando el aplicativo se estima en 45 segundos.

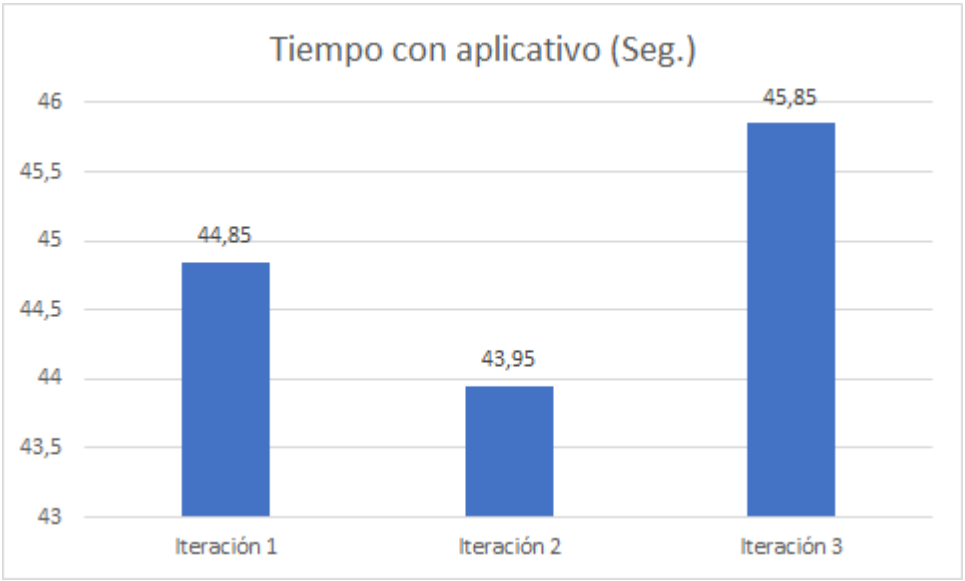


Figura 24 Resultado de pruebas realizadas utilizando el aplicativo

Fuente: Elaboración propia

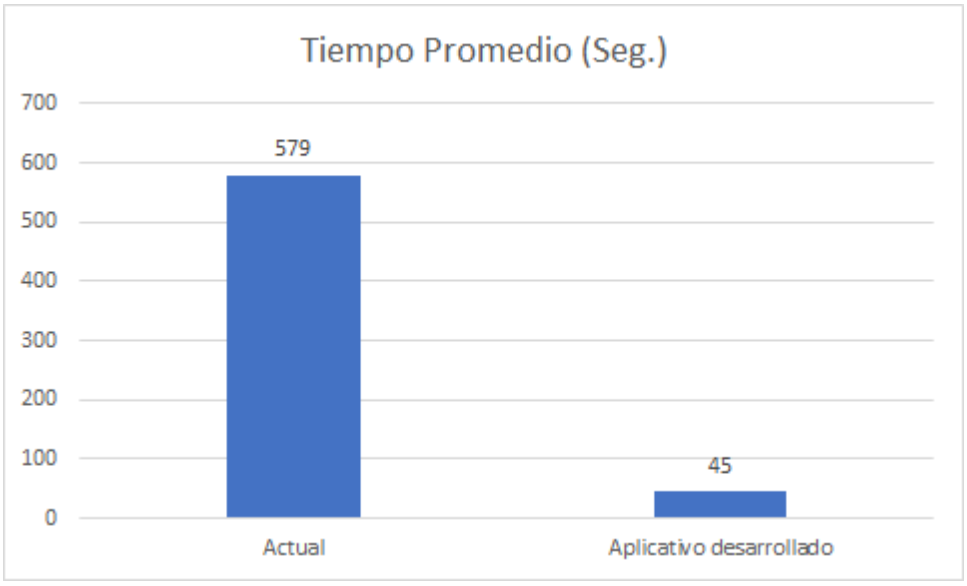


Figura 25. Comparación de tiempo promedio al utilizar el aplicativo

Fuente: Elaboración propia

Se realizaron 3 iteraciones de la prueba, donde se obtuvo un promedio estimado de que la búsqueda de un ingeniero pasaría de tomar 9 minutos y 39 segundos, a tan solo 45 segundos realizando la misma consulta. Además, se necesita ingresar a una sola herramienta sin tener que interactuar con varios sistemas. Se estima un ahorro de un 700% en cuanto a tiempo.

También, la posibilidad de utilizar el aplicativo ofrece muchas más opciones en las consultas como el caso del método del filtro de "mánager" el cual no existe en Gerrit.

