

Universidad de Valladolid

Máster universitario

Ingeniería Informática



Trabajo Fin de Máster

Gamificación en entornos de producción de la industria del software para favorecer los indicadores de calidad y reducir la deuda técnica

Realizado por **D. VÍCTOR GONZÁLEZ NÚÑEZ**

Universidad de Valladolid

27 de septiembre de 2021

Tutora: Dra.D^a Yania Crespo González-Carvajal

Resumen

En los entornos de producción de la industria del *software*, es muy importante el control del nivel de calidad de los productos desarrollados. En la actualidad se están aplicando estrategias de gamificación en diversos ámbitos, como la educación o la Ingeniería del *Software*, como factor motivador. Existen diferentes tipos de estrategias aplicables a los equipos de desarrollo para mejorar su desempeño y su compromiso organizacional, todo ello encaminado a una mejora del producto *software*.

El concepto de deuda técnica, como indicador relevante para los procesos de producción, influye en el coste implícito del desarrollo. Existen herramientas tecnológicas para la mejora de los procesos de gestión de proyectos *software* con el objetivo del control y reducción de la deuda técnica, favoreciendo la calidad.

En este trabajo se propone una solución de integración de la aplicación de control de calidad de código SonarQube con el sistema de configuración de versiones GitLab basado en un entorno automatizado de producción de *software*. Siguiendo la metodología DevOps, que permite la integración y distribución continua (CI/CD), se implementa una propuesta de estrategia de gamificación con el objetivo del control del desarrollo y la mejora de la calidad.

Descriptores

Gamificación, control de calidad, deuda técnica, industria del *software*, CI/CD, DevOps, GitLab, SonarQube

Abstract

In the production environments of the software industry, it is very important to control the quality level of the developed products. Gamification strategies are currently being applied in various fields such as education or software engineering as a motivating factor. There are different types of strategies applicable to development teams to improve their performance and organisational commitment, all aimed at improving the software product.

The concept of technical debt, as a relevant indicator for production processes, influences the implicit cost of development. There are technological tools for the improvement of software project management processes with the aim of controlling and reducing technical debt, favouring quality.

This paper proposes a solution for integrating the SonarQube code quality control application with the GitLab version configuration system based on an automated software production environment. Following the DevOps methodology, which allows continuous integration and distribution (CI/CD), a gamification strategy proposal is implemented with the aim of development control and quality improvement.

Keywords

Gamification, quality control, technical debt, software industry, CI / CD, DevOps, GitLab, SonarQube

*“El aprendizaje es lo único que la mente nunca agota,
nunca teme y de lo que nunca se arrepiente.”*

Leonardo Da Vinci

A mi tutora Yania, por su guía y generosa ayuda en este trabajo.

A mis profesores, por todo lo que me han enseñado durante el máster.

A quien con su contribución me ha ayudado a enriquecer el contenido de este trabajo.

A mis compañeros de clase, por su fructífera colaboración.

*Gracias a mi familia, en especial a mis padres, y a mis amigos,
por todo su apoyo, paciencia y comprensión.*

Índice

1	Introducción	10
1.1	Motivación	10
1.2	Objetivos	10
1.3	Organización de la memoria	11
2	Plan de Trabajo.....	12
2.1	Planificación	12
2.1.1	Fases.....	12
2.2	Recursos humanos	14
2.3	Recursos hardware y software.....	14
2.4	Estudio de costes.....	15
2.4.1	Costes de recursos humanos.....	15
2.4.2	Coste de recursos materiales	15
2.5	Coste total	15
3	Estado del arte	17
4	Entorno tecnológico	22
4.1	Gestión de proyectos software	24
4.1.1	Trabajo en equipo	24
4.1.2	Calidad en los proyectos software y entorno tecnológico.....	25
4.2	Sistemas de Control de Versiones.....	27
4.2.1	Tipos	28
4.2.2	La importancia del control de versiones	29
4.3	Git.....	30
4.3.1	Características	30
4.4	Gitlab	30
4.4.1	GitLab vs GitHub.....	32
4.4.2	GitLab CI/CD	32
4.4.3	<i>Pipelines</i> y GitLab Runner.....	33
4.5	SonarQube.....	34
4.5.1	SonarLint	37
4.5.2	Integración con Gitlab	37
4.5.3	SonarRunner.....	37
4.5.4	Otras herramientas de análisis de código	38
5	Propuesta de solución.....	39
5.1	Detalle de la solución	41
5.1.1	Análisis de requisitos.....	41

5.1.2	Diseño de la solución.....	42
5.1.3	Implementación y pruebas. Escenarios.....	50
6	Conclusiones y trabajo futuro	54
6.1	Valoración global de la actividad	55
6.2	Futuras líneas de trabajo.....	55
7	Bibliografía	57
8	Apéndices	61
8.1	Apéndice 1 - Guía de instalación, configuración y uso de GIT	61
8.2	Apéndice 2 – Guía de instalación y configuración de GITLAB	64
8.3	Apéndice 3 – Guía de instalación de SonarQube	66
8.4	Apéndice 4 – Guía de instalación y configuración de SonarQube Runner.....	68
8.5	Apéndice 5 - Características de los servidores.....	70

Índice de ilustraciones, gráficos y tablas

Ilustración 1 - Diagrama de Gantt: Programación fases. Elaboración propia.....	13
Ilustración 2 - Taxonomía de tipos de participantes de A. Marzewski.....	17
Ilustración 3 - Modelo de comportamiento de Fogg.	18
Ilustración 4- Cuadrante de tipos de deuda técnica.	20
Ilustración 5 - Estrategias para la reducción de la deuda técnica.	21
Ilustración 6 - Vista con los resultados de un análisis con SonarQube.	23
Ilustración 7 - CHAOS report. Resolución en los proyectos software desde 2011 a 2015.....	27
Ilustración 8 - Sistema de control de versiones centralizado.	28
Ilustración 9 - Sistema de control de versiones distribuido.....	29
Ilustración 10 - Interfaz de GitLab. Vista de la gestión de un proyecto..	31
Ilustración 11- Ejemplo de workflow con GitLab. Pipeline y GitLabRunner.	33
Ilustración 12 - GitLab CI example. Elaboración a partir de template GitLab.com.....	33
Ilustración 13- Interfaz de SonarQube. Resumen de análisis realizado.....	34
Ilustración 14- Interfaz de SonarQube. Vista de los resultados de análisis	35
Ilustración 15 - Interfaz de SonarQube. Vista de líneas duplicadas: Número y porcentaje	36
Ilustración 16 - Funcionamiento de SonarRunner.	37
Ilustración 17 - Interfaz de SonarQube. Integración con otros entornos.	44
Ilustración 18 - Vista de SonarQube. Configuración integración con GitLab.....	45
Ilustración 19 - Vista de SonarQube. Listado proyectos GitLab.....	45
Ilustración 20 - Interfaz de SonarQube. Gestión de Webhooks.....	46
Ilustración 21 - Diagrama de la solución propuesta. Lanzamiento del análisis.	47
Ilustración 22 - Diagrama de la solución propuesta. Consulta del ranking.....	48
Ilustración 23 - Vista del ranking. Prototipo.....	49
Ilustración 24 - Vista proyecto GitLab - Solicitud de fusión	62
Ilustración 25 - Vista del proyecto en GitLab. Rama actualizada.	62
Ilustración 26 - Vista edición de Claves SSH en GitLab.....	63
Gráfico 1 - Dedicación temporal. Elaboración propia.....	14
Tabla 1 - Coste total estimado para el proyecto. Elaboración propia.....	15
Tabla 2 - Características Servidor HPE ProLiant Microserver.....	16
Tabla 3 - Servidor HPE ProLiant DL380 Gen10	16

1 Introducción

1.1 Motivación

En los procesos de producción de *software*, una de las partes fundamentales es el control de la calidad del código y la gestión de la deuda técnica. En Ingeniería del *Software* tiene gran importancia su acometida, incorporando como elementos fundamentales la organización, la motivación y los conocimientos de los equipos de desarrollo a través de la aplicación de diferentes estrategias para la construcción de productos *software* de calidad.

En los entornos de la industria del *software* es muy determinante el factor humano. Los desarrollos necesitan de la formación de equipos multidisciplinares integrados por miembros que participen en las distintas fases del proyecto. Los miembros de estos equipos tienen que estar motivados y comprometidos tanto en los objetivos organizacionales como en las propias tareas que tienen asignadas. Esta motivación es uno de los retos que presenta la gestión de la calidad en Ingeniería del *Software*.

Es muy relevante el estudio de los conceptos y metodologías más representativas relacionadas con estrategias innovadoras que implican una mejora de la calidad del *software* y la reducción de deuda técnica. Hay factores que influyen determinantemente como la motivación de los desarrolladores que participan en el proyecto. A través de medidas de recompensa, el fomento de la comunicación o la mejora organizativa se puede conseguir influir en la calidad de los desarrollos, lo que se traduce en una mejora de la calidad del código.

Las metodologías de mejora de la gestión del *software* a través de la aplicación de diferentes estrategias de gamificación relacionadas con el ámbito de la calidad de los desarrollos ayudan a la reducción de la deuda técnica. Su gestión está relacionada con los sistemas de medición y control de la calidad a partir de diversos indicadores.

Por todo lo anterior, se propone elaborar una propuesta de estrategia de gamificación para su implementación en un entorno automatizado de producción de *software* basado en DevOps CI/CD, que permita la medición de la calidad del código con ayuda de una herramienta de análisis estático y que admita una integración y distribución continua.

1.2 Objetivos

El primer objetivo de este trabajo es el estudio de las diferentes técnicas y estrategias de gamificación en Ingeniería del *Software* aplicadas a la gestión de proyectos *software*. Se profundizará en el concepto de deuda técnica y en la calidad del *software*. A partir de la medición de diferentes métricas, se puede valorar la calidad del código, muy útil en la creación de productos *software*.

El segundo objetivo es la investigación de entornos tecnológicos automatizados para implementar una propuesta de estrategia de gamificación integrada en el marco de la producción de *software* basado en la integración y distribución continua. Existen diferentes herramientas tecnológicas para el control de calidad del *software* que pueden ser integradas con entornos que incluyan control de versiones y una integración y distribución continua. Se debe tener en cuenta la configuración de los proyectos, los grupos y los miembros de estos según el rol que tengan designado. Todo ello para realizar una propuesta de solución como implementación en un entorno de producción de la industria del *software*.

1.3 Organización de la memoria

En esta memoria se detalla el proceso de investigación y estudio realizado en este trabajo para la creación de una estrategia de gamificación para el control de la deuda técnica y de la calidad en entornos de la industria de la producción de *software*.

En primer lugar, se incluye una introducción como motivación de la realización del trabajo, incluyendo una definición de los objetivos. A continuación, se expone el plan de trabajo seguido, con sus diferentes fases reflejadas en un cronograma y explicando los recursos utilizados y un estudio de costes del proyecto.

Después se ha realizado una exposición teórica sobre los conceptos relacionados con el trabajo, empezando por el concepto de gamificación, pasando por las estrategias motivacionales y finalizando con la importancia de la calidad del *software* y la deuda técnica.

A continuación, se definen los elementos principales para la creación de un entorno tecnológico teniendo en cuenta la propuesta de solución elaborada. Luego se detalla y concreta la propuesta de solución, exponiendo la implementación de un entorno Gitlab integrado con SonarQube.

Finalmente, se hace un comentario sobre los resultados obtenidos, las conclusiones del trabajo y las propuestas de trabajo futuro. También se anexan un conjunto de apéndices con instrucciones para crear los entornos que se han propuesto.

2 Plan de Trabajo

Para la realización de este trabajo se ha elaborado una planificación inicial para organizar la tarea de investigación y estudio, la confección de los contenidos de la memoria y la creación de la propuesta de solución. En este capítulo se detallan los aspectos relacionados con la gestión del tiempo en las tareas de elaboración, así como los recursos tanto humanos como de *hardware* y *software*. Al final se incluye una estimación de los costes que supone la puesta en marcha del proyecto.

2.1 Planificación

Este trabajo fin de máster se ha realizado entre los meses de febrero y septiembre de 2021. La idea inicial se concibió en el verano de 2020, pero por motivos laborales se pospuso el inicio del trabajo de investigación. Para elaborar esta memoria se han aprovechado los conocimientos adquiridos y las conclusiones del proyecto de investigación de la asignatura de I+D+I del Máster de Ingeniería Informática, en la que se profundizó en los aspectos que han servido como sustento teórico para este trabajo.

Una de las tareas transversales para la realización de este trabajo ha sido la recopilación de artículos de investigación y otros textos técnicos consultados, a la vez que la elaboración de la memoria. Durante todo el proceso se ha realizado una tarea iterativa de revisión, para ir mejorando los contenidos según se obtenían nuevos aprendizajes.

La investigación se ha realizado con una dedicación aproximada de unas 4 horas al día, durante 7 meses, entre marzo y septiembre de 2021. Todo este proceso ha contado con el acompañamiento y la orientación del tutor académico en las sesiones de revisión del trabajo.

2.1.1 Fases

El trabajo se ha realizado en dos fases:

- **Primera fase:** en la que se elaboró el estado del arte sobre la gamificación, las estrategias de motivación, la calidad del código y la deuda técnica.
- **Segunda fase:** en la que se realizó un estudio de las herramientas de control de calidad, de integración continua y se realizó la propuesta de implementación.

Previo al inicio de la realización del trabajo, se realizó una tarea de investigación general sobre contenidos y herramientas que pueden cumplir los requisitos y objetivos de este trabajo, haciendo una tarea de revisión de las propuestas tecnológicas más innovadoras. Con esta información se realizó el primer plan de trabajo, incluyendo la definición de tareas y su estimación temporal.

En la primera fase, el método de trabajo que se ha utilizado ha sido la realización de diferentes fases de una investigación: planificación, ejecución, procesamiento e informe. Inicialmente, se ha definido la planificación de la investigación y la preparación de documentación teniendo en cuenta la dedicación total y los plazos. Posteriormente, se ha elaborado un estudio previo sobre los temas de cada una de las fases, realizando una recopilación de fuentes a partir de

2. Plan de Trabajo

repositorios de artículos como Scopus¹ o Web of Science.²

En la segunda fase, se realizaron las tareas de estudio de las herramientas para el control de calidad y de integración, que sirvieron como base para la propuesta de implementación. En esta fase se han ampliado las fuentes con documentos técnicos relacionados con el marco de producción *software*. Además, se realizó la propuesta de solución y se culminó la elaboración de la memoria, añadiendo las conclusiones del trabajo e incluyendo apéndices para ampliar la información. Finalmente, se ha elaborado una presentación para la sesión de exposición del trabajo.

A continuación, se describen las tareas que se han realizado, junto a su estimación temporal. El trabajo se ha dividido en tareas de realización según los objetivos del trabajo.

1. Investigación inicial. Elaboración de la planificación.
2. Investigación sobre estrategias de gamificación.
3. Investigación sobre tipología de deuda técnica.
4. Investigación sobre sistemas de calidad: indicadores y métricas.
5. Investigación sobre herramientas de gestión de la configuración y de la calidad de *software*.
6. Estudio de entornos de producción de la industria del *software*.
7. Diseño de la propuesta: implementación.
8. Revisión. Recopilación y reelaboración de la información para la memoria. Conclusiones y apéndices.
9. Elaboración de la presentación del trabajo.

En el siguiente diagrama de Gantt se expone la programación en las diferentes fases de elaboración del trabajo:

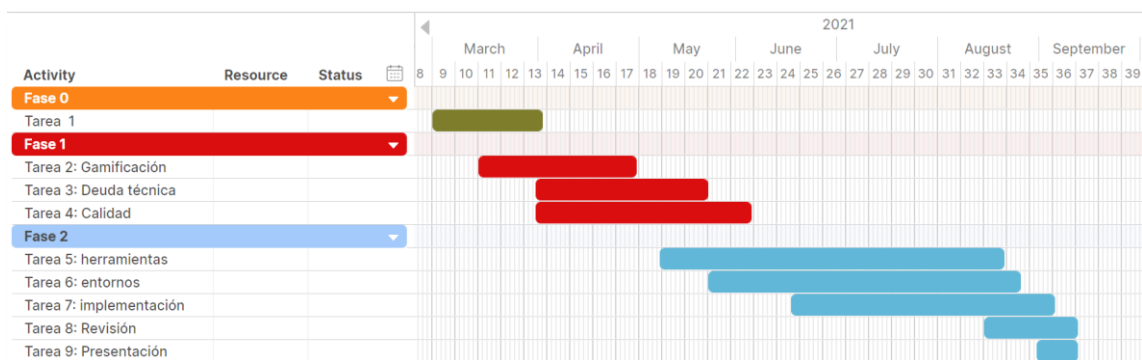


Ilustración 1 - Diagrama de Gantt: Programación fases. Elaboración propia.

Al finalizar el trabajo, se tuvo que hacer una revisión de los plazos, ampliando el tiempo de realización de tareas, ya que la investigación fue ambiciosa y se profundizó en aspectos que en un principio no se iban a acometer. En el caso de la definición de la propuesta de solución, también se le dedicó más tiempo del planteado inicialmente al ampliar la investigación de herramientas tecnologías para realizar la implementación. En el diagrama expuesto aparece la dedicación temporal real.

¹ Scopus Search. World largest abstract and citation database of research literature www.scopus.com/

² Web of Science. WOS FECYT. wos.fecyt.es/

2. Plan de Trabajo

La mayor parte del tiempo planificado se ha dedicado a la investigación sobre el estado actual de las herramientas de configuración de *software*, los entornos de producción y la realización de la propuesta de implementación.

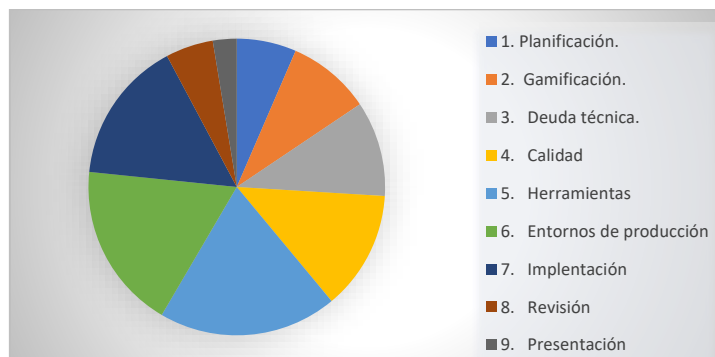


Gráfico 1 - Dedicación temporal. Elaboración propia.

2.2 Recursos humanos

Este proyecto se ha realizado de manera individual, pero para una realización completa, que incluya la implementación, exigiría un equipo con un gestor de proyectos, un administrador de sistemas, un programador de aplicaciones y un testeador.

2.3 Recursos hardware y software

Los recursos *hardware* que se han utilizado para elaborar el trabajo han sido los siguientes:

- Ordenador sobremesa PC All in One HP 24-df0006ns
 - SO Windows 10 Home 64 bits
 - Procesador AMD Ryzen™ 3 3250U (2,6 GHz, 4 MB de caché L3, 2 núcleos)
 - Gráficos AMD Radeon™
 - Memoria 8 GB DDR4-2400 SDRAM (1 x 8 GB)
 - Disco duro SSD SATA de 256 GB
 - Pantalla con retroiluminación WLED y FHD IPS de 23,8 pulg.

En cuanto al *software*, los recursos utilizados han sido los siguientes:

- Paquete Office 365 para empresas, utilizando el editor de textos Word, la hoja de cálculo Excel y el editor de presentaciones PowerPoint. Se han utilizado para la gestión de los datos, la elaboración de la memoria, la creación de tablas, gráficos y de la presentación.
- Mendeley, para la gestión de referencias bibliográficas.
- Adobe Reader XI, para la visualización de documentos PDF.
- Navegadores web Google Chrome Versión 93.0.4577.63 y Mozilla Firefox versión 91.0.2.
- Notepad++, para la edición de código.
- Tomsplanner (free), para la elaboración de diagramas.

2. Plan de Trabajo

- Gimp, para la edición de ilustraciones.
- Canva, para la edición de ilustraciones.
- Oracle VM Virtual Box 6.1, para la creación de máquinas virtuales para el diseño y elaboración de la solución. Para las máquinas se ha utilizado distribuciones de SO de Ubuntu 20.04 y de CentOS 8.
- SonarQube versión 9.0.1, de julio de 2021.
- Gitlab versión 14.1.3 de agosto 2021.
- Trello, como planificador de tareas del trabajo.
- Evernote, como sistema de organización de información sobre el trabajo.
- Microsoft Teams, como herramienta de almacenamiento de archivos en la nube: documentación, borradores, artículos, código, etc., y también como tablón de comentarios y sistema de videoconferencia para realizar el seguimiento del trabajo.

2.4 Estudio de costes

A partir de los recursos necesarios, tanto *hardware* como *software*, se realiza una estimación de los costes para la implementación. En el caso de este trabajo, para el diseño de la solución se ha utilizado un entorno de virtualización, pero en este estudio de costes se incluyen los datos para un alojamiento en servidores profesionales, muy recomendable para un mejor rendimiento del sistema. Otra opción sería utilizar sistemas de servidores alojados en proveedores externos, que implica un coste mensual o anual.

2.4.1 Costes de recursos humanos

En este trabajo, el autor ha asumido todos los roles del proyecto. Se pondera un salario como analista *polivalente* teniendo una cuenta una duración de 7 meses, con una dedicación de unas 20 horas semanales, lo que se traduciría en unas 150 jornadas laborales y 600 horas en total. Se estima el salario de un profesional de con estas características en 42.000 euros brutos por año, significando un total de **12.250 euros**; 20,42 euros la hora de trabajo.

2.4.2 Coste de recursos materiales

- Ordenador sobremesa PC All in One HP 24-df0006ns (licencia Windows 10 incluida): **650 euros**
- Licencia Paquete Office 365 (anual): **69 euros**

2.5 Coste total

La estimación del coste total del proyecto sería de 12.969€.

Coste personal	12.250,00 €
Coste material	719,00 €
Total	12.969,00 €

Tabla 1 - Coste total estimado para el proyecto. Elaboración propia

2. Plan de Trabajo

También se ha realizado una revisión de servidores profesionales para una mejor implementación de la solución propuesta. A continuación, se muestra las características de dos servidores, según el tamaño de la empresa donde se va a implementar la solución. El primero es para un ámbito de una pequeña o mediana empresa, y el segundo para una gran empresa.

Estimación del precio sobre los servidores:

Organización pequeña o mediana:

• Servidor HPE ProLiant MicroServer Gen10 Plus	
Características:	
	Procesador Intel Xeon E-2224
	16 GB de memoria RAM
	HDD de 1 TB SATA
	Bahías para unidades de factor de forma grande sin conexión <i>hot-plug</i>
	Fuente de alimentación externa de 180 W
	Tienda HP (directamente desde el proveedor) ³
Precio:	Desde 616,00 € impuestos no incluidos.

Tabla 2 - Características Servidor HPE ProLiant Microserver

Organización grande:

Servidor HPE ProLiant DL380 Gen10	
Características:	
	Procesador Intel Xeon Gold 3,2 GHz
	32 GB DDR4-SDRAM
	HDD 60 TB
	Bastidor (2U)
	Tienda HP (directamente desde el proveedor) ⁴
Precio:	Desde 1.782,00 € impuestos no incluidos.

Tabla 3 - Servidor HPE ProLiant DL380 Gen10

A estos costes se le añadirían los costes de mantenimiento: equipamiento, CPD de alojamiento, electricidad, etc. En los apéndices de esta memoria se muestra el detalle ampliado con todas las características de estos servidores.

³ Servidor HPE ProLiant MicroServer Gen10 Plus. HP Enterprise Online Shop. <https://bit.ly/3twWC3b>

⁴ Servidor HPE ProLiant DL380 Gen10. HP Enterprise Online Shop <https://bit.ly/3E1MN1Q>

3 Estado del arte

En esta sección se realiza una recopilación de artículos y textos técnicos relacionados con el objeto de estudio de esta memoria. Se comienza por qué es y para qué sirve la gamificación en estrategias aplicadas a la Ingeniería del *Software*. También se recoge el concepto y tipología de deuda técnica y de la calidad de *software* como proceso de desarrollo a partir de indicadores que determinan dicha calidad.

En el proceso de desarrollo de *software* por parte de equipos de trabajo es recomendable basarse en métodos y técnicas de Ingeniería del *Software* para conseguir una buena calidad del código. Una parte relacionada con la calidad es el control de la deuda técnica. Esto se debe tener en cuenta en el equipo de creación de *software* debido a que un desarrollo de baja calidad con una gran deuda técnica tiene efectos negativos en el futuro.

Actualmente se producen dificultades en los procesos de creación de *software*, como pueden ser los desarrollos en poco tiempo, con hitos de entrega urgente o con una errónea priorización de las tareas organizada por los gestores del proyecto, incluso por la baja importancia de la calidad y la deuda técnica por parte de los desarrolladores. Una de las opciones para poder mejorar este aspecto es la implementación de estrategias de gamificación en los desarrollos.

Se ha realizado un estudio sobre el concepto de gamificación y los ámbitos en los que se han aplicado a través de diferentes estrategias. La gamificación es el proceso activo que usa mecánicas y dinámicas del juego en entornos no lúdicos con el objetivo de cambiar el comportamiento y aumentar la motivación entre los participantes [1].

Toda estrategia de gamificación tiene elementos comunes. Primero, las dinámicas, que son los objetivos que se quieren conseguir con la estrategia. También son elementos relevantes las mecánicas, que son las acciones que se van ejecutando durante el proceso de realización de la gamificación. Las mecánicas pueden ser de distintos tipos: cooperación, retos, recompensa e incentivos, etc. Toda mecánica conduce a conseguir el objetivo para el que hemos diseñado la gamificación.

Para crear una buena estrategia de gamificación hay que tener en cuenta los participantes que van a formar parte. Andrzej Marczewski realizó una taxonomía de tipos de participantes como exploradores, asesinos, triunfadores o socializadores [2]. Estos cuatro son los principales, pero el autor también añade a los disruptores y los filántropos, con lo que llega a una taxonomía que detalla seis perfiles de participantes.

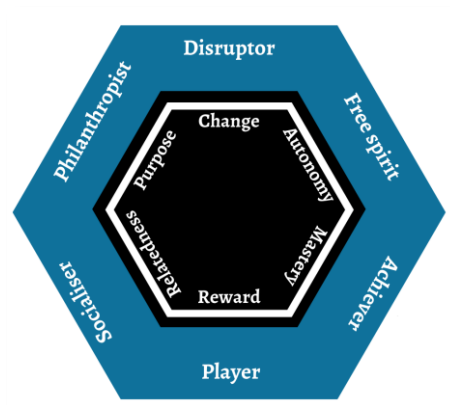


Ilustración 2- Taxonomía de tipos de participantes de A. Marczewski. Elaborado a partir de [2]

3. Estado del arte

En la aplicación de las estrategias de gamificación también hay que tener en cuenta los factores psicológicos. La motivación de los miembros de una organización está relacionada con su conducta y su propensión psicológica. Se pueden destacar dos modelos psicológicos que influyen en la gamificación: el modelo de Fogg y la teoría de la autodeterminación.

En primer lugar, el modelo de Fogg [3] trata de los elementos conductuales, teniendo en cuenta tres factores: la motivación para realizar la conducta, la habilidad para poner en marcha la conducta y el disparador, que es la situación apropiada para realizar la tarea.

En cuanto al segundo modelo, la teoría de la autodeterminación [4], trata sobre las necesidades innatas que permiten un funcionamiento óptimo y un crecimiento personal, y que provocan un desarrollo de las motivaciones intrínsecas en el proceso de realización. Se relaciona con la necesidad de relacionarnos, de ser competente y de la autonomía para la autorregulación conductual destinada a mejorar el bienestar personal y el desempeño de las personas en las organizaciones.

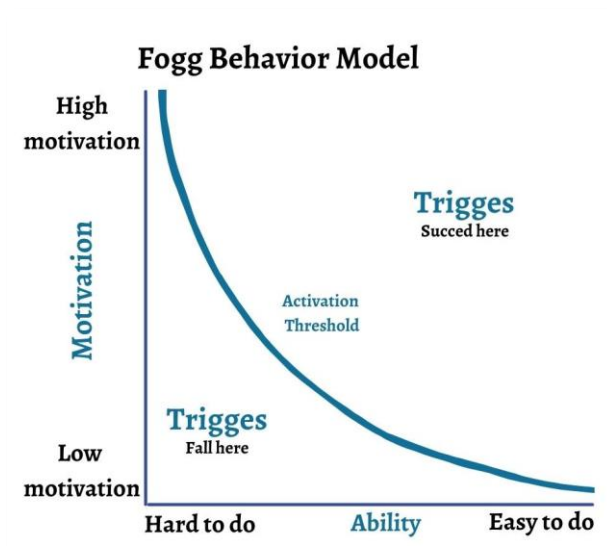


Ilustración 3 - Modelo de comportamiento de Fogg. Elaboración a partir de [3]

La gamificación es un proceso innovador. Muchas consultoras están implementando estrategias de gamificación aplicadas en sus procesos. Por ejemplo, la consultora PricewaterhouseCoopers utiliza estrategias de este tipo para campañas de *marketing* para aumentar la innovación y el compromiso del cliente [5].

En cuanto al ámbito de la Ingeniería del *Software* [6], la aplicación de estrategias de gamificación en entornos de producción de *software* tiene ventajas y desventajas tanto para el desarrollo como para la organización en la que se aplican. La principal ventaja está relacionada con la motivación de los desarrolladores. Con la gamificación se potencia la iniciativa para mejorar los procesos de desarrollo del *software*, creando nuevas estrategias de diseño y consiguiendo códigos de mayor calidad.

Otra ventaja es que se promueve la colaboración y la participación de todos los miembros de la organización. Con ello se mejora el conocimiento de los procesos y servicios de negocio en los que está implicada la organización. Colateralmente, mejora la comunicación entre los miembros, así como su implicación y rendimiento en los proyectos en los que participan [7].

En cuanto a las desventajas, la aplicación de estrategias de gamificación puede tener dificultades en su implementación. Primero, porque existe un tiempo para preparar y adaptar los procesos de desarrollo, del que muchas organizaciones no disponen debido a un elevado ritmo de trabajo. También se puede producir una mala implementación, con un exceso de elementos gamificadores que provoquen una *sobre-gamificación* [8].

Otra desventaja es la dificultad de mantener el interés de los desarrolladores. El impacto inicial puede ser bueno, pero con el paso del tiempo puede perder el efecto motivador. Además, existe el riesgo de que exista una excesiva competitividad, dejando de lado los objetivos de la estrategia de gamificación.

Los ámbitos de la Ingeniería del *Software* donde se pueden aplicar estas estrategias son áreas como la gestión de procesos y metodologías de desarrollo, la gestión del conocimiento, calidad, la realización de pruebas, la metodología de evaluación o la gestión de proyectos [9].

Por ejemplo, en el ámbito de las metodologías ágiles, se está implantando para ayudar a aplicar buenas prácticas. Están apareciendo términos como "*Agile Gamification*" que se refiere a la aplicación de estrategias de gamificación en entornos ágiles, con un impacto muy positivo de mejora en los desarrollos [10].

Existen estudios en los que se afirma que el área de la Ingeniería del *Software* que tiene un mayor número de estrategias de gamificación es el de desarrollo de *software* [6, 11, 12]. Además, estos estudios indican que su aplicación en diferentes ámbitos obtiene resultados positivos entre los participantes.

Uno de los ámbitos donde existe una mayor aplicación de las estrategias de gamificación es el educativo [13]–[15]. En concreto, en el área de Ingeniería del *Software* se utilizan estrategias para la enseñanza de conceptos relacionados y para el aseguramiento de la calidad de los desarrollos. Actualmente ya se están aplicando en programas de entrenamiento de aprendizaje de lenguajes de programación. Por ejemplo, se emplea en herramientas online como Codecademy o Khan Academy. También se ha aplicado en entornos como la industria del turismo, para la organización de empresas de este tipo, así como la promoción y el *marketing* de destinos turísticos [16].

A continuación, se ha estudiado el concepto de calidad en Ingeniería del *Software*, así como los indicadores para medirla, relacionándolo con el concepto de deuda técnica y sus las diferentes tipologías.

La deuda técnica se produce por deficiencias en los desarrollos debidos a falta de tiempo o de presupuesto. Los desarrolladores a veces realizan un código sin el tiempo y esfuerzo suficiente, y este ahorro se paga con deuda. El concepto de deuda técnica [17] lo introduce Ward Cunningham en 1992, relacionándolo con el coste que se produce en los desarrollos de *software* al presentar carencias en el código debido a una entrega temprana. Y a partir de esta definición, Martin Fowler [18] propuso la subdivisión de la deuda técnica en cuatro tipos, que representó en el cuadrante de deuda técnica. En el mismo se define la deuda imprudente, la deuda prudente, la deuda deliberada y la deuda involuntaria.

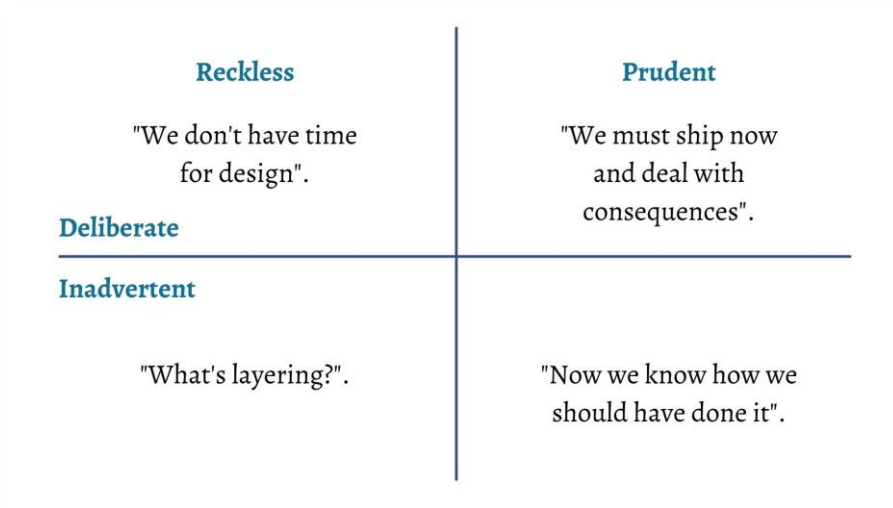


Ilustración 4- Cuadrante de tipos de deuda técnica. Elaboración a partir de [18]

La importancia de mantener la deuda técnica baja está relacionada con la productividad y la calidad de las aplicaciones [19,20]. Es un hándicap para las organizaciones que sus trabajadores estén familiarizados con el concepto y las implicaciones negativas relacionadas. La deuda técnica se puede gestionar de varias formas, siendo un factor importante la priorización de la deuda técnica respecto al *backlog* [21]. Uno de los problemas más habituales es el *shadow backlog*: a partir de errores que aparecen en los desarrollos, estos se traducen en más tiempo de trabajo, pero no se terminan de acometer para cumplir plazos.

Existen estrategias para una mejor gestión que las organizaciones pueden usar para inculcar la importancia de la reducción de la deuda técnica y los comportamientos en los desarrollos de *software*.

En uno de los estudios más relevantes se presentan cuatro tipos de estrategias para reducir la deuda técnica a partir de un modelo de cuadrante de gestión de la deuda junto con recomendaciones sobre cómo implementar tales estrategias en la práctica[22]. En primer lugar, presentan la estrategia alentadora, por la que la motivación se realiza a través del líder del equipo, teniendo en cuenta la concienciación del problema, la comunicación entre los miembros del equipo o la priorización de las tareas de control de deuda.

La segunda es la de la penalización, donde se aplica una consecuencia negativa para el trabajador cuando se detecta que el umbral de deuda técnica se rebosa. La tercera es la del forzamiento, con el establecimiento de normas y requisitos que se deben cumplir a partir metodologías que diseña la organización.

La última se basa en la gratificación. Esta estrategia premia la reducción de umbrales objetivos, a través de incentivos económicos o de otro tipo. El estudio desvela que los trabajadores no están a favor de este tipo de medidas de refuerzo, ya que tienen algunos inconvenientes. Hay que tener cuidado con el uso intensivo de las motivaciones extrínsecas (complementos económicos o materiales extra) ya que pueden llegar a anular las motivaciones intrínsecas, lo que provoca un fenómeno denominado sobrejustificación. El estudio concluye que la mejor valorada de las cuatro es la estrategia alentadora, ya que es la más motivadora para realizar un proceso de mejora de deuda técnica.

Focus on desired behavior	Focus on undesired behavior	
<p>Encouraging</p> <p>Strategy:</p> <ul style="list-style-type: none"> • Mindset and attitudes from managers. • Attention on the importance of TD. • Raising awareness. <p>Tactics:</p> <ul style="list-style-type: none"> • Educational sessions about TD. • Dedicated time for TD remediations. 	<p>Forcing</p> <p>Strategy:</p> <ul style="list-style-type: none"> • Mandatory adapting to rules and requirements. • Give authority to conduct TD remediation tasks. <p>Tactics:</p> <ul style="list-style-type: none"> • Shared ownership. • Transition from and encouraging strategy. 	Without individual or team dis/incentives
<p>Rewarding</p> <p>Strategy:</p> <ul style="list-style-type: none"> • Reward an explicit behavior. • Part of the craftsmanship. <p>Tactics:</p> <ul style="list-style-type: none"> • Monetary or intangible compensations. • Risk if counterproductive results. 	<p>Penalizing</p> <p>Strategy:</p> <ul style="list-style-type: none"> • Penalizing and explicit behavior or when not reaching of a defined goal. <p>Tactics:</p> <ul style="list-style-type: none"> • Monetary fines - salary reductions. • Importance of establishing fair, adequate, and succinct rules. 	With individual or team dis/incentives

Ilustración 5 - Estrategias para la reducción de la deuda técnica. Elaboración a partir de [22]

En otro artículo se estudia el impacto de diferentes estrategias en Ingeniería del *Software* respecto a la gestión de deuda técnica en un contexto educativo[23]. En el estudio se muestran dos estrategias diferentes, una basada en la penalización y otra en la recompensa (*stick vs carrot*), aplicadas en un entorno de aprendizaje real.

Los resultados del estudio muestran que la estrategia de la recompensa funciona mejor que la estrategia basada en la penalización, para animar y ayudar a los estudiantes a producir un código de calidad con baja deuda técnica.

Relacionado con la medición de la calidad están presentes las métricas, útiles para ver la tendencia sobre el aumento o la disminución de la calidad del *software*. Es muy importante que más allá de reflexionar sobre un valor concreto de una métrica, seamos conscientes del umbral que tiene que pasar para convertirse de un valor normal a un valor no recomendable.

En el desarrollo para entornos de producción de la industria del *software*, existen diferentes estándares y normas. Comenzando por la motivación en el desarrollo de tareas, que aparece en normas de control y gestión de la calidad como la norma ISO 10006, que indica que para el desempeño eficaz del equipo se requiere que estén motivados y dispuestos al trabajo en equipo.

También destaca la norma ISO 12207, como estándar para los procesos de ciclo de vida del *software* de una organización [7], y la norma ISO 14598, utilizada para la evaluación de productos *software*, junto a la norma ISO 25000, para el desarrollo de productos *software* con criterios para la especificación y evaluación de calidad. Finalmente, se ha revisado la norma ISO 9126, como estándar sobre calidad y métricas asociadas tanto para la evaluación del producto *software* como para la definición de requerimientos de calidad.

4 Entorno tecnológico

En cuanto al entorno tecnológico, se han estudiado las diferentes herramientas para poder implementar un entorno automatizado para la aplicación de estrategias de gamificación que esté integrado en el marco de producción *software*. El estudio se ha centrado en DevOps CI/CD, revisando diferentes herramientas del ámbito de la gestión del *software*.

Se ha comenzado por el análisis de la metodología DevOps, que es el acrónimo de “*Development*” y “*Operations*” y que se puede definir como el conjunto de prácticas para el proceso de desarrollo *software* teniendo en cuenta la colaboración de los equipos de desarrollo junto con los de sistemas [24]. El objetivo es conseguir que estos desarrollos sean más seguros y ágiles, y se realicen con mayor calidad en el menor tiempo posible, consiguiendo una disminución del coste. Este concepto también se puede definir como una cultura que produce mejoras en la organización del trabajo a través de buenas prácticas relacionadas con el proceso de desarrollo, las pruebas y la integración y entrega continuas, centrándose en la colaboración y la distribución de tareas y responsabilidades, para que los equipos de desarrollo puedan centrar sus esfuerzos en la parte del desarrollo de la aplicación.

En cuanto a las integraciones CI/CD, suponen un método para distribuir las aplicaciones desarrolladas con la ayuda de mecanismos de automatización y control del ciclo de vida en las etapas de integración y prueba, distribución e implementación.

Se han estudiado herramientas relacionadas con estas prácticas. Todas ellas tienen las características comunes de facilitar el proceso de desarrollo, la visualización del progreso, la automatización de tareas, el control de la calidad y el seguimiento.

Para la aplicación y prueba del estudio anteriormente citado [23] se ha utilizado la herramienta SonarQube, como centro de implementación del entorno para obtener de forma automática medidas relacionadas con la deuda técnica. SonarQube es una plataforma que permite la evaluación de la calidad de código fuente a partir de un análisis estático sobre dicho código. Dispone de una interfaz gráfica para mostrar los diferentes indicadores para la mejora del código: código duplicado, muerto, estándares de codificación, bugs, complejidad ciclomática, test unitarios y de integración, cobertura de código o incluso la identificación de *code smells*. A través de los *webhooks* se notifican los resultados a los servicios externos cuando se completa el análisis de un proyecto.

Además, con SonarQube se puede definir un *scoring* teniendo en cuenta el *Total Quality* del proyecto con las métricas definidas. La fórmula tiene un peso configurable que permite obtener un valor total. Para ello se utiliza indicadores de calidad de diferentes tipos como la calidad del diseño, de la arquitectura, del código o de las pruebas. En las siguientes secciones se realizará una explicación más detallada de SonarQube y sus características.