CAPÍTULO V
CONCLUSIONES Y RECOMENDACIONES

CAPÍTULO V: CONCLUSIONES Y LIMITACIONES

1. Conclusiones

- Se realizó una investigación a fondo de la herramienta *Gerrit* y su REST API para tener un entendimiento claro de cómo retorna la información, cuáles son los valores que retorna, y cómo se realizan consultas específicas, con el fin de obtener la información filtrada a partir de datos provistos por el usuario.
- Se efectuó todo un análisis en conjunto con los gerentes para conocer todas las necesidades de información por parte de la organización. con el fin de definir el alcance de la aplicación y dejar previstas para expandir el aplicativo en caso de que la unidad de negocio genere nuevos requerimientos.
- Se logró realizar una capacitación con un grupo focal donde se explicaron las mejoras implementadas por medio del aplicativo web desarrollado, además de algunas particularidades de este, como el proceso de exportación de la información a Excel, y se validó la mejora en el tiempo invertido para realizar este proceso con este grupo de ingenieros.
- Se logró cumplir el 100% de los objetivos.

2. Limitaciones

A grandes rasgos, las limitaciones que se presentaron para el desarrollo del software fueron mínimas. Sin embargo, sí tuvieron un gran impacto en el diseño e implementación del aplicativo. A continuación de describen algunas de ellas:

- Uso obligatorio del REST API, lo cual hace que la aplicación deba ser compatible con el protocolo HTTP.
- La exportación de los resultados obtenidos a través del motor de búsqueda solo puede ser exportado en formato CSV con el fin de facilitar la creación de reportes por parte de los gerentes y usuarios en general.
- La herramienta y el filtro de búsqueda por manager solo aplica para el departamento de Aruba, ya que el producto o proyecto que se desarrolla dentro de

la compañía no está disponible para todos los empleados de HPE, sino solamente para los que tienen acceso al proyecto de AOSCX.

- Se posee una dependencia a la aplicación easyMail, lo cual ocasiona que, si el aplicativo “easyMail” falla, el filtro de búsqueda por mánager se vería impactado.

3. Trabajos futuros

El aplicativo fue desarrollado bajo código abierto. Sin embargo, como ha sido desarrollado dentro de la organización, el contenido del mismo no está disponible al público. Su código e implementación pertenecen a Hewlett-Packard Enterprise debido a la cláusula de propiedad intelectual. El código desarrollado se encuentra dentro de los servidores empresariales de *GitHub*.

Debido a lo anterior cualquier trabajo a futuro debe ser implementado dentro de la compañía, siguiendo las prácticas de trabajo establecidas por la compañía, utilizando el REST API de la aplicación de *Gerrit* y el REST API para la aplicación EasyMail o cualquier otra aplicación con la cual se requiera interoperabilidad.

REFERENCIAS

REFERENCIAS

Arubanetworks. (n.d.). Your journey. Your Edge. Disponible en: https://www.arubanetworks.com/company/about-us/

Barrera, A. (n.d). JSON: ¿Qué es y para qué sirve?. Disponible en: https://www.nextu.com/blog/que-es-json/

Chacon, S. (2014). Pro Git (Vol. 2) [1]. Disponible en: https://git-scm.com/book/es/v2/

Chromium.org, (2016). Issue 5190: existing reviewers get cc'd on a new reviewer's notification email. Disponible en: https://bugs.chromium.org/p/gerrit/issues/detail?id=5190#c3

El, F. (2011). 5 tendencias en software empresarial más que probables en los próximos años |Velneo. Disponible en: https://velneo.es/5-tendencias-software-empresarial-mas-probables-los-proximos-anos/

Esaú, A. (2017). Qué es la Metodología Agile. OpenWebinars.net. Disponible en: https://openwebinars.net/blog/que-es-la-metodologia-agile/

García, J. (2018). 10 tendencias en Desarrollo Web para 2018 - Blog de arsys.es. Blog de arsys.es. Disponible en: https://www.arsys.es/blog/programacion/tendencias-desarrolloweb-2018/

Gerrit-review.googlesource.com. (n.d.). Gerrit Code Review - REST API. Disponible en: https://gerrit-review.googlesource.com/Documentation/rest-api.html

Gerrit-documentation. (n.d). Gerrit Code Review - A Quick Introduction. Disponible en: https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.3/intro-quick.html

Git.eclipse.org. (n.d). Gerrit Code Review Product Overview. Disponible en: https://git.eclipse.org/r/Documentation/intro-quick.html

GoogleSource.com. (n.d). Gerrit Code Review – NoteDb Backend. Disponible en: https://gerrit-review.googlesource.com/Documentation/note-db.html

Gracia, L. M. (2017, September 18). ¿Qué es Gerrit? Disponible en: https://unpocodejava.com/2017/09/18/que-es-gerrit/

Hewlett Packard Enterprise internal wiki. (n.d). The Fellowship of the Tools. Disponible en: https://rndwiki.corp.hpecorp.net/confluence/display/hpnevpg/The+Fellowship+of+the+Tools

ISTQB®. (2018). Certified Tester Foundation Level 2018 (CT FL) Syllabus. Disponible en: https://www.istqb.org/documents/pr/ISTQB-releases-CTFL2018.pdf

Json.org. (n.d). Introducción a JSON. Disponible en: http://www.json.org/json-es.html

Kupiainen et al. (2015) Eetu Kupiainen, Mika V. Mäntylä, and Juha Itkonen, Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. Information and Software Technology. 62 (2015), 143-163.