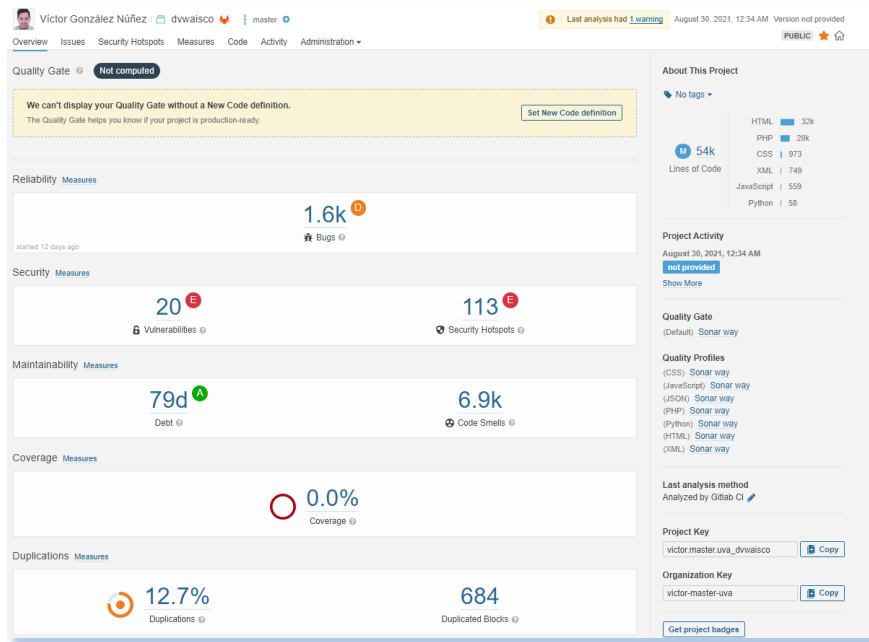


Ilustración 6 - Vista con los resultados de un análisis con SonarQube. Elaboración propia

En la propuesta de [23] se tiene en cuenta los atributos de calidad de mantenibilidad, fiabilidad y seguridad. La fórmula que utiliza es la suma de la deuda técnica junto con el código duplicado sumando los esfuerzos de corrección de errores potenciales y de vulnerabilidad. En caso de empate se aplican diferentes criterios para su resolución. Esta información se refleja en una *leaderboard* donde aparecen en la parte superior los equipos con menor *scoring*.

Existen diferentes herramientas de gestión de proceso *software* que utilizan gamificación. Destaca Trogón, un sistema de gestión de proyectos que consiste en una representación de un bosque mostrando las diferentes fases y tareas del proyecto en forma de diferentes elementos que componen el bosque. El sistema se basa en el modelo UAREI (Usuario-Acción-Regla-Entidades-*Interface*) que representa la mecánica del juego [25].

Cada una de las tareas tiene una dificultad de realización y un tiempo. Según se realizan, el participante recoge insignias y medallas, bajo la revisión del responsable del proyecto. Esto se refleja en la representación del árbol, que aumenta según se completan los hitos del desarrollo. Al final del proceso se muestra un ranking con las mejores puntuaciones. De manera similar funciona Taskville [26], pero utilizando una representación de ciudades para cada equipo de trabajo donde cada tarea que realizan se implementa en la construcción de un edificio.

En cuanto a la gestión de la configuración del software, se revisaron varios tipos de sistemas para el control de versiones como GitLab y GitHub, ambos basados en Git como sistema de control de forma distribuida. Respecto a la integración continua, GitLab permite integración continua gratuita mientras que GitHub permite esta integración con herramientas de terceros. Con la herramienta Gitlab CI se puede realizar una integración continua versionada, para que cada rama tenga una configuración propia y que por parte de los colaboradores se puedan añadir otras ramas. GitLab CI dispone de *runners*, gestión de *secrets* y *pipelines*. En la siguiente sección se ofrece una explicación más detallada sobre GitLab CI.

En cuanto a otras herramientas de integración continua de los procesos de desarrollo, destacan Jenkins y Travis CI. También existen otras como Bamboo o Nevercode. Todas ellas tienen arquitectura extensible, permiten una integración IDE, con plataformas en la nube y son de fácil uso. Además, destacan otras herramientas como JIRA, que es la gestión de software de Atlassian, que también permite la integración de herramientas de CI/CD. También hay que destacar la herramienta Quboo⁵, que tiene la capacidad de recibir los datos de las diferentes integraciones y realizar su procesamiento. Con el *plugin* de SonarQube se puede conectar al repositorio donde está el código y calcular la puntuación según las métricas definidas teniendo en cuenta el umbral de deuda técnica.

4.1 Gestión de proyectos software

La gestión de proyectos ayuda a las organizaciones a tener una planificación, cumplir con los objetivos planteados: presupuesto, plazos de entrega, alcance, etc., además de mejorar la eficiencia, el trabajo en equipo y la identificación de dificultades y riesgos. Existen diferentes metodologías de gestión de proyectos como es la metodología en cascada, Agile, Kanban, SCRUM, LEAN, etc. Para la elección de la metodología que se va a aplicar en un proyecto, hay que valorar el tipo de desarrollo que va a realizar, los procesos que tiene la organización, el tipo de equipo con el que se cuenta (habilidades, capacidad). Una buena gestión de proyectos proporciona un valor extra a la organización, que garantiza una entrega bien hecha para la satisfacción del cliente.

Para lograr un buen proyecto, hay que tener en cuenta que hay que realizar una planificación realista según los recursos con los que se cuenta y las estimaciones temporales. Siempre este proceso mejora con la experiencia ya que el *background* de realización en proyectos pasados ayuda a estimar plazos y recursos: se aprende de los éxitos y de los fracasos del pasado.

Es recomendable utilizar herramientas de gestión de proyectos para tener una buena coordinación comunicación entre los miembros del equipo, poder definir y asignar tareas de forma dinámica, además de centralizar el conocimiento. Algunas de estas herramientas son JIRA⁶, Zoho Projects⁷ o RedMine⁸.

4.1.1 Trabajo en equipo

En el trabajo de los equipos de proyectos *software* existen diferentes factores que influyen en su éxito. Con el desarrollo de las telecomunicaciones y la extensión del teletrabajo los equipos de trabajos no comparten lugar de físico de desarrollo de sus tareas. Además, hay que tener en cuenta que disponer de las personas idóneas con las habilidades adecuadas para las necesidades de un proyecto no significa que su participación en el proyecto sea exitosa.

Por ello, la gestión de los diferentes miembros de los equipos es importante para conseguir un rendimiento óptimo. Es sustancial crear un entorno de equipo colaborativo, ya que la productividad aumenta cuando existe una buena comunicación interna, ya que mejora la eficiencia y la velocidad de consecución de los objetivos. Por lo tanto, el equipo de proyecto se puede definir como un grupo de personas con un objetivo en común y que comparte una identidad común y tiene diferentes normas para lograr dicho objetivo.

⁵ Quboo docs. Quboo Gamification Platform for IT organizations <https://docs.quboo.io/>

⁶ Jira Software. Atlassian. <https://www.atlassian.com/es/software/jira>

⁷ Zoho Projects. Software de gestión de proyectos basado en la nube. Zoho. <https://www.zoho.com/es-xl/projects/>

⁸ Redmine. Aplicación web de gestión de proyectos flexibles. <https://www.redmine.org/>

También hay que tener en cuenta las habilidades creativas de los miembros implicados en los equipos de desarrollo que, además de tener habilidades y conocimientos técnicos, es relevante que tengan elementos relacionados con la inteligencia emocional: calidad humana, motivación o perseverancia. [27]

La creatividad se puede tener como un recurso para aumentar la innovación. Existen trabajos en los que se plantea la relevancia de la integración en el desarrollo *software* de dos componentes : el cognitivo y el afectivo [28]. Es decir, la gestión de la creatividad en los equipos de trabajo favorece la alineación de los miembros con los objetivos organizaciones, además de aumentar la productividad y su sentido de pertenencia.

Para conseguir el éxito en un desarrollo, hay que disponer de una metodología de trabajo con unos procedimientos elaborados y adaptados al proyecto que se va a acometer. Además, tener una relación previa a la realización del proyecto contribuye a una mejor comunicación. Respecto al jefe de equipo, un liderazgo claro y fuerte puede ayudar a obtener el mejor rendimiento de cada uno de los miembros, conociendo el estilo de trabajo de estos miembros y sus habilidades para poder adaptarlas a los objetivos según el alcance del proyecto. El jefe debe de tener una alta motivación y una visión que inspire a los miembros del equipo. La motivación es algo que entra también en juego y, como se ha detallado en secciones anteriores, influye según se apliquen diferentes estrategias.

Por lo tanto, una gestión de proyectos ayuda a la mejora de la calidad de estos. Sin un control de plazos, procesos y organización de tareas, el resultado puede ser un producto de mala calidad.

4.1.2 Calidad en los proyectos *software* y entorno tecnológico

El control de la calidad de los proyectos *software* es fundamental. En todos los desarrollos hay que tener en cuenta que el producto que llega al cliente, más allá de cumplir los requisitos definidos, debe de tener un nivel calidad para no crear un problema con el tiempo, ya sea por uso o por mantenimiento. Esto se puede traducir en pérdida de tiempo: retrasos en el desarrollo y, por lo tanto, en la entrega, costes económicos, baja productividad del equipo de trabajo, etc. Hay que dedicar tiempo al cuidado de la calidad en los proyectos, aplicando métodos de trabajo sistemáticos como control de calidad con normas ISO/IEC 15504 SPICE o el modelo de madurez y capacidad integrado CMMI.

Cada día más empresas y administraciones públicas exigen cumplir ciertas normas, incluso exigiendo certificaciones oficiales para el control de calidad con el objetivo de obtener un valor añadido o evitar problemas futuros. Además, la calidad en el desarrollo se está empezando a considerar un elemento de innovación en los procesos de creación como mejora, ofreciendo un producto de más calidad para los clientes, lo que se traduce en una mejora de la competitividad. Esta calidad tiene condicionantes como son las restricciones, por ejemplo, de presupuesto o de plazos de fabricación.

Esta es una cuestión muy relevante para la industria del *software*. Peter Neumann viene realizando una recopilación de grandes fallos de *software* desde el año 1997 [29]. Según [30] los principales problemas en el *software* se deben a no comprender los requisitos (50%), no comprender el diseño o un traslado incorrecto de los requisitos (30%) y un error de programación o diseño mal entendido. Además de existir una deficiente formación en el ámbito del control de la calidad.

Existen medidas para paliar estos problemas. Según [31], se puede conseguir una mejora de la calidad a partir de la creación de un sistema de evaluación y medición de tiempo, esfuerzo, errores para la corrección, incluyendo la documentación de todo el proceso y la difusión del coste de la corrección. También eliminando los errores de requerimientos y diseño, y recopilando información de usuarios y directores a través de entrevistas. La mejora de la calidad se entiende como un proceso continuo.

La calidad se puede clasificar a nivel organizacional o a nivel de proyecto. En el nivel organizacional se tienen en cuenta todas las fases de creación y las acciones realizadas para aplicar ese control de calidad en todos los proyectos de la organización, teniendo en cuenta la estrategia que defina una política de calidad, que incluya una planificación de actividades y medidas (evaluaciones, revisiones, etc.), con una asignación de recursos (personal, desarrolladores, equipos informáticos, etc.). En cuanto al nivel de proyecto, se refiere a un plano más concreto. En mi opinión, ambas deben de ir de la mano para conseguir asegurar la calidad.

En cuanto a los equipos de trabajo, en el momento de la incorporación de un miembro al equipo de trabajo, se encuentra ante un reto de familiarización con una mezcla de aspectos tecnológicos y organizacionales, llegando a tardar semanas en ser productivo para la organización [32]. Sobre este asunto se tiene que destacar la ley de Brooks, que afirma que cuando una persona se incorpora a un proyecto que ya acarrea un atraso, lo retrasará más [33].

En el estudio [34] se realizaron un conjunto de entrevistas a desarrolladores y jefes de equipo de dieciséis empresas de desarrollo. Sus principales conclusiones fueron que existen determinadas barreras para los nuevos miembros de un equipo en un proyecto. Para los desarrolladores, destacan las dificultades con la documentación, tanto del código como de los requerimientos, y el entendimiento de la solución como visión global del sistema y de los problemas de negocio que se intentan resolver. En el caso de los jefes de equipo, la dificultad se centra también en el entendimiento de la solución y en la metodología del trabajo como proceso de familiarización del proceso de creación, las rutinas de trabajo, etc.

Las principales acciones para mitigar estas dificultades fueron asignar a un miembro “experto” para que haga una tarea de orientación sobre la nueva incorporación, centrándose en aspectos técnicos del proyecto y de integración con los miembros del equipo. También un seguimiento continuo de las tareas, así como un buen proceso de documentación, favorece en el proceso de disminución de dificultades para la incorporación de nuevos miembros.

Hay diferentes modelos de calidad para asegurar los niveles de esta durante el ciclo de vida de los productos *software*. Existen diferentes guías, normas o metodologías para aplicar en el proceso de construcción de *software* y evaluar según diferentes criterios. Según [35], los modelos de calidad se estructuran desde los factores de calidad que tienen que cumplir, que se componen de diferentes criterios de los que forman parte diferentes métricas. Además, el autor diferencia distintos niveles según el enfoque de evaluación según calidad a nivel de proceso, de producto y en uso. Según McCall, se deben de tener en cuenta los factores de calidad del *software*, que se pueden clasificar según tres perspectivas: operativa, mantenimiento y evolutiva[36].

En el sector de los profesionales de la ingeniería del *software* se produce un debate sobre si el estado de los proyectos *software* está en crisis o no [37]. En este debate se presentan estudios donde se considera que el más relevante sobre la materia es el *Chaos Report*, un informe muy popular para sector de las Tecnologías de la Información [38]. En este informe se muestran los resultados del estudio sobre el fracaso de los proyectos *software*.

4. Entorno tecnológico

Su objetivo es encontrar formas eficientes de desarrollo de sistemas para lograr proyectos de mayor calidad y éxito. Este informe lo realiza The Standish Group⁹⁹, una organización independiente que lo lleva realizando desde el año 1994. El último informe es el publicado en 2015. En el mismo se muestran los resultados de la medición de los proyectos según tres variables:

- Acabado en plazos (onTime)
- Acabado en presupuesto (onBudget)
- Resultados satisfactorios

En la siguiente ilustración se muestra la resolución de todos los proyectos de *software* por industria, entre los años 2011 y 2015, analizados por la organización.

	Successfull	Challenged	Failed
Baking	30%	55%	15%
Financial	29%	56%	15%
Government	21%	55%	24%
Healthcare	29%	53%	18%
Manufacturing	28%	53%	19%
Retail	35%	49%	16%
Services	29%	52%	19%
Telecom	24%	53%	23%
Other	29%	48%	25%

Ilustración 7 - CHAOS report. Resolución en los proyectos software desde 2011 a 2015. Elaborado a partir de [38]

En mi opinión, los avances en desarrollo de software junto con el aumento en la difusión y conocimiento de las metodologías, provocará que haya unos mejores resultados en los siguientes informes, aumentando los casos de éxito y reduciendo los fallidos, pero también disminuyendo los discutidos. Con ello, tendremos software más confiable, cada vez se evitarán más errores, existirán menos sobrecostes, y todo esto se traducirá en una gran ventaja competitiva por el amplio ahorro para las organizaciones.

4.2 Sistemas de Control de Versiones

La gestión y configuración de versiones es una parte fundamental en la gestión de proyectos en la industria de la producción de *software*. Un control de versiones es un sistema que facilita el registro de los cambios realizados en los archivos que forman un proyecto, ya sea código fuente, documentos u otro tipo de ficheros. Esto permite que los desarrolladores tengan diferentes versiones del código de un proyecto y que puedan realizar un seguimiento de los cambios que se han realizado, facilitando la vuelta a una versión anterior, la comparación de cambios y el

⁹⁹ The Standish Group. <https://www.standishgroup.com/>

propio control de los cambios realizados, además de observar quién, cómo y cuándo se han realizado estas modificaciones, para una mejor gestión de los miembros del proyecto.

En la práctica, algunos desarrolladores simplemente crean copias de sus trabajos en sus equipos locales, utilizando diferentes carpetas o simplemente cambiando el nombre de las versiones o directamente sobrescribiéndolas. Con los sistemas de control de versiones, se permite un control más robusto sobre las tareas que realizan los desarrolladores, ofreciendo un historial de los cambios, además de un sistema de copia de seguridad externa del código fuente. Además, es muy útil para el trabajo con diferentes entornos: preproducción, producción, etc.

Finalmente, los sistemas de control de versiones disponen de mecanismos de resolución de conflictos, en el caso de que se realicen cambios en el mismo archivo del código fuente por parte de dos o más miembros del equipo de desarrollo.

4.2.1 Tipos

Existen dos tipos de sistemas de control de versiones según el tipo de acceso y la localización de los repositorios: los sistemas de control de versiones centralizados y los distribuidos. Los primeros, *CVCS - Centralized Version Control Systems*, tienen un repositorio único que centraliza la información alojada en un servidor central. La gran ventaja es que facilita el trabajo colaborativo entre los miembros de un equipo de desarrollo. Por ejemplo, cuando uno de los miembros realiza una actualización, esta se hace directamente en el repositorio.

La desventaja de este sistema es que no están en local, por lo que es requisito necesario tener conexión con la red donde se aloja el servidor central. Además, si se produce un problema en el servidor central, puede dar lugar a la pérdida de toda la información. Algunos de sistemas de control centralizados más conocidos son CVS¹⁰ o Subversion¹¹.

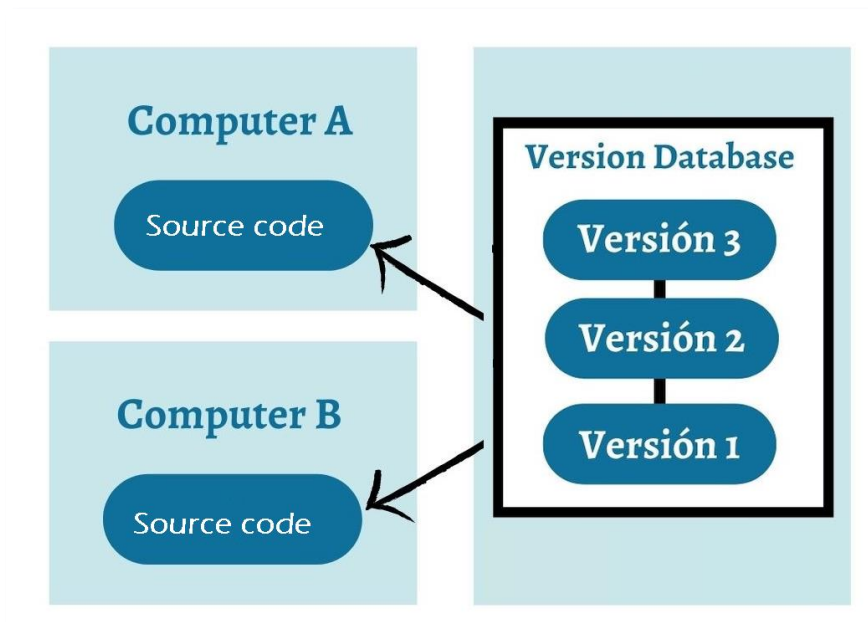


Ilustración 8 - Sistema de control de versiones centralizado. Elaboración propia

¹⁰ CVS - Concurrent Versions System. NonGNU. www.nongnu.org/cvs/

¹¹ Subversion is an open source version control system. Apache Subversion. subversion.apache.org/

En cuanto a los sistemas de control de versiones distribuidos (*DVCS - Distributed Version Control Systems*), a diferencia de los centralizados no dependen de un único servidor donde se aloje el código del proyecto. Los desarrolladores disponen de una copia local del repositorio sobre la que actualizan el código, para luego enviar esos cambios a un repositorio principal. Las principales ventajas son que, si el servidor central tiene problemas, los datos se pueden recuperar desde los repositorios locales de cada desarrollador. Además, las operaciones se producen de forma más rápida, al no necesitar acceder a la red del servidor central. Los usuarios tienen una copia completa de todos los datos. Algunos de los sistemas de control distribuidos más reconocidos son Git, Monotone o Mercurial.

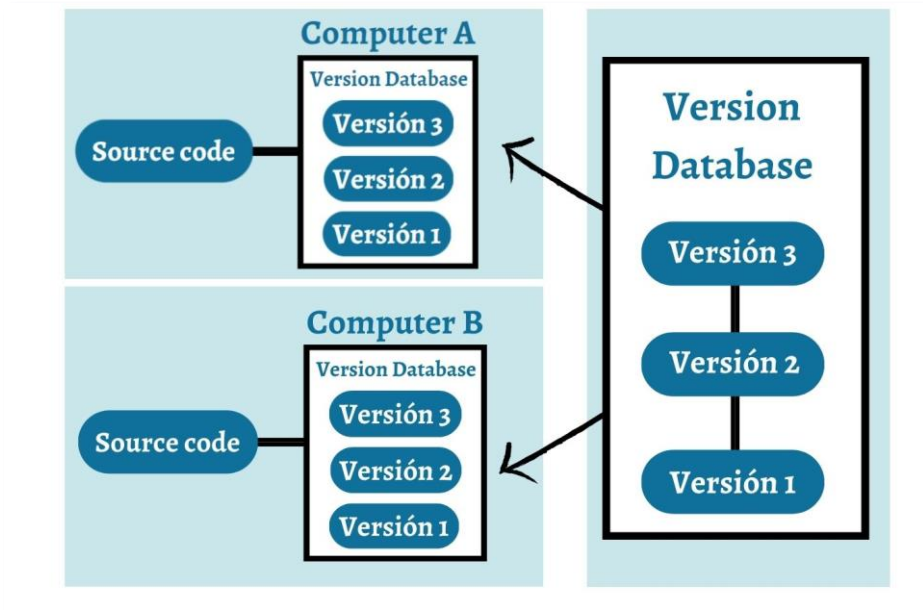


Ilustración 9 - Sistema de control de versiones distribuido. Elaboración propia.

Existen diferentes enfoques sobre la aplicación de técnicas de versionado. La primera sería el desarrollo lineal, donde todos los miembros del equipo suben en una única línea del control de versiones. Con esta estrategia se pueden producir errores por lo que la versión no sería muy estable. Alternativamente, se utiliza otro enfoque basado en un desarrollo en paralelo con varias ramas. Se crea una rama por cada nueva tarea o funcionalidad. Cuando se termina y sea una versión estable, se puede integrar en la rama principal.

4.2.2 La importancia del control de versiones

Disponer de un control de versiones aporta beneficios para la organización, como la creación de una estrategia de versionado que mejore el control sobre el proyecto o el papel de cada miembro del equipo, fomentando la colaboración y la aplicación de buenas prácticas para la documentación de los procedimientos, incluso el mantenimiento de los entornos. Esto se traduce en una mejora del rendimiento del equipo, evitando el descontrol del código y los errores repetitivos e innecesarios, que provocan retrasos en la entrega y en la propia calidad del código.

A través de la implementación de reglas de proceso de desarrollo, el responsable del equipo puede asignar roles y niveles de responsabilidad para la aplicación de las modificaciones del código, fundamental en el proceso de gestión del lanzamiento de una versión, para poner a disposición de los usuarios una nueva funcionalidad. Esto se puede valorar con indicadores de éxito en la gestión de versiones, como puede ser el tiempo de desarrollo, el impacto sobre el presupuesto del proyecto o la satisfacción de los usuarios/clientes.

4.3 Git

Git es el Sistema de Control de Versiones multiplataforma de código abierto, desarrollado por Linus Torvalds, el creador del *kernel* de Linux. Es el sistema más utilizado en el mundo para trabajar con proyectos de todo tipo de tamaños con rapidez, seguridad y flexibilidad. Git tiene una gran comunidad de usuarios, así como documentación y materiales que facilitan su rápida adaptación a los equipos de trabajo. Su funcionamiento es compatible con los sistemas operativos más utilizados y los entornos de desarrollo más populares debido a que una gran parte de las herramientas de desarrollo de *software* están integradas con Git. Es la base de plataformas de alojamiento de proyectos *software* como GitLab¹² o GitHub¹³.

4.3.1 Características

Es un sistema de control de versiones distribuido, en el que los usuarios disponen de repositorios locales donde realizar los cambios en el código para luego sincronizar esta copia local con el repositorio que se encuentra en el servidor. Para ello se utiliza un sistema de ramas, en el que las nuevas funcionalidades se aplican desde las ramas locales de los usuarios, para posteriormente combinarlas con la rama principal. El sistema crea una instantánea con los archivos permitiendo controlar las diferencias entre versiones.

Esta arquitectura distribuida se traduce en un mejor rendimiento. Al desarrollar los cambios en local, los usuarios pueden seguir trabajando con sus versiones en sus repositorios locales hasta tener todas las correcciones realizadas, para finalmente actualizar esos cambios en la rama principal. También facilita la creación de un historial de cambios, a partir de ramas de cada versión y un sistema de etiquetado, además del mantenimiento de la integridad del código fuente.

Git dispone utiliza el algoritmo “SHA1”, que protege los archivos, las versiones, los *tags*, etc., con un algoritmo de *hash* criptográficamente seguro que certifica la trazabilidad del proceso de versionado [39].

4.4 Gitlab

Gitlab es un servicio web de control de versiones de código abierto basado en Git. Dispone de un gestor de repositorios para realizar un seguimiento de las diferentes versiones del código de un proyecto. Permite el desarrollo de *software* de manera colaborativa, con un sistema de miembros y organizaciones que gestiona los roles y el acceso a las diferentes ramas de un proyecto.

¹² GitLab <https://gitlab.com/>

¹³ GitHub <https://github.com/>

4. Entorno tecnológico

Dispone de diferentes versiones, desde el plan gratuito hasta planes de pago que ofrecen funciones de flujo de trabajo más avanzadas y una mayor capacidad. Es uno de los sistemas de control de versiones más utilizado del mundo, incluso en proyectos de grandes empresas tecnológicas como Intel, RedHat, la NASA o Sony [40].

Permite una gestión de los proyectos, tanto privados como abiertos, permitiendo su compartición con otros usuarios. También dispone de la figura de proyectos internos, donde los usuarios que pertenecen a un equipo de trabajo pueden acceder a los proyectos de su equipo, realizar copias con *forks* o trabajar con las *pull/merge requests* para aceptar o no los cambios sugeridos para determinadas ramas. Además, dispone de un sistema de integración continua y de pruebas unitarias para evitar problemas de integración. El flujo de trabajo permite controlar todo el proceso de desarrollo y la gestión del control de versiones, fomentando la colaboración entre miembros del equipo, y mejorando la comunicación y el aprendizaje.

Su versión en la nube, Gitlab.com¹⁴, dispone de una gran cantidad de medidas de seguridad. También ofrece otros servicios como las wikis, para crear espacios de documentación de los archivos o un sistema de gráficos para revisar las interacciones de los miembros del equipo de desarrollo, además de ofrecer la opción de autenticación contra LDAP y el acceso por SSH. Permite la integración en los ciclos de vida de los desarrollos y pruebas de seguridad de aplicaciones tanto estáticas (SATS) como dinámicas (DAST).

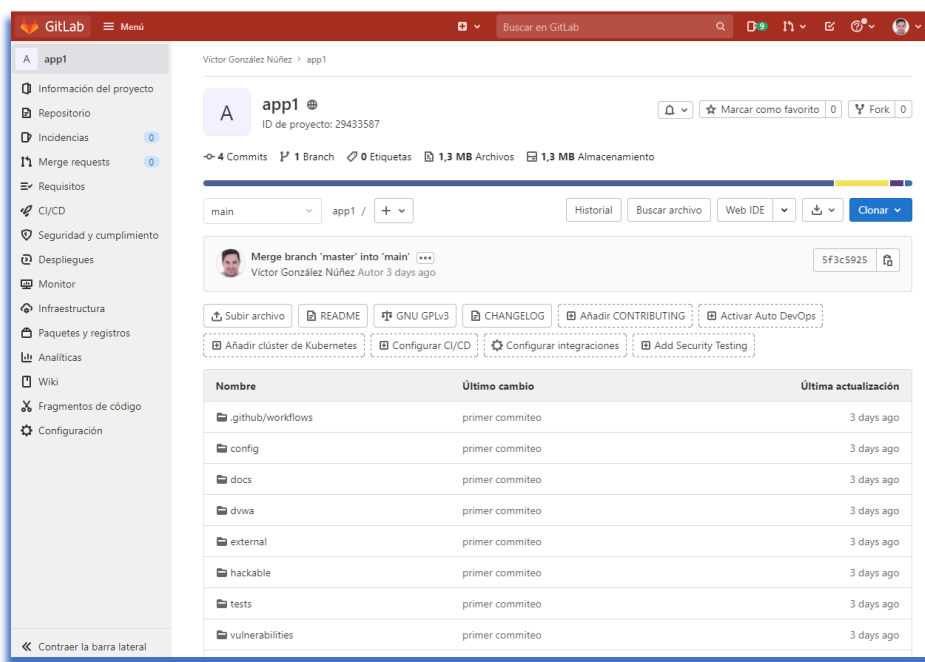


Ilustración 10 - Interfaz de GitLab. Vista de la gestión de un proyecto. Elaboración propia.

Su administración se realiza con un acceso web y dispone de un área donde aparecen los proyectos. Hay que destacar el concepto de *namespace* como espacio de nombres como agrupación de proyectos de un usuario. También ofrece grupos de usuarios, que permite asignar un nivel de permisos sobre el proyecto del que forman parte. Los diferentes permisos que se asignan permiten desde el acceso como invitados hasta un acceso con el control total del proyecto.

¹⁴ GitLab is the DevOps Platform. GitLab. gitlab.com

Además de ofrecer la posibilidad de compartir proyectos, también dispone de un sistema de comentarios de usuarios y también de *requests* que fomentan la mejora de proyectos de manera colaborativa.

En el ámbito empresarial, su funcionamiento en la práctica consiste en conceder permiso a determinados miembros implicados en el proyecto para que puedan hacer *push* directo al repositorio del proyecto. De esta manera, los usuarios podrán crear ramificaciones del código con las nuevas funcionalidades o realizar solicitudes de combinación (*merge requests*) que podrán ser supervisadas por el jefe de equipo.

4.4.1 GitLab vs GitHub

GitHub está considerado la otra gran alternativa como proyecto de desarrollo de proyectos. Una de las principales ventajas de GitLab frente a GitHub es que es totalmente *opensource*, se puede instalar en un servidor de forma gratuita y no tiene límite de repositorios privados ni de miembros colaboradores. Además, ofrece mayor granularidad en los permisos de acceso a los usuarios, su facilidad de uso e instalación. GitLab ofrece de manera gratuita CI/CD, sin necesidad de utilizar un tercero. Además, destaca por sus mejoras en la gestión del código, la seguridad y las tareas de planificación del proyecto y la documentación de su plataforma. También tienen determinadas diferencias en el uso de la terminología, como la denominación de una organización y un repositorio en GitHub que “equivale” a los grupos y proyectos en GitLab.

4.4.2 GitLab CI/CD

La integración y entrega continua supone una mejora para los equipos de trabajos de desarrollo *software*. En el proceso de integración y automatización de los procesos de desarrollo de código y los equipos de sistemas se puede hacer uso de herramientas para que los desarrolladores puedan colaborar e integrar sus desarrollos de manera sencilla. Existen herramientas que ayudan a configurar los proyectos *software* para hacer esta tarea automáticamente, con un uso fácil para la compilación del código, la realización de pruebas o la publicación. Las herramientas más relevantes son Atlassian Bamboo, Bitbucket Pipelines, Jenkins y Gitlab CI/CD. Además, también destacan soluciones como AWS CodePipeline¹⁵, para trabajar con la infraestructura de Amazon Web Services, o Azure Pipelines¹⁶, como plataforma de pruebas e integración continua de Microsoft.

GitLab CI/CD se ha considerado el “DevOps completo” por permitir una gestión integrada de CI/CD, con una entrega de *software* de manera rápida y eficiente, fácil de adoptar, con más de treinta millones de usuarios [41]. Proporciona la visualización del ciclo de vida DevOps dentro del mismo sistema. Además, la gestión de prioridades con *issues*, simplifica la revisión del código y automatizando las pruebas de calidad del código permitiendo un escaneo para encontrar vulnerabilidades. En cuando a la integración con servicios de CI, permite hacerlo con terceros como Jenkins, pero no es obligatorio para realizar la integración continua. Dispone de cobertura para enganches (*hooks*), para la conexión entre repositorios y GitLab. Respecto a la entrega continua (CD), ayuda a la automatización del lanzamiento y la entrega de aplicaciones de manera automatizada teniendo un control total de todo el proceso. Esto se detalla en la siguiente sección.

¹⁵ AWS CodePipeline <https://aws.amazon.com/es/codepipeline/>

¹⁶ Azure Pipelines <https://azure.microsoft.com/es-es/services/devops/pipelines/>

4.4.3 Pipelines y GitLab Runner

En cuanto a la creación de los flujos de trabajo con los *pipelines* de CI/CD, se pueden hacer desde la misma interfaz de GitLab. Para ello se utilizan los GitLab Runners, que son unas máquinas virtuales aisladas cuyo funcionamiento sirve para ir realizando los pasos de cada fase del *pipeline* a través de la API de Gitlab CI. Al finalizar el proceso, envía los resultados al cuadro de mandos de GitLab. En la creación del *pipeline* se van añadiendo los pasos por los que va a pasar el código. Por ejemplo, se puede crear una fase de *built* para instalar dependencias, a continuación, una fase de pruebas, y añadir una fase final con descripción del proceso de entrega.

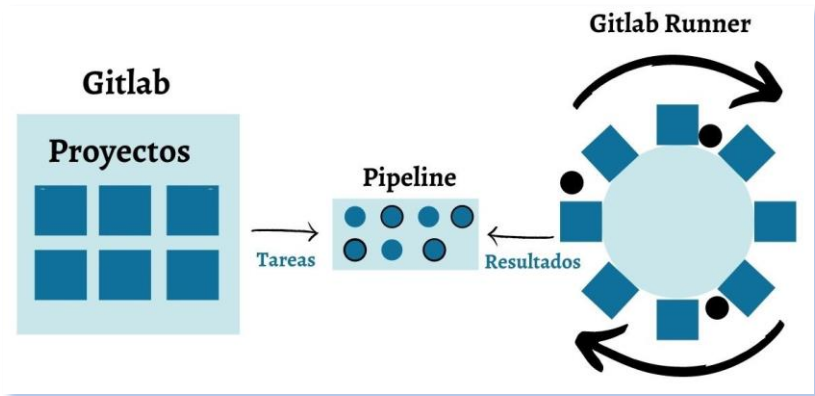


Ilustración 11- Ejemplo de workflow con GitLab. Pipeline y GitLabRunner. Elaboración a partir de [42]

Esta definición de pasos se añade en el fichero yaml del proyecto. Este formato YAML (*Ain't Markup Language*) es muy *entendible* para los usuarios y permite añadir las librerías predefinidas que queremos que utilice el *runner*. La definición se realiza en el fichero “.gitlab-ci.yml”, dónde se definen las tareas, que Gitlab transforma en *pipelines* para que, a continuación, el *runner* las ejecute. Gitlab ofrece un validador CI Lint tool para comprobar la sintaxis del fichero YAML.

```
stages:          # List of stages for jobs, and their order of execution
  - build
  - test
  - deploy

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."

unit-test-job:   # This job runs in the test stage.
  stage: test
  script:
    - echo "Running unit tests... This will take about 20 seconds."
    - sleep 20
    - echo "Code coverage is 90%"

lint-test-job:   # This job also runs in the test stage.
  stage: test
  script:
    - echo "Linting code... This will take about 10 seconds."
    - sleep 10
    - echo "No lint issues found."

deploy-job:      # This job runs in the deploy stage.
  stage: deploy
  script:
    - echo "Deploying application..."
    - echo "Application successfully deployed."
```

Ilustración 12 – GitLab CI example. Elaboración a partir de template GitLab.com

4.5 SonarQube

SonarQube es una herramienta para la verificación de la calidad del código. Dispone de un cuadro de mando con diferentes opciones que muestran indicadores de calidad e información relevante sobre el código. Se utiliza para revisar la calidad del código que realizan los desarrolladores en la industria del *software*. A partir de un análisis estático (proceso de evaluación sin necesidad de ejecución) del código, ofrece una retroalimentación al usuario con el estado del desarrollo, mostrando los puntos débiles del código y las recomendaciones de cómo mejorarlo.

Dispone de un conjunto de reglas que incluyen información sobre ejemplos de código que muestran la solución recomendada, lo que favorece el aprendizaje de los desarrolladores. Estas recomendaciones pueden pasar a formar parte de las buenas prácticas de desarrollo en las organizaciones como guía de resolución de problemas. Con ello se garantiza que el desarrollo se está realizando de una manera correcta y se están aplicando buenas prácticas, disminuyendo la probabilidad de errores. Este control estricto ayuda a la reducción de la deuda técnica.

La aplicación tiene licencia LGPL y soporta 27 lenguajes de programación, entre los que destaca Python, C/C++, JAVA o Javascript. La interfaz de la aplicación muestra una vista general del proyecto con los valores obtenidos respecto a las métricas. En cada análisis, esta información se va actualizando automáticamente. Además, desde la vista general de proyectos, se puede ver de forma resumida el estado de todos los proyectos de un vistazo.

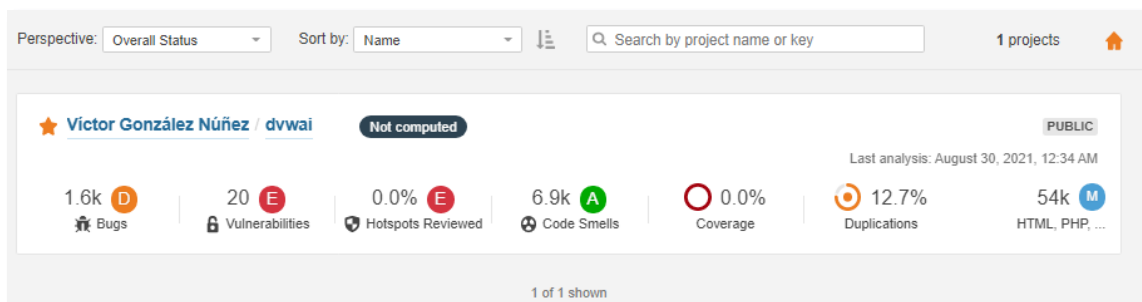


Ilustración 13- Interfaz de SonarQube. Resumen de análisis realizado. Elaboración propia.

Respecto a la seguridad, SonarQube ofrece Static Application Security Testing (SAST), un análisis con pruebas de seguridad de aplicaciones estáticas que permite la identificación de fuentes de vulnerabilidades y puntos calientes (*hotspots*) o de acceso inseguros que pueden ser un riesgo para la aplicación, mostrando una descripción del problema para su posible solución. Además, ofrece cobertura con la información del Top 10 de OWASP¹⁷, que es un indicador de los diez riesgos de seguridad para las aplicaciones web y que elabora la Open Web Application Security Project (OWASP), que es una organización sin ánimo de lucro que publica este estudio cada tres años.

Además, permite la integración con herramientas de análisis estático que facilitan comprobar las reglas de calidad que se determinen. Estas herramientas son Checkstyle, que analiza según reglas de codificación definidas; PMD para malas prácticas y problemas futuros; y FindBugs, para la búsqueda de errores en el código. Con esta herramienta, el desarrollador puede realizar un análisis estático del código donde SonarQube muestra la información sobre determinadas métricas de calidad como el código duplicado, errores potenciales, la complejidad ciclomática, los comentarios o los test unitarios y de integración. Permite la integración con herramientas de integración continuas como Jenkins o Bamboo Atlassian.

¹⁷ OWASP Top Ten. OWASP. owasp.org/www-project-top-ten



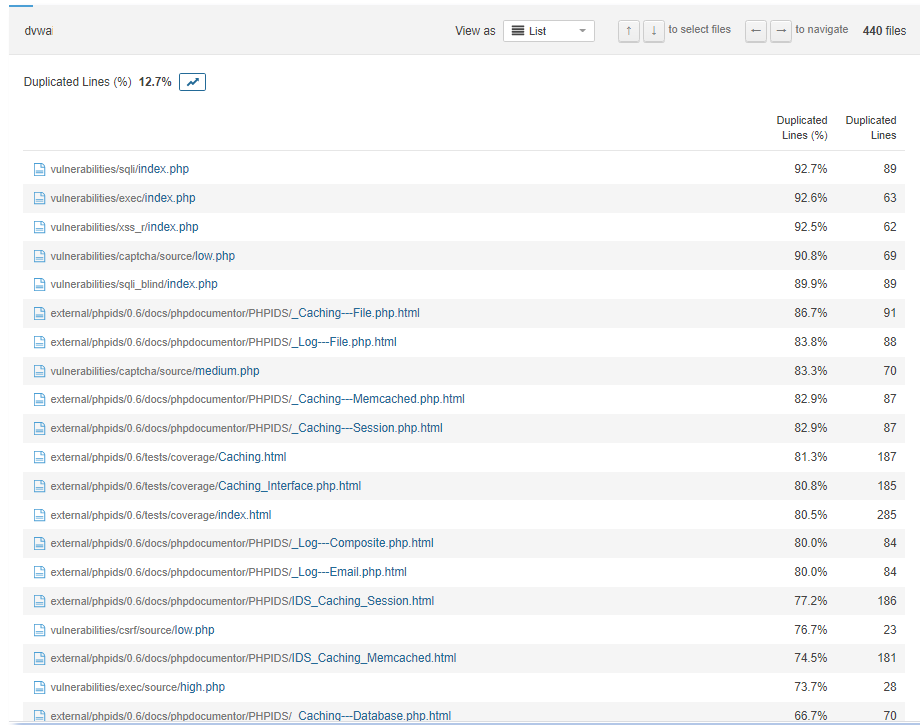
Ilustración 14- Interfaz de SonarQube. Vista de los resultados de análisis de vulnerabilidades

Las métricas de calidad más relevantes son las siguientes:

- **Bugs y vulnerabilidades:** Muestra puntos débiles del código, vulnerabilidades que pueden ser focos de problemas de seguridad.
- **Quality Gate:** Es un conjunto de criterios de calidad que debe de cumplir el código para que sea apto para pasar a la siguiente etapa de publicación. Con ello se evita la entrega de código con errores, mejorando la reputación organizacional. Esta *quality gate* muestra dos estados: *passed* o *failed*.
- **Code smells:** Métrica para valorar la calidad de elaboración del código, que determina si en un futuro puede provocar problemas de mantenibilidad. Un código limpio favorece su entendimiento por miembros del equipo que no han estado implicados en su desarrollo, disminuyendo el tiempo dedicado a su comprensión, lo que se traduce en una mejorar del rendimiento de los equipos de trabajo. Esta métrica está relacionada con la deuda técnica.
- **Complejidad ciclomática:** Mide la complejidad lógica de nuestro código. Calcula el número de caminos independientes que aparecen, desde la entrada hasta la salida. Una alta complejidad ciclomática dificulta el mantenimiento del código.
- **Complejidad cognitiva:** Valora la calidad de la estructura del código y su entendimiento. El código complejo o confuso puede incluir más errores, incluso de ser más complicado para entender para otros desarrollares que no han estado implicados en el trabajo.
- **Duplicidad:** Una parte del código fuente aparece duplicado varias veces. Una buena práctica es evitar esta duplicidad para que, en caso de tener que mantener código y realizar alguna modificación, se pueda evitar replicar los cambios en cada parte repetida. Esto produciría una pérdida de tiempo en su mantenimiento.
- **Código muerto:** Código inservible, que aparece en el desarrollo pero que no se utiliza para nada.
- **Comentarios:** Esta métrica valora el nivel de comentarios y su calidad. Los comentarios ayudan a entender el código, sobre todo para desarrolladores que no han estado implicados desde el inicio en la creación de ese código. Un buen porcentaje de comentarios no significa que el código sea mejor, ya que estos deben de ser adecuados y suficientemente explicativos.
- **Cobertura de pruebas:** Este indicador muestra si la cobertura de pruebas cumple con los niveles de calidad deseados y si hay que aumentar los test.