Lehtonen, S. (2015). Metrics for Gerrit code reviews. Disponible en: https://tampub.uta.fi/bitstream/handle/10024/97660/GRADU-1435734321.pdf

Macrotrends.net. (n.d.). Hewlett Packard Enterprise: Number of Employees 2013-2021 | HPE. Disponible en: https://www.macrotrends.net/stocks/charts/HPE/hewlett-packard-enterprise/number-of-employees

Marti. (2009) Gerrit: Google-style code review meets git. Disponible en: https://lwn.net/Articles/359489/

Martín (2011). Gerrit, un sistema de revisión de código muy jugoso. Brigomp.blogspot.com. Disponible en: http://brigomp.blogspot.com/2011/09/gerrit-un-sistema-de-revision-de-codigo.html

Mediawiki.org. (2018). Gerrit - MediaWiki. Disponible en: https://www.mediawiki.org/wiki/Gerrit/es

Milanesio, L. (2018). Gerrit Code Review Plugin - Jenkins Disponible en: https://wiki.jenkins.io/display/JENKINS/Gerrit+Code+Review+Plugin

Orbis Sapienta. (n.d). Qué es JSON. Disponible en: https://obelearningservices.com/cursos/json/lecciones/json-introduccion/temas/json-introduccion-que-es/

Openstack.org. (n.d). Gerrit Code Review - A Quick Introduction. Disponible en: https://review.openstack.org/Documentation/intro-quick.html

Pressman, Roger S., Ingeniería del Software. Vol. I. 2005

Red Hat. (2020). ¿Qué es una API de REST?. Disponible en: https://www.redhat.com/es/topics/api/what-is-a-rest-api

Robinson, D. (2017). The Incredible Growth of Python | Stack Overflow. Disponible en: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Significados. (2021). Tipos de investigación. Disponible en: https://www.significados.com/tipos-de-investigacion/

Stack Overflow. (2018). Stack Overflow Developer Survey 2018. Disponible en: https://insights.stackoverflow.com/survey/2018/

Vasu, S. (2017). Continuous Integration Setup Using Gerrit and Jenkins | | PromptCloud. Disponible en: https://www.promptcloud.com/blog/continuous-integration-setup-gerrit-jenkins

Vélez Manzano, R. (n.d.). Millennials: La generación que define la calidad de las aplicaciones. Universidad Cenfotec. Disponible en: https://www.ucenfotec.ac.cr/blog/millennials-la-generacion-que-define-la-calidad-de-las-aplicaciones

ANEXOS

Anexo Gerrit API

https://gerrit-review.googlesource.com/Documentation/rest-api.html

Anexo Carta del tutor

Anexo: Carta de Tutor

Heredia, 29 de mayo de 2018

Señores
COMISIÓN DE TRABAJOS FINALES DE GRADUACIÓN
Presente

Estimados señores:

Por medio de la presente, el suscrito **Jose Pablo Calvo Suarez**, portador de la cédula de identidad número **111350163**, manifiesto mi anuencia en ser TUTOR del Proyecto de Graduación denominado **"Facilitar la gestión de revisión de código, pruebas y documentación a los gerentes, mediante un aplicativo basado en un nuevo modelo de metadata de Gerrit."** elaborado por los estudiantes **Oscar Alas Escalante**, cédula de identidad **402190360** y **Miguel Avilés Jenkins**, cédula **114140211** así como aceptación de las funciones y responsabilidades que dicha labor implica.

Atentamente,

Ms.c Jose Pablo Calvo Suarez
Académico
Universidad Nacional

Anexo III, pág. 4

Anexo Carta de la empresa

Hewlett Packard
Enterprise

Anexo: Carta de la empresa

Heredia, 5 de Junio de 2018

Señores
Comisión de Trabajos Finales de Graduación
Presente

Estimados señores:

En mi calidad de Gerente de la empresa **Hewlett Packard Enterprise**, muy respetuosamente, nos permitimos manifestarle nuestro apoyo incondicional a los ingenieros **Oscar Alas Escalante** y **Miguel Avilés Jenkins** para que desarrolle el proyecto de graduación denominado **"Facilitar la gestión de revisión de código, pruebas y documentación a los gerentes, mediante un aplicativo basado en un nuevo modelo de metadata de Gerrit".**

No omitimos manifestarles nuestro agradecimiento por la aprobación del citado proyecto, ya que para esta empresa es una valiosa oportunidad contar con el apoyo académico e investigativo de la Universidad y sus estudiantes.

Sin otro particular,

Jonathan Carlo Loo
Gerente
Hewlett Packard Enterprise