4. Entorno tecnológico

- **Coverage:** Indicador de cobertura de código que muestra el porcentaje de código validado por los test.
- **Test unitarios y test de integración:** Son unas pruebas que demuestran si el código tiene un funcionamiento correcto antes de aplicar los cambios definitivos.



	Duplicated Lines (%)	Duplicated Lines
vulnerabilities/sql/index.php	92.7%	89
vulnerabilities/exec/index.php	92.6%	63
vulnerabilities/xss_r/index.php	92.5%	62
vulnerabilities/captcha/source/low.php	90.8%	69
vulnerabilities/sql_blind/index.php	89.9%	89
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Caching--File.php.html	86.7%	91
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Log--File.php.html	83.8%	88
vulnerabilities/captcha/source/medium.php	83.3%	70
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Caching--Memcached.php.html	82.9%	87
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Caching--Session.php.html	82.9%	87
external/phpids/0.6/tests/coverage/Caching.html	81.3%	187
external/phpids/0.6/tests/coverage/Caching_Interface.php.html	80.8%	185
external/phpids/0.6/tests/coverage/index.html	80.5%	285
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Log--Composite.php.html	80.0%	84
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Log--Email.php.html	80.0%	84
external/phpids/0.6/docs/phpdocumentor/PHPIDS/IDS_Caching_Session.html	77.2%	186
vulnerabilities/csrf/source/low.php	76.7%	23
external/phpids/0.6/docs/phpdocumentor/PHPIDS/IDS_Caching_Memcached.html	74.5%	181
vulnerabilities/exec/source/high.php	73.7%	28
external/phpids/0.6/docs/phpdocumentor/PHPIDS/_Caching--Database.php.html	66.7%	70

Ilustración 15 - Interfaz de SonarQube. Vista de líneas duplicadas: Número y porcentaje

Además, se destaca el valor del SQALE rating. Este valor, cuyas siglas son *Software Quality Assessment based on Lifecycle Expectations*, está relacionado con el concepto de deuda técnica, que ayuda a la mejora de calidad basada en expectativas. Este *rating* muestra una escala de entre la A y la E según la ratio de la métrica. Cada letra tiene un color, que va desde la gama del verde de la A hasta el rojo de la E. Por ejemplo, en cuanto la métrica de la deuda técnica, cuanto más cerca de la A esté el proyecto, menor porcentaje de deuda técnica y mayor calidad tiene el proyecto.

Estas métricas son las que posteriormente se van a utilizar en la implementación para la elaboración de la puntuación de “*scoring*” y crear el *ranking* con los mejores desarrollos. Para los equipos de trabajo les resulta muy útil, ya que permite ver en un simple vistazo todos los proyectos de un departamento, incluso la elaboración de informes periódicos.

Respecto al control de código e integración, SonarQube es compatible con Git y SVN, para el control de versiones y con los principales *build frameworks* como ANT, Maven o Gradle. Además, facilita la integración con sistemas de CI/CD para realizar un análisis de código automatizado y controlado con un flujo de trabajo. Este control ayuda a la detección temprana de problemas en los inicios del ciclo, ahorrando posibles problemas derivados. A través de WebAPI y Webhooks se permite el flujo de información con otras herramientas. SonarQube ofrece otro producto, SonarCloud¹⁸ como plataforma para el análisis de código continuo en la nube.

¹⁸ SonarCloud: Automatic Code Review, Testing, Inspection <https://sonarcloud.io/>

4.5.1 SonarLint

Es una extensión compatible con los principales IDE del mercado (Visual Studio, JetBrains, Eclipse, etc.) que permite la detección de errores y vulnerabilidades en el mismo momento de la escritura del código. Su funcionamiento consiste en la integración en un IDE, y muestra comentarios para detectar errores mientras se realiza el proceso de escritura del código. También se adapta a la misma configuración de reglas que SonarQube y permite la obtención de notificaciones directamente en el IDE en tiempo real cuando cambia el *Quality Gate* del proyecto, incluso da consejos sobre la calidad y la seguridad del código mientras se está desarrollando.

4.5.2 Integración con Gitlab

SonarQube se puede integrar con GitLab, permitiendo el conocimiento del estado del código a través de las métricas y la detección de problemas antes del proceso de fusión con la rama principal. El *quality gate* ayuda en esta función evitando el pase de una versión que no cumple con las reglas definidas, pudiendo bloquear automáticamente una fusión con *quality gate "failed"*. Además, SonarQube admite la autenticación delegada y la sincronización de los miembros de los grupos de Gitlab, facilitando la integración continua en los *pipelines* de cada proyecto Gitlab.

Además, dispone de *webhooks*, que son eventos que sirven para enviar información a servicios externos sobre los resultados de un proyecto analizado. Se pueden configurar varios *webhooks* por cada proyecto.

4.5.3 SonarRunner

Con esta herramienta se puede analizar la calidad código sin necesidad de una aplicación *built* tipo Maven o Ant. El funcionamiento es sencillo, se ejecuta `sonar-runner` y el resultado se carga en el cuadro de mandos de SonarQube, mostrando los valores obtenidos.

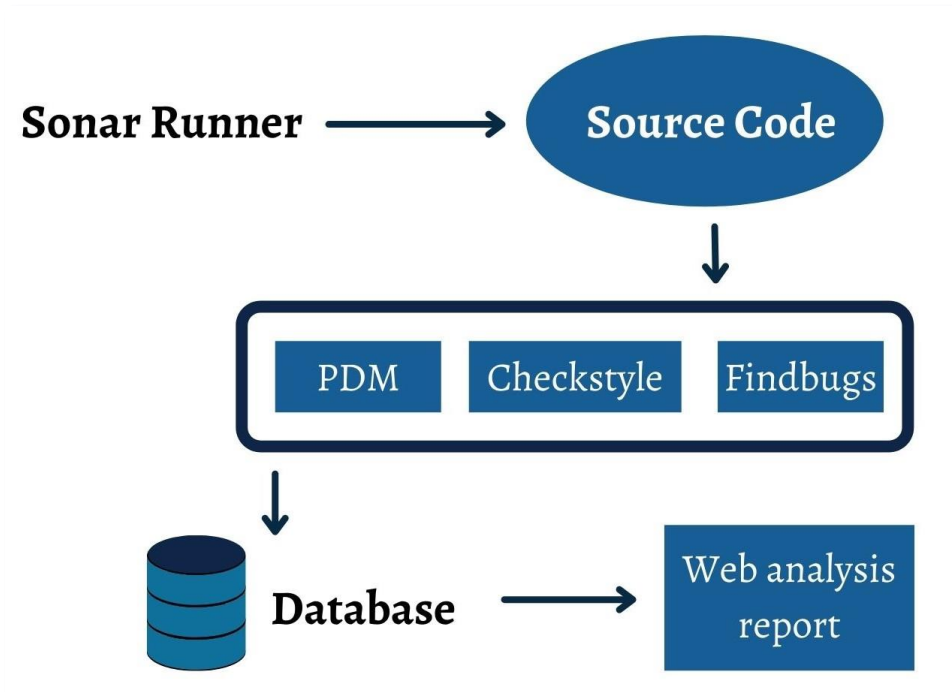


Ilustración 16-Funcionamiento de SonarRunner. Elaboración a partir de [43]

4.5.4 Otras herramientas de análisis de código

Los equipos de desarrollo dedican muchas horas en la revisión de código para lograr identificar los errores en la fase de elaboración del código. Existen diferentes metodologías que se pueden aplicar para hacer un proceso de revisión siguiendo ciertas normas y estándares para crear productos *software* de mayor calidad.

Además, para realizar este proceso, uno puede utilizar diferentes herramientas que están especializadas en facilitar la revisión del código. Ayudan en aspectos como la automatización de pruebas, la organización del código o la detección de errores tempranos en el código. Por ejemplo, herramientas como Phabricator¹⁹ para una revisión de código que incluye un plan de pruebas. Hay otras herramientas especializadas en lenguajes concretos. En el caso de Java, existen soluciones para realizar un análisis estático de código. En [44] se destacan las librerías que pueden incluirse como *plugins* en los IDE como PMD, CkeckStyle para el estilo, JaCoCo para la cobertura y Metric para la evaluación de métricas. También existen propuestas comerciales como los productos de Parasoft²⁰, que ofrece soluciones para desarrollos en C/C++, C# o .NET.

Existen otras soluciones “nicho” encaminadas al control de la seguridad del código; comerciales como CodeScan o Fortify SCA, y *opensource* como Buegsscam o Yasca [45]. En cuanto a las aplicaciones generalistas de análisis estático de código destaca Sonargraph²¹, una aplicación comercial de análisis estático de código para lenguajes de programación como Java, C#, Python o Kotlin. Dispone de herramientas específicas como Sonargraph-Architect o Sonargraph-Built con funcionalidades de control de código.

¹⁹ Phabricator. Discuss, plan, code, review and test. Phacility. <https://www.phacility.com/phabricator/>

²⁰ Productos Parasoft. Soluciones para la calidad y la seguridad continuos. <https://es.parasoft.com/request-a-demo/>

²¹ Sonargraph Overview. Hello2morrow. <https://www.hello2morrow.com/products/sonargraph>

5 Propuesta de solución

A partir de la investigación realizada se plantea la siguiente propuesta de implementación para una estrategia de gamificación integrada en el marco de la producción de la industria del *software*. Se propone la creación de un sistema de control de código con recogida de datos de manera automática ayudado con una herramienta de gestión *software*, que integra el análisis y la medición con indicadores de calidad del código de los proyectos para un entorno de la industria del *software*, aplicando una estrategia de gamificación alentadora.

Se ha elegido el ámbito de los entornos de producción de la industria del *software* sobre el de los proyectos de *Open Source* para una mejor aplicación de la estrategia de gamificación. En los *Open Source* la aplicación de estas estrategias puede provocar un aumento de la participación entre los desarrolladores pero de formas no deseables derivando en simple interés en la obtención de recompensas y olvidándose del objetivo de mejora de la calidad y de reducción de la deuda [46]. Además, la implementación de gamificación se convierte en factor motivador y de aceptación para la incorporación de nuevos elementos organizativos o de adopción de procesos de transformación en las organizaciones [47].

La estrategia de gamificación que se propone aplicar está basada en *encouraging* (estrategia alentadora). Para ello los líderes de los equipos de trabajo son los que gestionan el proceso de desarrollo proporcionando orientación y complementando acciones específicas para la mejora de la calidad del código y la reducción de la deuda técnica, fomentando la comunicación entre los miembros del equipo y el conocimiento colectivo. Incluso pudiendo realizar críticas constructivas, pero en todo momento poniendo el foco en la motivación y la concienciación sobre la importancia de la calidad del desarrollo. Además, su aplicación se plantea como estrategia corporativa, con una implicación de arriba a abajo de la organización, alineada con los objetivos de calidad y de reducción de la deuda técnica.

Se han estudiado diferentes métricas para aplicarlas a la elaboración del entorno automatizado que implemente la propuesta de estrategia de gamificación en el ámbito de la industria del *software*. Siguiendo la idea del estudio [23] se definirá un *scoring* a partir de unas métricas determinadas. Como esta propuesta es para el ámbito de la industria del *software*, a diferencia de [23], el *scoring* se calcula con valores a partir de tasas. Se han estudiado de manera más detallada a qué atributo o atributos se asigna más prioridad a partir de los pesos de la ecuación. El *scoring* que se calcula en [23] está elaborado a partir de una fórmula para el caso de un mismo proyecto, donde todos los equipos están implicados en el mismo tipo de proyecto con la misma funcionalidad. En ese caso se utilizan métricas con valores absolutos.

En cambio, en el estudio comparativo para los dos cursos académicos, puesto que cada uno tiene un proyecto diferente (para evitar el efecto aprendizaje o copia), las métricas que utiliza están expresadas como tasas y no como valores absolutos. Estas tasas se utilizan para el diseño del experimento, para comparar la calidad de los códigos en el curso en el que se aplica la estrategia de penalización frente a la calidad de los códigos del curso en el que se aplica la estrategia de recompensa. Estas tasas requieren de validación en sí mismas como métricas, por ejemplo, la validación de que el valor es mejor cuanto menor es.

En esta propuesta se formula el cálculo del *scoring* a partir de los atributos de calidad relacionados con la mantenibilidad, la fiabilidad y la seguridad. En el ámbito de la industria del *software* se pueden dar principalmente dos escenarios: desarrolladores que participan en el

mismo proyecto con contribuciones para diferentes tareas, y empresas que están llevando a cabo proyectos en paralelo y que desean tener un ranking para detectar los mejores proyectos. En ambos casos se trata de proyectos con funcionalidades y tamaños diversos, por lo que no se pueden aplicar métricas absolutas si no que se deben utilizar tasas. Para calcular el *scoring* se aplican valores multiplicadores a los coeficientes que pueden modificar el peso para darle más o menos prioridad a cada atributo. En [23], para el cálculo del *scoring* se valoran los atributos de TDR (accumulated technical debt ratio), DCD (density of duplicated code), PB_RE (rate of the potential bugs remediation effort), SV_RE (rate of the vulnerabilities remediation effort). A diferencia de [23], en este trabajo se proponen los siguientes atributos que se presentan para hacer el cálculo del *scoring*, tasas que cuanto menor valor tienen son mejores:

- ✓ *Smell density*
- ✓ *Security rate*
- ✓ *Reliability rate*
- ✓ *Cognitive complexity rate*
- ✓ *Cyclomatic complexity rate*

Se podría hacer una aproximación para elaborar un *scoring* en nuestra propuesta valorando el doble un atributo: la tasa de complejidad ciclomática. Con esta tasa, se mide el número de rutas linealmente independientes del código para determinar la complejidad de un código. De esta manera, se podría priorizar la calidad del diseño: bucles, anidación, etc. independientemente del número de líneas totales.

Una vez realizado el análisis, la información obtenida se representa a través de una aplicación web que permita la comparación de métricas entre los distintos perfiles de desarrolladores. Esto se implementa en un sistema de *leaderboard* para mostrar los desarrolladores que obtienen mejor puntuación. La información de este *ranking* se actualizará semanalmente y servirá para mostrar a los desarrolladores con mejores resultados, así como también las mejores estrategias de reducción de deuda, información muy relevante para los responsables de los desarrollos. Esta información puede ser útil en el proceso de toma de decisiones, por ejemplo, para la composición o redefinición de equipos de trabajo, o para la elaboración de las estrategias de definición y asignación de las tareas a sus miembros.

Se definen tres niveles según las insignias conseguidas por cada participante. Para conseguir cada una de ellas se tienen en cuenta el número de tareas que ha realizado, la dificultad de estas y el tiempo de su realización. En base a lo descrito, para su implementación se utiliza como sistema de control Gitlab, ya que para la instalación en un servidor propio se puede utilizar la versión gratuita (*Community Edition*), permitiendo ramificaciones protegidas con repositorios privados con equipos de trabajo amplios. Además, permite la integración continua propia y ofrece más derechos de usuario, ofreciendo una gestión de estos con roles y un registro de contenedores compatible con Gitlab CI.

El sistema de CI/CD implementa mecanismos de control para incorporar al *pipeline* del proyecto de desarrollo con la ayuda de SonarQube [41]. Con esta herramienta se podrá revisar la calidad del código teniendo en cuenta la matriz de radar de gráfico con indicadores de mantenimiento, fiabilidad, usabilidad, portabilidad y eficiencia. También revisando medidas de complejidad ciclomática, de código duplicado, test de integración o cobertura de código.

En todo el proceso de implementación se tienen en cuenta las recomendaciones de normas y estándares, en especial la norma ISO 25000, sobre sistemas y requisitos de calidad de *software* y evaluación.

5.1 Detalle de la solución

Se realiza la siguiente propuesta de solución para implementar la estrategia de gamificación para el control de la calidad y reducción de la deuda técnica. En primer lugar, se describen los requisitos de aplicación, a partir de la propuesta realizada anteriormente. Después, presenta el diseño utilizando como elementos clave SonarQube para el análisis estático y Gitlab para mostrar el resultado de los análisis y el *ranking*. Finalmente, se muestra la implementación como solución para aplicar la estrategia de control de calidad y deuda. En las siguientes secciones se detallan estas fases del análisis, diseño e implementación de la solución.

5.1.1 Análisis de requisitos

En esta sección se detalla el análisis de requisitos de la solución propuesta. El principal objetivo es la creación de una solución para aplicar la estrategia de gamificación diseñada para controlar la deuda técnica y la calidad en entornos de producción de la industria del *software*. Esto se realizará a través de la configuración de herramientas como SonarQube y Gitlab.

La condición de partida es que la implementación se integre de forma sencilla en un *workflow* de trabajo habitual para un desarrollo en un entorno de la industria de la producción *software*. Otro requisito deseable es que su integración se pueda aplicar en diferentes entornos de desarrollo de manera fácil, es decir, que la solución sea adaptable y compatible con los principales sistemas de control de calidad.

A continuación, se hace una exposición de los principales requisitos funcionales para la solución:

- El sistema permitirá la gestión de proyectos *software*.
- El sistema facilitará la gestión de la asignación de usuarios y la creación de grupos de trabajo según proyectos.
- El sistema proveerá de un sistema de control de versiones de código.
- El sistema permitirá la realización de análisis de código estático con unas métricas determinadas.
- El sistema cargará automáticamente los resultados de los análisis en el entorno del control de código.
- El sistema calculará la puntuación total de calidad del análisis a partir del valor de unas métricas determinadas
- El sistema mostrará los resultados de los análisis, ordenados de mayor a menor puntuación, en forma de *leaderboard*.

Requisitos no funcionales:

- El sistema tendrá disponibilidad 24/7, con un tiempo de respuesta rápido, mantenible y escalable; con seguridad en el acceso.
- El sistema cumplirá los estándares de calidad de aplicaciones de control de calidad de *software*.

Flujo de datos

El flujo de datos contará con las siguientes fases:

1. Inicio del <i>workflow</i>
2. Análisis del código
3. Generación de datos: <i>scoring</i>
4. Envío a Gitlab
5. Visualización de los datos

5.1.2 Diseño de la solución

A partir del análisis de requisitos, se elabora el diseño de la solución para la creación de un sistema de control de calidad de código.

5.1.2.1 Objetivo del diseño:

- Integración de la entidad SonarQube con Gitlab.
- Análisis de la calidad del código con SonarQube. Envío automático de datos a Gitlab.
- Definición de un *script* de CI/CD para la definición del *pipeline* con el flujo de trabajo del proyecto.
- Configuración de proyectos y grupos de usuario en Gitlab.
- Visualización de un listado los proyectos ordenados por *scoring*.

Como se ha descrito en el objetivo, se van a utilizar dos herramientas: GitLab y SonarQube.

A continuación, se detalla la funcionalidad que va a aportar GitLab:

- Control de código y alojamiento de las diferentes versiones de las fuentes de los proyectos.
- Gestión de proyectos, para la creación de proyectos y versionado.
- Creación de grupos de usuarios, asignados a cada proyecto.
- Gestión de usuarios, para dar de alta, modificar o asignar.
- Sistema con una interfaz para mostrar el resultado de los análisis y de *ranking* de proyectos.

Por su parte, SonarQube aporta lo siguiente:

- Análisis estático de código fuente de los proyectos
- Creación de una puntuación a partir de métricas de calidad.
- Sistema fuente de datos hacia el entorno Gitlab, para alimentar los valores del *ranking* de proyectos.

5.1.2.2 Funcionamiento esperado de la solución e integración

Funcionamiento esperado:

Desde el entorno de usuario se realiza una modificación de una versión del código fuente. Al realizar el *pull*, el sistema automáticamente hace un control de la calidad, y envía los resultados a la aplicación que los almacena. El sistema reordenará el *ranking* con los últimos valores actualizados y lo mostrará en una vista *front*.

Integración:

Para la consecución de los objetivos del diseño, GitLab y SonarQube se integran utilizando herramientas que facilitan este proceso. La integración se ha realizado de manera muy similar a [23].

Para esta integración se utiliza el flujo de trabajo de Gitlab, con GitLab CI/CD, que permite la integración y distribución continua sin necesidad de otras herramientas. En el *pipeline* se incluyen los pasos necesarios para realizar el flujo, incluyendo el análisis con Sonar. Al iniciar el proceso, desde SonarQube se recibe la petición de análisis. La herramienta realiza un análisis estático del código y obtiene unos valores para diferentes métricas de control de calidad. Gracias a un *webhook*, que se detallará más adelante, se han obtenido los valores de los indicadores y se trasladan al servidor que los almacena. También se ha realizado el cálculo del *scoring*, teniendo en cuenta los valores obtenidos por las métricas, para elaborar la puntuación del proyecto en el *ranking*. Siguiendo el criterio descrito en la sección anterior se tienen en cuenta los atributos relacionados con la mantenibilidad, la fiabilidad y la seguridad.

Esta fórmula incluye pesos para ponderar los indicadores:

- ✓ SD: *Smell density* = code smells/ncloc
- ✓ SR: *Security rate* = security remediation effort/ncloc
- ✓ RR: *Reliability rate* = reliability remediation effort/ncloc
- ✓ CoC: *Cognitive complexity rate* = cognitive complexity/ncloc
- ✓ CyC: *Cyclomatic complexity rate* = (complexity functions)/ ncloc

La ecuación propuesta para el cálculo de la valoración de un proyecto es el siguiente:

$$\text{Score} = 0.1 * \text{SD} + \text{SR} + \text{RR} + 2 * \text{CoC} + \text{CyC}$$

En esta ecuación se muestra una mayor ponderación para los indicadores de *smell density* y de *cognitive complexity*. El significado de este *score* se basa en que cuanto menor valor se obtiene, mejor calidad tendrá el proyecto, por lo que tendrá una posición superior en el *ranking*.

Características del entorno

El entorno propuesto tiene 5 servidores

1. En primer lugar, el servidor local donde el desarrollador implementa el código.
2. El servidor SonarQube.
3. El servidor de bases de datos de SonarQube: Servidor PostgreSQL.
4. El servidor de GitLab.
5. El servidor de bases de datos de GitLab: En este caso MySQL.

5.1.2.3 Descripción de la integración

La propuesta consiste en el envío de datos de las métricas determinadas y de la puntuación precalculada del *scoring* que servirá para alimentar al *ranking*. En el momento de llegar la puntuación al servidor, se almacena y actualiza el valor para el usuario. El nuevo valor se comparará con el resto de puntuaciones del *ranking*, actualizando las posiciones y mostrando la vista en el *front-end* de servicio web con el nuevo resultado del *ranking*.

El diseño de la solución se puede configurar con alternativas para que se actualice al instante o según un criterio temporal: a diario, semanalmente, etc. En este *front-end* se muestra el listado de envíos de código, la tabla de clasificación con el *ranking* actualizado y la información de los usuarios.

Integración SonarQube-GitLab

En el sistema propuesto las entidades de SonarQube y de GitLab están directamente integradas, para permitir realizar análisis de calidad desde el repositorio de proyectos.

SonarQube ofrece diferentes opciones para la creación de un nuevo proyecto en su entidad. Dispone de un configurador para las plataformas más populares: GitLab, GitHub, Bitbucket y Azure, junto a una configuración manual del proyecto.

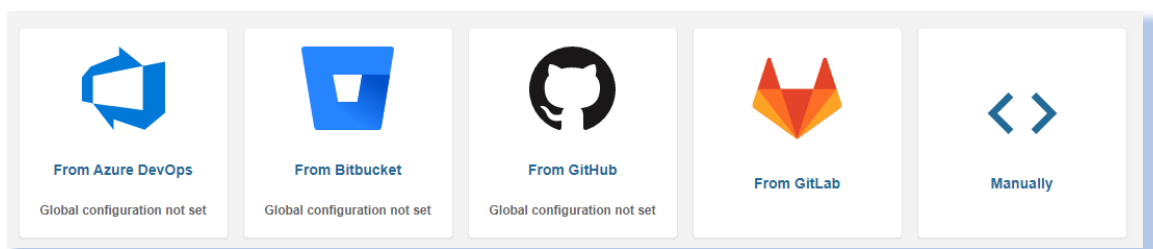


Ilustración 17 - Interfaz de SonarQube. Integración con otros entornos.

Para la creación de un nuevo proyecto de forma manual, el configurador solicita el nombre del proyecto y su *key*. A continuación, solicita la herramienta con la que se quiere realizar el análisis del repositorio. Se pueden elegir los sistemas Azure Pipeline, GitLab Ci entre otros, junto a Jenkins, muy utilizado en entornos de integración.

Con la opción *Locally*, se puede realizar un análisis local con SonarQube directamente. Para ello se tiene que crear un token, que se utiliza para identificar al proyecto cuando se realiza un análisis. Si se ha visto comprometido, puede revocarse en cualquier momento en su cuenta de usuario. Después el sistema solicita el tipo de *built* que se va a utilizar (Maven, Gradle, otros) junto al sistema operativo anfitrión. Finalmente, el sistema genera un conjunto de comandos para ejecutar en la carpeta del proyecto.

Por ejemplo, si se quiere utilizar Maven para el proyecto, se ejecuta:

```
mvn sonar:sonar \
-Dsonar.projectKey=proyecto20 \
-Dsonar.host.url=http://192.168.1.139:9000 \
-Dsonar.login=XXXXXX51900f017f7613b4dc8dd480fb13eff5c1
```

En el caso de ser para otro tipo de *built*, se necesita utilizar Sonar Scanner. Para ello se puede descargar de la página oficial²². Después se agrega el directorio */bin* como variable de entorno del *PATH*. Un ejemplo sería el siguiente conjunto de comandos:

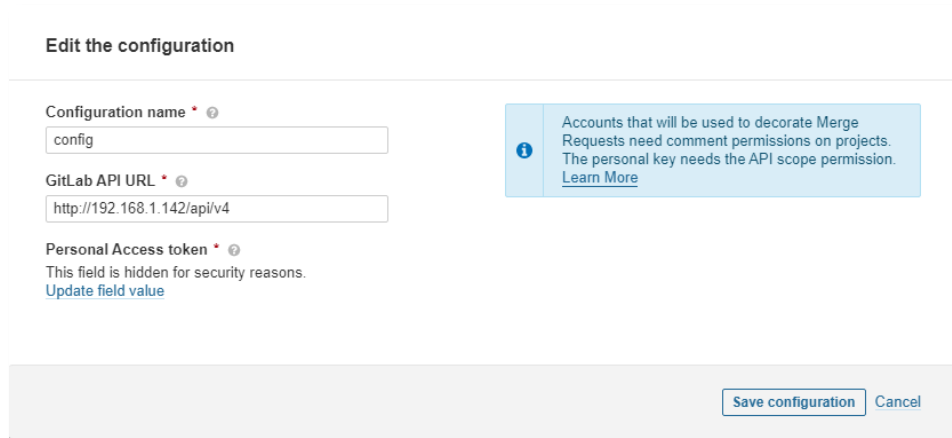
```
sonar-scanner \
-Dsonar.projectKey=proyecto20 \
-Dsonar.sources=. \
-Dsonar.host.url=http://192.168.1.139:9000 \
-Dsonar.login=XXXXXX351900f017f7613b4dc8dd480fb13eff5c1
```

²² SonarScanner download. SonarSource. <https://redirect.sonarsource.com/doc/download-scanner.html>

5. Propuesta de solución

Todo el proceso se puede automatizar integrando GitLab con SonarQube. Para ello se pueden integrar todos los proyectos en la vista de SonarQube, para realizar un análisis directamente. La configuración se realiza desde la configuración general para la instancia de SonarQube: Administración >> Configuración general >> Integraciones ALM.

Esto permite proporcionar detalles de análisis, incluso un quality gate para los pull requests directamente en la interfaz de proveedor ALM. Para su realización se debe de crear una configuración, añadiendo su nombre, la URL de la API de la entidad GitLab y el *personal Access token*.



The screenshot shows the 'Edit the configuration' page in SonarQube. It contains three input fields: 'Configuration name' with the value 'config', 'GitLab API URL' with the value 'http://192.168.1.142/api/v4', and 'Personal Access token' which is hidden. A blue information box on the right states: 'Accounts that will be used to decorate Merge Requests need comment permissions on projects. The personal key needs the API scope permission. Learn More'. At the bottom right, there are 'Save configuration' and 'Cancel' buttons.

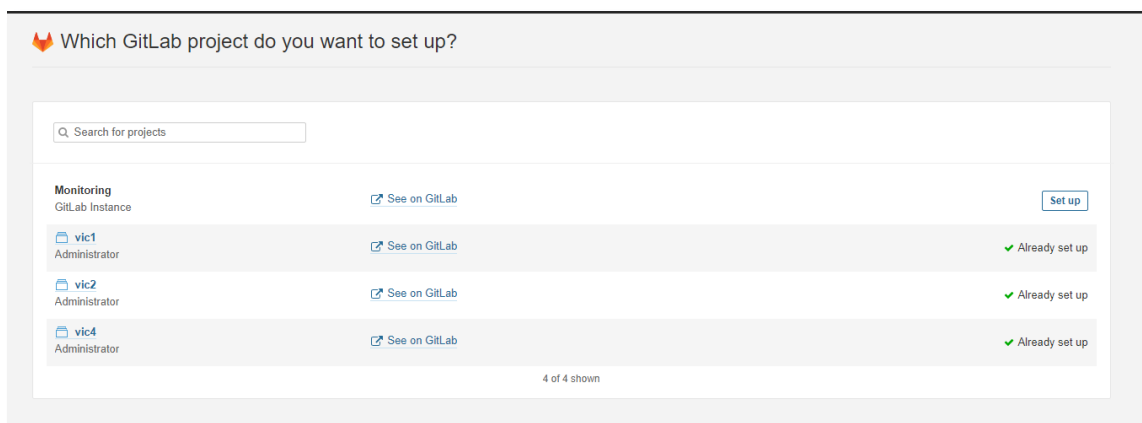
Ilustración 18 - Vista de SonarQube. Configuración integración con GitLab.

Para comprobar la API de la entidad de GitLab se puede utilizar un *curl*.

```
curl "http://192.168.1.142/api/v4/projects"
```

A continuación, se puede habilitar la autenticación de GitLab para que los usuarios del entorno inicien sesión. Para ello se debe añadir la URL de GitLab, el ID de aplicación y el *secret*. Además, la propiedad 'sonar.core.serverBaseURL' debe establecerse en la URL pública.

Una vez establecida la integración, desde la vista de SonarQube, permitirá tener un listado de proyectos que están en GitLab, para realizar un análisis estático directamente.



The screenshot shows the 'Which GitLab project do you want to set up?' page. It features a search bar and a list of projects. The first project is 'Monitoring' (GitLab Instance) with a 'Set up' button. The other three projects are 'vic1 Administrator', 'vic2 Administrator', and 'vic4 Administrator', each with a 'See on GitLab' link and a green checkmark indicating they are 'Already set up'. The page shows '4 of 4 shown' at the bottom.

Ilustración 19 - Vista de SonarQube. Listado proyectos GitLab

5. Propuesta de solución

Además, también permite realizar un análisis, por ejemplo, utilizando GitLab CI. Su realización consta de tres fases:

1. Añadir la project key. Para ello se crea en el repositorio el sonar-project.properties

```
sonar.projectKey=root_vic4_AXvm3opjsq1rX6bRQE6_  
sonar.qualitygate.wait=true
```

2. Definir las variables de entorno. Desde la configuración de GitLab se añaden las variables de entorno: Settings >> CI/CD >> Variables

Las dos variables que se debe de completar son las siguientes:

- SONAR_TOKEN
- SONAR_HOST_URL

3. Se añade al fichero de configuración (*.gitlab-ci.yml*) los datos del proyecto. Por ejemplo, con la siguiente definición:

```
sonarqube-check:  
  image:  
    name: sonarsource/sonar-scanner-cli:latest  
    entrypoint: [""]  
  variables:  
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar" # Defines the location of the  
analysis task cache  
    GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required  
by the analysis task  
  cache:  
    key: "${CI_JOB_NAME}"  
    paths:  
      - .sonar/cache  
  script:  
    - sonar-scanner  
  allow_failure: true  
  only:  
    - master # or the name of your main branch
```

WebHook en SonarQube

En el caso de SonarQube, para que llegue el análisis de forma correcta se tiene que configurar un *webhook*. Este será el encargado de enviar todos los resultados a la aplicación para mostrar la información del *ranking*.

Para el envío de información desde SonarQube a GitLab, se define este *webhook* como servicio web en propio servidor de Sonar. En SonarQube se definen los *webhook* dentro de la propia interfaz web en >>Project Settings >> Webhooks.

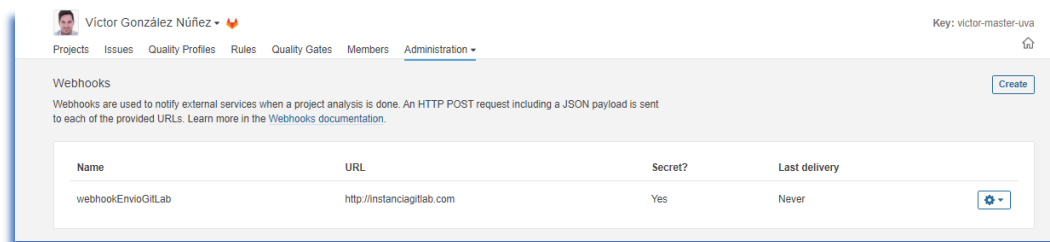


Ilustración 20 - Interfaz de SonarQube. Gestión de Webhooks

5. Propuesta de solución

Este *webhook* calcula la puntuación total a partir de la fórmula del *scoring* y envía la información al servidor GitLab. Su funcionamiento consiste en que cuando un desarrollador ejecuta un análisis, se invoque un servicio web. El sistema llama a través de la API REST de SonarQube para recoger estos datos. En el servidor de GitLab se almacena toda la información para actualizar el *ranking*.

El funcionamiento de la solución según la integración propuesta es la siguiente:

1.- El desarrollador trabaja en una máquina como servidor local realizando modificaciones en una rama del proyecto.

2.- El desarrollador lanza el análisis con SonarQube utilizando SonarScanner.

3.- En ese momento el *webhook* entra en escena y llama al servicio web del proyecto. El servicio web llama a la API REST de SonarQube para recoger las métricas con las que se elabora el *scoring*.

4.- La información es enviada, tanto el valor precalculado del *scoring* del proyecto como de los atributos que influyen en un posible desempate.

5.- La información se almacena y se actualiza para el usuario. El *ranking* del proyecto se actualiza según la nueva puntuación recogida.

6.- El desarrollador puede revisar el resultado obtenido y su nueva posición en el *ranking*.

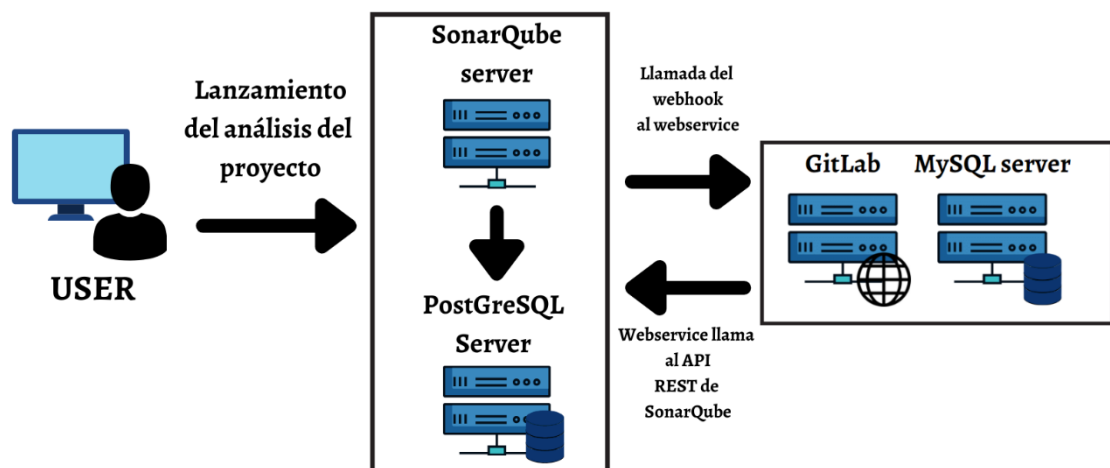


Ilustración 21 - Diagrama de la solución propuesta. Lanzamiento del análisis.

Empate de puntuaciones

Se puede producir una situación de empate de puntuaciones. En este caso, se propone que para el desempate se realice una comparación de dos atributos. Primeramente, se tendrá en cuenta el valor de la complejidad ciclomática, con una posición superior en caso de menor valor. En caso de repetirse el empate, se comparará el valor de la complejidad cognitiva, con una posición superior en caso de menor valor.

Esto implica que el sistema desarrollado con el *webhook* debe de enviar tanto la puntuación total precalculada como los atributos de desempate. Por lo que para cada análisis realizado se tiene que almacenar:

- El nombre, fecha de realización y otros datos identificativos del análisis.
- El *scoring* precalculado.
- El valor de los atributos de desempate.

Otros métodos de *scoring*

Alternativamente, otra de las opciones para aplicar en la elaboración del *scoring* es añadir un conjunto de condiciones previas de calidad de código, para que se realice la actualización de la versión. En caso de no cumplirse, el código estará en "*failed*". Aplicando esta metodología, se podría implementar un sistema para que los códigos que estén en este estado no puedan incorporarse al *ranking* o aparezca con puntuación que incluya algún tipo de penalización.

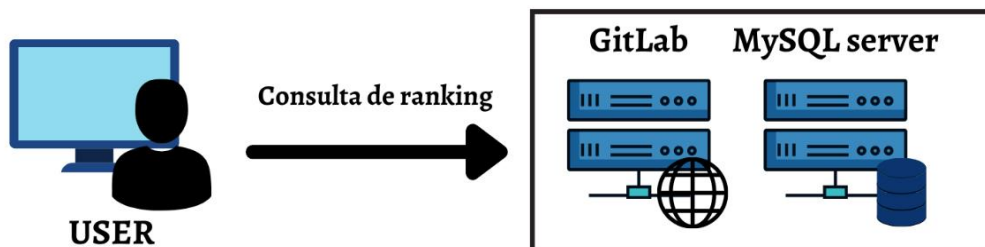


Ilustración 22 Diagrama de la solución propuesta. Consulta del ranking

Implementación del *ranking*.

La implementación del *ranking* se realiza en el servidor de GitLab. Una vez recogidos los datos después de la llamada a la API REST de SonarQube, se almacenan en el servidor. Esta información que se recibe son los datos del proyecto analizado, el resultado precalculado del *scoring* y los valores de los atributos de desempate.

Para que esta información se muestre en GitLab, se tiene que realizar una modificación de la entidad. De manera nativa, GitLab no permite mostrar más campos que los de su configuración estándar de proyectos. En la siguiente sección se explicará el uso de hooks para realizar modificaciones en la instancia GitLab con el objetivo de mostrar un ranking.

A continuación, se muestra un prototipo de vista en GitLab con el conjunto de proyectos. En el mismo se ofrece un listado ordenado con los proyectos, junto su posición en el ranking y su puntuación, además de la fecha de actualización.

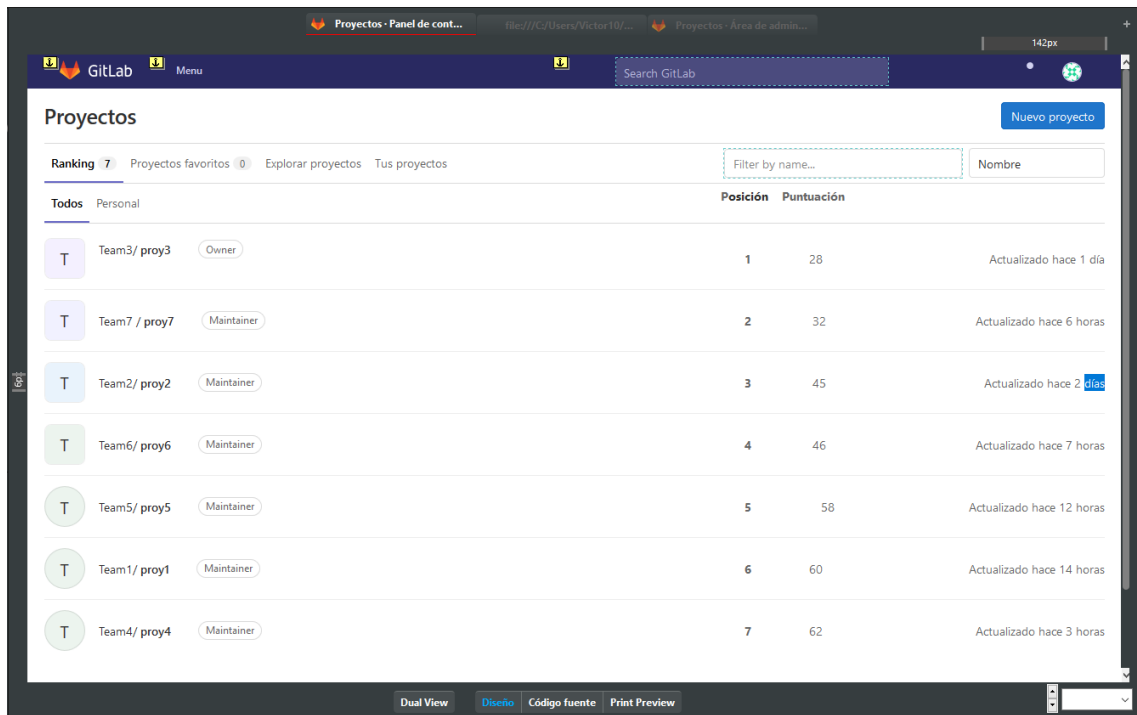


Ilustración 23 - Vista del ranking. Prototipo.

File Hooks en GitLab

A través de la utilización de *hooks*, se puede conseguir crear un *leaderboard* para mostrar el *ranking* de proyectos de usuarios. Los *hooks* permiten alterar el comportamiento de un componente *software*. En el caso de GitLab, se pueden utilizar para realizar integraciones personalizadas sin modificar el código fuente. Su aplicación se puede realizar solicitudes de HTTP POST en diferentes eventos.

Configuración de los *file hooks*:

1.- Para la configuración de estos *file hooks*, se deben de situar en el directorio de `file_hooks` de la entidad. En una instalación con la versión de GitLab CE se encuentra en

```
/home/git/gitlab/file_hooks/
```

2.- Dentro del directorio se crear el *hook*. Este fichero tiene que tener permisos de propiedad del usuario de Git.

3.- Los datos del archivo *hook* se proporcionan como JSON.

4.- Se puede revisar el log de funcionamiento en:

```
log/file_hook.log
```

Existen diferentes *hooks*, por ejemplo, para crear mensajes automáticos de creación de un nuevo proyecto.

Para crear un *hook* de sistema se puede hacer desde el menú de *system hooks*. Desde la interfaz de GitLab >>menú >> admin >> System Hooks

Se añade la URL y el *token* secreto. Se puede habilitar la verificación SSL.

Se configura y se agrega el enlace del sistema.

La documentación oficial de GitLab muestra todos los *hooks* del sistema²³.

Requisitos técnicos

El entorno propuesto y las herramientas de implementación deben tener unas características de requisitos técnicos para un correcto funcionamiento. En los apéndices de la memoria se amplía esta información.

- SonarQube. Características del sistema anfitrión:
 - Servidor con 2GB de RAM
 - Java JDK instalado
 - Base de datos: MySQL, Oracle o PostgreSQL

Más información en la documentación oficial de SonarQube, en la sección de *requirements*²⁴.

- Gitlab. Características del sistema anfitrión.
 - Servidor con 4GB de RAM
 - Base de datos PostgreSQL

Más información en la documentación oficial de GitLab²⁵.

Especificaciones de mantenimiento.

Finalmente, se describen unos consejos para el mantenimiento del sistema implementado en la propuesta:

- Gestión ágil de usuarios: altas, modificaciones, bajas.
- Limpieza con frecuencia determinada de versiones de código que ya no aportan valor a la organización.
- El sistema debe proveer de una configuración de copias de seguridad y de recuperación en un tiempo “aceptable”.
- Será necesario la revisión con frecuencia de las versiones de Gitlab y SonarQube, así como las actualizaciones de los Sistemas Operativos anfitriones, para que el sistema esté actualizado con mayor seguridad para el entorno.

5.1.3 Implementación y pruebas. Escenarios

Una vez diseñada la solución se puso en marcha su implementación. Para ello se utilizó una infraestructura de máquinas virtuales con sistemas Ubuntu y CentOS. Se trabajaron varios escenarios de pruebas para comprobar la funcionalidad de las herramientas y valorar la conveniencia de cada una de ellas respecto al diseño de la solución.

²³ GitLab System hooks. GitLab Docs. https://docs.gitlab.com/ee/system_hooks/system_hooks.html

²⁴ Requirements. SonarQube Docs. <https://docs.sonarqube.org/latest/requirements/requirements/>

²⁵ Install requirements. GitLab Docs. <https://docs.gitlab.com/ee/install/requirements.html>

En la sección de apéndices de esta memoria aparecen las guías de instalación de las herramientas utilizadas con el detalle de su configuración. En todos los escenarios se han tenido en cuenta las características que deben de tener para que estas herramientas funcionen correctamente: versión de SO, almacenamiento, memoria RAM, etc.

Las pruebas se han realizado usando un código que contiene errores. El código utilizado es el código Damn Vulnerable Web Application (DVWA)²⁶, que es una aplicación web en PHP que tiene deficiencias en el código y resulta muy vulnerable.

En la siguiente sección se muestran los escenarios que se plantearon para la prueba de las herramientas y sus funcionalidades detalladas en la sección de entorno tecnológico de esta memoria.

5.1.3.1 1^{er} escenario

Control de versiones local y gitlab.com

Requisitos:

Máquina virtual con la versión de Ubuntu 20.04.3 LTS

Instalación de GIT 2.32.0

Cuenta de usuario en gitlab.com

Resumen:

Se realizaron las pruebas iniciales para comprobar el funcionamiento de GitLab.com y el control de versiones de código. Las pruebas consistieron en realizar actualizaciones de las diferentes ramas de un proyecto. Para ello se instaló Git en una máquina local, y se configuró su comunicación con la instancia de usuario en gitlab.com

Resultado:

Se realizó una actualización de una rama de un proyecto, y los cambios quedaron registrados y controlados en Gitlab.com.

5.1.3.2 2^o escenario

Control de versiones local y entorno Gitlab.

Requisitos:

Máquina virtual con la versión de Ubuntu 20.04.3 LTS

Instalación de Git 2.32.0

Instalación de GitLab CE 14.0.1

Resumen:

Se realizaron las pruebas iniciales para comprobar el funcionamiento de un entorno local de Gitlab y el control de versiones de código. GitLab ofrece la posibilidad de descargar, instalar y mantener una instancia de GitLab para un entorno local. De esta manera, se consigue un mayor control y una mejor personalización del entorno que en gitlab.com

Las pruebas consistieron en realizar actualizaciones de las diferentes ramas de un proyecto. Para ello se instaló Git en una máquina local. En la misma máquina se instaló una instancia de Gitlab, para ver los resultados directamente. Se configuró su comunicación entre instancias. Alternativamente, sin necesidad de Git, se podría haber realizado la actualización de la rama.

²⁶ Damn Vulnerable Web Application (DVWA). <https://dvwa.co.uk/>

Resultado:

Se realizó una actualización de una rama de un proyecto, y los cambios quedaron registrados y controlados en Gitlab.

5.1.3.3 3^{er} escenario

Análisis de calidad de código con SonarCloud

Requisitos:

Máquina virtual con la versión de Ubuntu Ubuntu 20.04.3 LTS

Resumen:

SonarCloud es una instancia de SonarQube, pero en versión de aplicación en la nube. Desde un repositorio local se realizó un análisis estático de código contra esta instancia. Con un código fuente establecido en un directorio se envió a esta instancia. En la SonarCloud apareció el resultado.

Resultado:

Se realizó un análisis estático de código, mostrando diferentes métricas de control de calidad en la instancia en la nube.

5.1.3.4 4^º escenario

Análisis de calidad de código con SonarQube

Requisitos:

Máquina virtual con la versión de Ubuntu Ubuntu 20.04.3 LTS

La instalación de SonarQube es la versión 9.0.1.46 CE

Resumen:

Se instaló en una máquina local una instancia de Sonar. Desde un repositorio local se realizó un análisis estático de código con SonarQube. Con un código fuente establecido en un directorio se ejecutó el *sonar-scanner*. Con ello se envió el código a la instancia de SonarQube, para que lo analizara. En la instancia apareció el resultado del análisis. Alternativamente, se realizó una prueba similar, pero con una instancia de SonarQube generada con un contenedor Docker.

Resultado:

Se realizó un análisis estático de código, mostrando diferentes métricas de control de calidad en SonarQube.

5.1.3.5 5^º escenario

Análisis de calidad de código con SonarQube e integración con Gitlab.

Requisitos:

Máquina virtual con la versión de Ubuntu Ubuntu 20.04.3 LTS x2

Instalación de SonarQube es la versión 9.0.1.46 CE

Instalación de GitLab CE 14.0.1

Resumen:

Se instaló en una máquina local una instancia de SonarQube. En otra máquina se realizó una instalación de GitLab CE. En un primer momento se hizo en la misma máquina, pero la configuración del acceso por URL generó problemas.

En la primera máquina se ejecutó un análisis estático de código. Se configuró el *webhook* en SonarQube. Se ejecutó dicho *webhook*, calculando el *scoring* a partir de los datos de las métricas analizadas. Después el código se actualizó en la rama correspondiente, según se ha definido en el *pipeline* de Gitlab CI/CD, y llega a GitLab. El resultado se mostró como *ranking* de valoraciones de calidad de proyectos, actualizándose según la puntuación obtenida.

Resultado:

Se realizó el análisis y con el *webhook* se enviaron los resultados. Se mostró el proyecto en GitLab. Queda como trabajo futuro el implementar el código necesario para que el *ranking* se muestre en la vista *front-end* en GitLab.

6 Conclusiones y trabajo futuro

En este trabajo se ha estudiado el ámbito de la producción de *software* aplicando estrategias de gamificación para ayudar en la motivación de los desarrolladores, con el objetivo de conseguir una mayor calidad del código y una reducción de la deuda técnica. Estas diferentes estrategias intervienen en la motivación de los equipos de trabajo.

Actualmente se están aplicando estrategias de gamificación en diversos ámbitos como la educación, o en áreas relacionadas con la Ingeniería del *Software* como la gestión de procesos, la calidad, o las metodologías de evaluación y gestión de proyectos. La implementación de gamificación se convierte en un factor motivador y de adopción de procesos organizacionales.

En cuanto a la calidad del *software*, se ha profundizado en el concepto de deuda técnica y su tipología con las subdivisiones que propone Martin Fowler en su cuadrante con los tipos de deuda imprudente, prudente, inadvertida y deliberada. Se han presentado cuatro estrategias para reducir la deuda técnica, llegando a la conclusión de que la mejor es la estrategia alentadora. En esta estrategia los líderes de los equipos de trabajo proporcionan orientación y acciones específicas para la mejora de los desarrollos, poniendo el foco en la concienciación y la motivación del equipo de trabajo. Toda estrategia tiene elementos comunes para su diseño y aplicación teniendo en cuenta los participantes a los que va dirigidos y los factores psicológicos que les influyen, por lo que en otros ámbitos diferentes pueden ser otras las más recomendables. Un ejemplo es el ámbito educativo, en donde tiene mejor resultado la estrategia de recompensa frente a la de penalización.

La gestión de proyectos requiere de una planificación realista a partir de los recursos con los que se cuenta. Un factor importante es la experiencia, ya que se aprende de los éxitos y de los fracasos del pasado. El equipo de trabajo debe de contar con una figura con un liderazgo claro y fuerte, que proponga una metodología con procedimientos adaptados al proyecto, existiendo guías y normas para la definición del proceso de construcción *software*.

Para mejorar la calidad en los proyectos existen medidas como la evaluación y medición de diferentes factores que influyen en la consecución de un buen producto. Hay informes como el *Chaos Report* que tratan sobre estos factores y sobre el éxito o fracaso de los proyectos *software*. Una metodología que ayuda a la buena consecución de los proyectos es DevOps CI/CD, que se centra en la mejora de los procesos de desarrollo a través de buenas prácticas para la integración y entrega continuas y las pruebas.

Por otro lado, la aplicación de sistemas de control de versiones permite una gestión robusta de las tareas y del historial de cambios en los desarrollos, fomentando la colaboración y la aplicación de buenas prácticas, mejorando en aspectos como los plazos de entrega.

Teniendo en cuenta el entorno tecnológico, se ha realizado una propuesta de solución de un sistema de control de calidad de código basado en la integración y distribución continua, implementando una estrategia de gamificación alentadora, con un entorno automatizado que permite medir la calidad del código con diferentes atributos y que genera un *scoring* para destacar los proyectos con mejores valores según los indicadores de calidad.

Esta propuesta se basa en un sistema de control de código con recogida de datos de manera automática en entornos de producción de la industria del *software* con ayuda de SonarQube y GitLab.

SonarQube, verifica la calidad del código permitiendo la realización de análisis estáticos mostrando diferentes métricas sobre indicadores de calidad del código: *code smells*, *bugs*, complejidad cognitiva o test unitarios y de integración. Dispone de la extensión SonarLint, para los principales IDE, y de una versión web como es SonarCloud.

Por otro lado, GitLab es un servicio web de control de versiones basado en Git para el desarrollo de forma colaborativa. Dispone de GitLab CI/CD para la integración y entrega continua, que permite una gestión integrada y una entrega rápida y eficiente a través de la creación de flujos de trabajo con *pipelines*, para diseñar las diferentes fases de integración.

La solución integra estas dos herramientas a través de la implementación con GitLab CI/CD, donde se incluyen los mecanismos de control en el *workflow* de los proyectos para el análisis de la calidad del código a partir de indicadores de calidad, calculando un *scoring* a partir de atributos relacionados con la mantenibilidad, la fiabilidad y la seguridad de cada uno de los proyectos, todo ello teniendo en cuenta normas y estándares como ISO 25000.

6.1 Valoración global de la actividad

En cuanto a la valoración global de la actividad, considero que ha sido muy útil académicamente. En primer lugar, por aplicar las fases de los procesos de investigación, en especial en la planificación y la elaboración de la memoria. La revisión de artículos de diferentes fuentes me ha permitido conocer sitios de referencia con fuentes de mucha calidad para obtener materiales útiles para el estudio e investigación.

En segundo lugar, los contenidos estudiados me han servido para ampliar mis conocimientos sobre estrategias de gamificación, elemento que considero muy innovador, y para profundizar en la calidad del código y la deuda técnica, muy útil para la mejora de los desarrollos. En cuanto al entorno tecnológico, me ha servido para conocer los entornos de producción CI/CD y ahondar en soluciones de control de calidad, así como sus alternativas.

En este proyecto he podido comprobar el potencial de herramientas como Gitlab y la integración con CI/CD, además de los beneficios de SonarQube en la realización de análisis estático de código. Es importante hacer este tipo de análisis en cualquier proyecto *software* para la mejora del trabajo en equipo y el ahorro de recursos organizacionales. El diseño de la propuesta de solución ha sido muy útil para conocer en detalle estos entornos, tanto a nivel de configuración como de funcionalidad.

6.2 Futuras líneas de trabajo

Una vez finalizada la memoria, surgen diferentes líneas de trabajo futuro, según diferentes ámbitos de aplicación. En primer lugar, se podría continuar el trabajo realizando un diseño más detallado del ranking, con una implementación a partir del uso de *hooks* y *badgets* de GitLab.

Además, se podría estudiar realizar diferentes propuestas de implementación con otras opciones de integración. Se puede extender el uso de la implementación a otras herramientas como Maven o Gradle para la automatización del código. Además, existen otros sistemas de integración continua, como Jenkins, Travis CI o Bamboo Atlassian, que pueden estar ya establecidas en las organizaciones. Su integración sencilla favorecería la implantación y asimilación por parte de los equipos de trabajo en entornos tecnológicamente compatibles, de los que ya pueden estar familiarizados con su uso.

En cuanto a la integración de la solución, se podría detallar su implementación en servidores profesionales con una personalización de la configuración según la distribución de sistema operativo o mediante la optimización de despliegues con contenedores como Docker, que pueden proporcionar una automatización de la virtualización de las aplicaciones integradas.

Respecto a la propuesta de *scoring*, se podría valorar cambiar la valoración para ponderar más otras métricas como, por ejemplo, la tasa de complejidad ciclomática. La definición de los umbrales para las métricas según las necesidades organizacionales, analizados a través de los indicadores, puede favorecer la comprobación de que la mejora en los desarrollos es correlativa con los valores recogidos en el *score*. Esto podría provocar que la fórmula se tuviera que rehacer modificando la ponderación de cada atributo añadiendo o eliminando alguno según las necesidades organizacionales. Por ello, se podría realizar un estudio con entrevistas y/o encuestas para valorar las métricas que influyen más en la calidad de los desarrollos según el tipo de organizaciones, el tamaño de estas o tipo de proyectos *software* que realizan, con el objetivo de adaptar el *scoring* de una manera más personalizada a las necesidades organizacionales.

Finalmente, se podría profundizar en implementaciones basadas en otras estrategias de motivación, como puede ser la gratificación, a partir del aumento de incentivos laborales como el incremento de la retribución o las condiciones de trabajo: horas libres, flexibilidad de entrada/salida, etc. También en otros aspectos motivacionales del trabajo en los equipos de desarrollo: salarios, movilidad, tipo de proyecto o nivel de creatividad.

7 Bibliografía

Referencias bibliográficas

- [1] M. Trinidad, A. Calderón, and M. Ruiz, "GoRace: A Multi-Context and Narrative-Based Gamification Suite to Overcome Gamification Technological Challenges," *IEEE Access*, vol. PP, p. 1, Apr. 2021, doi: 10.1109/ACCESS.2021.3076291.
- [2] G. F. Tondello, R. R. Wehbe, L. Diamond, M. Busch, A. Marczewski, and L. E. Nacke, "The gamification user types Hexad scale," in *CHI PLAY 2016 - Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, Oct. 2016, pp. 229–243, doi: 10.1145/2967934.2968082.
- [3] S. Silva, "La Cultura Ciudadana como proyecto ciudadano," 2016, in *Pensar y construir el territorio desde la cultura* Edition: 1Publisher: Universidad EAFIT - Alcaldía de Medellín Editors: Camilo Tamayo pp. 5–8.
- [4] C. Mario, "Theories of Self-determination," in *Global Encyclopedia of Public Administration, Public Policy, and Governance*, Springer International Publishing, 2019, pp. 1–6.
- [5] A. Mittelmark, "Enterprise gamification. Buzzword or business tool," *PwC*, 2021.
- [6] D. de P. Porto, G. M. de Jesus, F. C. Ferrari, and S. C. P. F. Fabbri, "Initiatives and challenges of using gamification in software engineering: A Systematic Mapping," *J. Syst. Softw.*, vol. 173, Mar. 2021, doi: 10.1016/j.jss.2020.110870.
- [7] M. S. Bianciotti, C. Salgado, M. Peralta, and A. Sánchez, "Gestión de proyecto de software: un método basado en gamificación para mejorar la calidad del producto y desempeño de equipos de desarrollo," *Departamento de Informática Facultad de Ciencias Físico-Matemáticas y Naturales Universidad Nacional de San Luis*. <http://sedici.unlp.edu.ar/handle/10915/62020> (accessed Jun. 03, 2021).
- [8] G. Brougère, "Paradoxes of Gamification," in *The Gamification of Society*, Wiley, 2021, pp. 1–18.
- [9] G. A. García-Mireles and M. E. Morales-Trujillo, "Gamification in Software Engineering: A Tertiary Study," in *Advances in Intelligent Systems and Computing*, Oct. 2020, vol. 1071, pp. 116–128, doi: 10.1007/978-3-030-33547-2_10.
- [10] M. M. Alhammad and A. M. Moreno, "What is going on in agile gamification?," in *ACM International Conference Proceeding Series*, 2018, vol. Part F147763, doi: 10.1145/3234152.3234161.
- [11] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering - A systematic mapping," in *Information and Software Technology*, Jan. 2015, vol. 57, no. 1, pp. 157–168, doi: 10.1016/j.infsof.2014.08.007.
- [12] E. E. Briceño Arceo, R. A. Aguilar Vera, J. C. Díaz Mendoza, and J. P. Ucán Pech, "Gamificación para la mejora de procesos en ingeniería de software: Un estudio exploratorio," *ReCIBE. Rev. electrónica Comput. Informática, Biomédica y Electrónica*, vol. 8, no. 1, pp. 1–19, Jun. 2019, [Online]. Available: <https://www.redalyc.org/articulo.oa?id=512259512004>.

- [13] R. Atal, "ANUKARNA : A Software Engineering Simulation Game for Teaching Practical Decision Making in Peer Code Review," 2015. Accessed: Jun. 03, 2021. [Online]. Available: <https://repository.iiitd.edu.in/xmlui/handle/123456789/279>.
- [14] B. Marín, J. Frez, J. Cruz-Lemus, and M. Genero, "An empirical investigation on the benefits of gamification in programming courses," *ACM Trans. Comput. Educ.*, vol. 19, no. 1, Jan. 2019, doi: 10.1145/3231709.
- [15] B. Marín, "Lessons Learned About Gamification in Software Engineering Education," 2020, pp. 174–197.
- [16] E. Özkul, E. Uygun, and S. Levent, "Digital Gamification in the Tourism Industry," 2020, pp. 169–203.
- [17] E. Allman, "Managing technical debt," *Commun. ACM*, vol. 55, no. 5, pp. 50–55, May 2012, doi: 10.1145/2160718.2160733.
- [18] M. Fowler, "TechnicalDebtQuadrant." <https://martinfowler.com/bliki/TechnicalDebtQuadrant.html> (accessed Jun. 03, 2021).
- [19] M. Wiese, M. Riebisch, and J. Schwarze, "Preventing Technical Debt by Technical Debt Aware Project Management," Mar. 2021, Accessed: Jun. 03, 2021. [Online]. Available: <http://arxiv.org/abs/2103.10317>.
- [20] N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners," *Information and Software Technology*, vol. 102. Elsevier B.V., pp. 117–145, Oct. 01, 2018, doi: 10.1016/j.infsof.2018.05.010.
- [21] T. Besker, A. Martini, and J. Bosch, "Technical debt triage in backlog management," in *Proceedings - 2019 IEEE/ACM International Conference on Technical Debt, TechDebt 2019*, May 2019, pp. 13–22, doi: 10.1109/TechDebt.2019.00010.
- [22] T. Besker, A. Martini, and J. Bosch, "Carrot and stick approaches when managing technical debt," in *Proceedings - 2020 IEEE/ACM International Conference on Technical Debt, TechDebt 2020*, Jun. 2020, pp. 21–30, doi: 10.1145/3387906.3388619.
- [23] Y. Crespo, A. Gonzalez-Escribano, and M. Piattini, "Carrot and Stick approaches revisited when managing Technical Debt in an educational context," Apr. 2021, Accessed: Jun. 03, 2021. [Online]. Available: <http://arxiv.org/abs/2104.08993>.
- [24] B. Jammeh, "DevSecOps: Security Expertise a Key to Automated Testing in CI/CD Pipeline," Dec. 2020.
- [25] D. Ašeriškis, T. Blažauskas, and R. Damaševičius, "UAREI: Un modelo para la descripción formal y la representación visual de la gamificación de software," *DYNA*, vol. 84, no. 200, pp. 326–334, 2017, doi: 10.15446/dyna.v84n200.54017.
- [26] S. Nikkila, S. Lin, H. Sundaram, and A. Kelliher, "Playing in taskville: Designing a social game for the workplace," *Work. Gamification Using Game Des. Elem. Non-Gaming Context.*, pp. 1–4, Jan. 2011.
- [27] D. Martínez Cardero and M. González Arencibia, "Habilidades creativas en equipos de desarrollo de software," *Revista Atlante: Cuadernos de Educación y Desarrollo (septiembre 2019)*. e. <https://www.eumed.net/rev/atlante/2019/09/habilidades-creativas-software.html> (accessed Sep. 10, 2021).
- [28] Miguel Angel Hernández de la Rosa and E. H. Luque, "La Producción de Software como

- Labor Motivadora para el Desarrollo de la Creatividad,” Accessed: Sep. 10, 2021. [Online]. Available: <https://www.econlink.com.ar/produccion-de-software-creatividad>.
- [29] P. G. Neumann, “Illustrative Risks to the Public in the Use of Computer Systems and Related Technology,” *Computer Science Laboratory SRI International, Menlo Park CA 94025-3493*. <http://www.csl.sri.com/users/neumann/illustrative.html> (accessed Aug. 30, 2021).
- [30] M. Ben-Menachem and G. S. Marliss, “Software quality : producing practical, consistent software,” p. 326.
- [31] G. Lompfrey¹ and S. Hernandez, “La importancia de la calidad en el desarrollo de productos de software,” *Fac. Ing. y Tecnol. Univ. Morelia, México , Technical Rep. COMP-018-2008*, 2008.
- [32] A. Rastogi, S. Thummalapenta, T. Zimmermann, N. Nagappan, and J. Czerwonka, “Ramp-up Journey of New Hires: Do strategic practices of software companies influence productivity?,” *ACM Int. Conf. Proceeding Ser.*, pp. 107–111, Feb. 2017, doi: 10.1145/3021460.3021471.
- [33] F. Brooks, *Mythical Man-Month, The: Essays on Software Engineering, Anniversary Edition : Frederick, Brooks: Amazon.es: Libros*. Boston, MA: Addison Wesley, 1995.
- [34] G. Maturro, K. Barrella, P. Benitez, M. I. Lund, U. Nacional, and S. Juan, *Dificultades de los “recién llegados” a proyectos software en ejecución*. 2017.
- [35] M. Callejas Cuervo, A. Aldana, and A. Álvarez-Carreño, “Modelos de calidad del software, un estado del arte,” *Entramado*, vol. 13, Jun. 2017, doi: 10.18041/entramado.2017v13n1.25125.
- [36] B. Nuñez-Moraleda, N. Moraleda, and C. Angel, “El modelo de mccall como aplicación de la calidad a la revision del software de gestion empresarial,” *monografias.com*, Jan. 2000.
- [37] J. Garzas, “Mitos y misterios de la antigüedad de la ingeniería software - Javier Garzas.” <https://www.javiergarzas.com/2015/12/mitos-y-misterios-de-la-antiguedad-de-la-ingenieria-software.html> (accessed Sep. 11, 2021).
- [38] The Standish Group, “Chaos Report 2015,” 2015. https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (accessed Sep. 11, 2021).
- [39] ATlassian, “Qué es Git: conviértete en todo un experto en Git con esta guía,” 2021. <https://www.atlassian.com/es/git/tutorials/what-is-git> (accessed Sep. 11, 2021).
- [40] About GitLab and J. Chen, “Building an Open Source Company: Interview with GitLab’s CEO | GitLab,” 2016. <https://about.gitlab.com/blog/2016/07/14/building-an-open-source-company-interview-with-gitlabs-ceo/> (accessed Sep. 11, 2021).
- [41] “GitLab Integration | SonarQube Docs.” <https://docs.sonarqube.org/latest/analysis/gitlab-integration/> (accessed Sep. 03, 2021).
- [42] Devalatio’s Blog, “Pipelines en Gitlab, el camino hacia el CI/CD (parte I) | Devalatio’s Blog,” Mar. 02, 2020. <https://blog.devalat.io/pipelines-en-gitlab-el-camino-hacia-el-ci-cd-parte-i/> (accessed Sep. 11, 2021).
- [43] SonarQube Docs, “Configuring Analysis | SonarQube Docs,” 2021. <https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-maven/> (accessed Sep. 11, 2021).

- [44] M. Tumino and J. Bournissen, “Prácticas de Desarrollo Software: Un Estudio Exploratorio con Herramientas de Análisis Estático,” *Rev. Latinoam. Ing. Softw.*, vol. 3, p. 155, Oct. 2015, doi: 10.18294/relais.2015.155-160.
- [45] L. Rodríguez, OWASP, and ALS Software Lifecycle Optimization, “Herramientas para el análisis estático de seguridad y estado del arte,” 2018. Accessed: Sep. 11, 2021. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Spain_20080314_Herramientas_de_análisis_estático_de_seguridad_d_el_código_estado_del_arte.pdf.
- [46] L. Moldon, M. Strohmaier, and J. Wachs, “How Gamification Affects Software Developers: Cautionary Evidence from a Natural Experiment on GitHub,” Jun. 2020, doi: 10.1109/ICSE43902.2021.00058.
- [47] D. Kessing, H. Hoang, C. Benninghaus, and M. Löwer, *Entwicklung einer Gamification-Strategie zur Motivationssteigerung von Mitarbeitern in der Produktion während der digitalen Transformation*. 2020.
- [48] A. Pathak, “¿Qué es GitLab y dónde alojarlo?,” Feb. 25, 2021. <https://geekflare.com/es/gitlab-hosting/> (accessed Sep. 11, 2021).

Bibliografía general

- Crespo González-Carvajal, Yania. Documentación de la asignatura de “Calidad, auditoría y seguridad de procesos, servicios, recursos y productos software”, Máster en Ingeniería Informática. Universidad de Valladolid. Curso 2019-2020.
- Documentation for GitLab Community Edition, GitLab Enterprise Edition, Omnibus GitLab, and GitLab Runner <https://docs.gitlab.com/> (accessed Sep. 11, 2021)
- Quick reference guides. Documentation – Git. <https://git-scm.com/doc> (accessed Sep. 11, 2021)
- SonarQube Documentation. SonarQube Docs <https://docs.sonarqube.org/latest/> (accessed Sep. 11, 2021)
- System Hooks. GitLab Docs https://docs.gitlab.com/ee/system_hooks/system_hooks.html(accessed Sep. 11, 2021)
- Project badges API. GitLab Docs. https://docs.gitlab.com/ee/administration/file_hooks.html (accessed Sep. 11, 2021)

8 Apéndices

8.1 Apéndice 1 - Guía de instalación, configuración y uso de GIT

Resumen:

Desde un repositorio local se puede crear una versión nueva de un código fuente. A continuación, se detalla cómo realizarlo, paso a paso, con la ayuda del sistema de control de versiones distribuido GIT.

Requisitos previos:

- ✓ Se ha utilizado una máquina virtual con la versión de Ubuntu Ubuntu 20.04.3 LTS.
- ✓ Se dispone de una cuenta de usuario en una instancia de Gitlab con un proyecto creado previamente.

Guía de instalación:

Instalación y configuración de Git en la máquina virtual:

En primer lugar, se realiza una actualización de los paquetes del sistema:

```
sudo apt update
```

Para instalar la aplicación:

```
sudo apt-get install git
```

Para comprobar la correcta instalación y la versión instalada.

```
git --version  
git version 2.25.1
```

Para este trabajo se ha utilizado la versión: 2.25.1

Otras versiones:

Para Windows. Dispone de un instalador desde el sitio oficial²⁷.

Para sistemas Centos:

```
sudo yum install git
```

Configuración.

En este paso se vincula la cuenta de Gitlab con la instalación de Git.

Se añade la información de la cuenta:

```
git config --global user.name "Nombre usuario"  
git config --global user.email usuario@correo.com
```

Uso de Git

Se dispone del proyecto "app1" en nuestra instancia de Gitlab. A continuación, desde la máquina local, situado en la carpeta del proyecto, se inicia el repositorio. Inicializado repositorio Git vacío en /home/usuario/misproy/app1/.git/

```
git init
```

Después se relaciona el proyecto local con el que se encuentra en Gitlab. Se les asigna el mismo nombre:

```
git remote add origin git@gitlab.com:NOMBREUSUARIO/app1
```

Se agregan los cambios. En este caso se utiliza "." para asignar toda la carpeta por completo:

```
git add .
```

Se realiza el commit para confirmar los cambios:

```
git commit -m "nueva versión v4"
```

Finalmente, se realiza el push para subir los cambios a Gitlab:

```
git push -u origin master
```

²⁷ Git Downloads. Official Site. <http://git-scm.com/downloads>

```
remote: Resolving deltas: 100% (94/94), done.  
remote: To create a merge request for master, visit:  
remote:                                     https://gitlab.com/victor.master.uva/app1/-  
remote:                                     /merge_requests/new?merge_request%5Bsource_branch%5D=master  
remote: To gitlab.com:victor.master.uva/app1  
* [new branch]      master -> master
```

Rama 'master' configurada para hacer seguimiento a la rama remota 'master' de 'origin'.

En la nuestra instancia de Gitlab el proyecto aparecerá una solicitud de fusión a la rama máster del proyecto, de la siguiente manera:

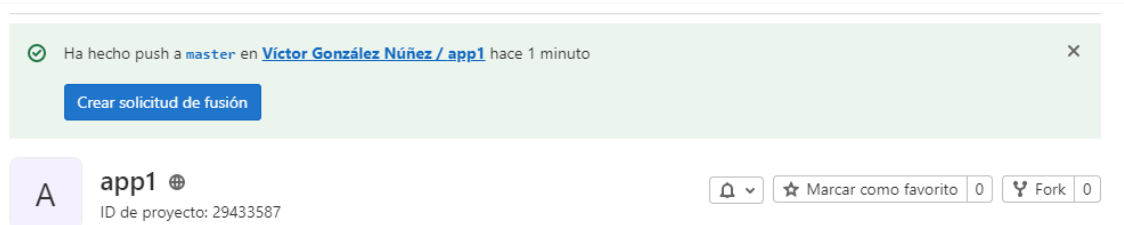


Ilustración 24 - Vista proyecto GitLab - Solicitud de fusión

Se acepta la fusión con la opción "Create merge request". La rama ya está actualizada, con la nueva versión de nuestro código:

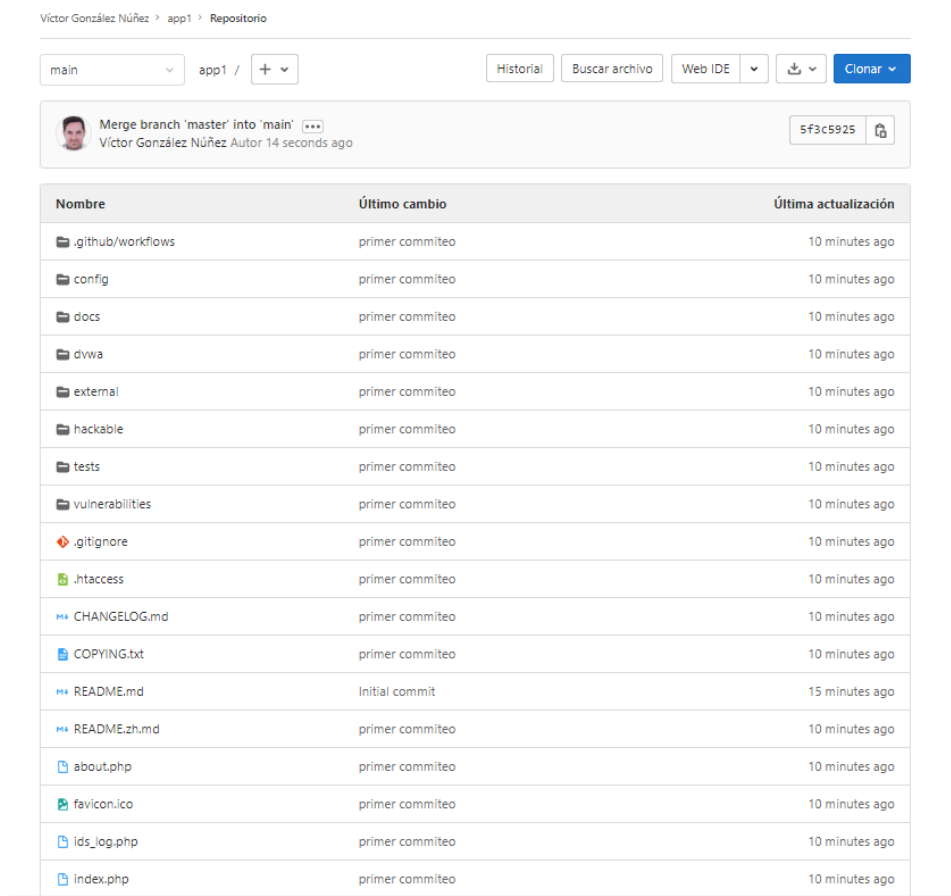


Ilustración 25 - Vista del proyecto en GitLab. Rama actualizada.

Problemas que surgieron en el proceso y solución:

Problema de comunicación entre el repositorio local y Gitlab.

Se tiene que realizar el emparejamiento con las claves para la comunicación SSH.

Para ello se crea la *key pair* para que las máquinas se comuniquen:

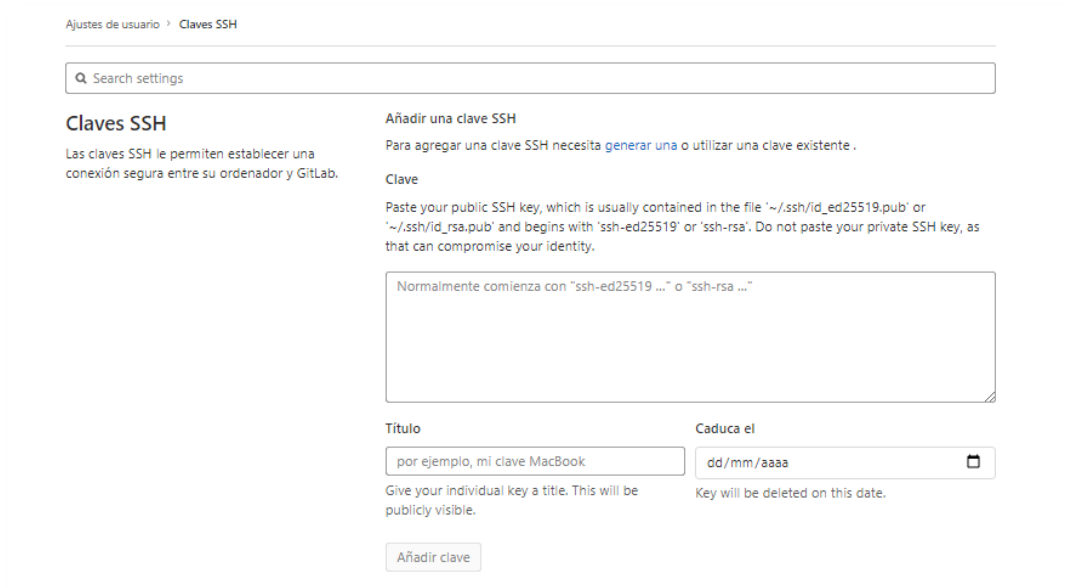
```
ssh-keygen -t ed25519 -C "keyapp"  
Generating public/private ed25519 key pair.  
Enter file in which to save the key (/home/user/.ssh/keyapp)
```

La *key* se crea en el directorio `/home/usuario/.ssh/`. En ese directorio aparece un `.pub` con la *key*.

Ahora se debe de agregar la *key* SSH pública a la cuenta de Gitlab y se mantiene la clave Privada. Esta *key* se lleva a Gitlab. Dentro de las opciones de configuración en Ajustes de usuario >> Claves SSH, se añade la clave.

Para ello se puede hacer un “cat” del `.pub`

```
cat c.lave.pub
```



The screenshot shows the 'Claves SSH' (SSH Keys) section of the GitLab user settings. At the top, there is a search bar for settings. Below it, the 'Claves SSH' section is titled, with a sub-header 'Añadir una clave SSH' (Add an SSH key). A note states: 'Para agregar una clave SSH necesita generar una o utilizar una clave existente.' (To add an SSH key, you need to generate one or use an existing one). The 'Clave' (Key) field is a large text area with a placeholder: 'Normalmente comienza con "ssh-ed25519 ..." o "ssh-rsa ..."'. Below the key field are two input fields: 'Título' (Title) with the example 'por ejemplo, mi clave MacBook' and 'Caduca el' (Expires) with the example 'dd/mm/aaaa'. A button 'Añadir clave' (Add key) is at the bottom.

Ilustración 26 - Vista edición de Claves SSH en GitLab

Información de ampliación:

Puede ampliar la información sobre esta configuración en la documentación oficial de GitLab²⁸.

²⁸ GitLab Docs <https://docs.gitlab.com/ee/ssh/index.html>

8.2 Apéndice 2 – Guía de instalación y configuración de GITLAB

Resumen:

GitLab ofrece una versión web desde *gitlab.com*. Alternativamente ofrece la posibilidad de descargar, instalar y mantener una instancia de GitLab para un entorno local. De esta manera se consigue una mejor personalización. A continuación, se detalla cómo realizar, paso a paso, su configuración.

Requisitos previos:

- ✓ Se ha utilizado una máquina virtual con la versión de Ubuntu 20.04.3 LTS.

Guía de instalación:

En primer lugar, se configuran los requisitos previos. Dependencias.

```
sudo apt-get update
sudo apt-get install -y curl openssh-server ca-certificates tzdata perl
```

Se agrega el repositorio:

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh
| sudo bash
```

Se instala con:

```
sudo apt install gitlab-ce
```

La versión con la que se ha trabajado es la siguiente: Gitlab versión 14.1.3 de agosto 2021.

Para comprobar que todo está correcto y revisar la versión instalada, se ejecuta el siguiente comando:

```
sudo gitlab-rake gitlab:env:info
```

Ampliación:

Para utilizar el sistema de envío de correos electrónicos para las notificaciones se puede instalar el servidor de correo Postfix:

```
sudo apt-get install -y postfix
```

Si se quiere modificar la configuración, el fichero se encuentra en `/etc/gitlab/gitlab.rb`

Se edita:

```
sudo vi /etc/gitlab/gitlab.rb
```

Por ejemplo, se puede añadir una URL externa para el acceso:

```
external_url 'https://example.co'
```

Después de guardar los cambios, se ejecuta el siguiente comando para reconfigurarlo:

```
sudo gitlab-ctl reconfigure
```

Problema con el usuario de acceso.

Reseteo de contraseña de usuario *root*.

Al realizar la instalación de la entidad GitLab, se puede producir un problema que no permite acceder a la ventana de establecimiento de contraseña de *root*. En ese caso no permite el acceso ni la recuperación del acceso.

Para resetear la contraseña se utiliza la consola de Ruby on Rails.

```
gitlab-rails console -e production
```

Dentro de la consola se busca el usuario:

```
user = User.where(id: 1).first
```

Una vez seleccionado, se cambia el *password*:

```
user.password = 'secret_pass'
user.password_confirmation = 'secret_pass'
```

Finalmente, se guardan los cambios.

```
user.save!
```

Aparece el siguiente mensaje de confirmación:

```
Enqueued ActionMailer::MailDeliveryJob (Job ID: b963f84f-660f-455d-a973-9213b1d72370) to Sidekiq(mailers) with arguments: "DeviseMailer", "password_change", "deliver_now", {:args=>[#<GlobalID:0x00007f62d1fdbae8 @uri=#<URI::GID gid://gitLab/User/1>>]}  
=> true
```

Soluciones de alojamiento de GitLab:

En este trabajo se han realizado pruebas en una máquina virtual. En el mercado existen soluciones de alojamiento especializadas [48] en entornos GitLab Hostings recomendados:

- A2 Hosting²⁹
- StackHero³⁰
- Gitlabhost³¹

²⁹ A2 Hosting <https://www.a2hosting.com/gitlab-hosting?aid=4a3e2f6e>

³⁰ StackHero <https://www.stackhero.io/en/services/gitlab>

³¹ Gitlabhost <https://gitlabhost.com/>

8.3 Apéndice 3 – Guía de instalación de SonarQube

Resumen:

Desde un repositorio local existe la posibilidad de realizar un análisis estático de código con SonarQube. En esta ocasión, se plantea la opción de utilizar una imagen Docker de SonarQube. A continuación, se detalla cómo realizarlo, paso a paso.

Requisitos previos:

- ✓ Se ha utilizado una máquina virtual con la versión de Ubuntu Ubuntu 20.04.3 LTS.
- ✓ La imagen de Docker de Sonar de su página oficial³².
- ✓ La instalación de SonarQube es la versión 9.0.1.46 CE.

Guía de instalación:

En primer lugar, se debe de instalar Docker en el servidor, para posteriormente hacer uso de la imagen de SonarQube.

Instalación de Docker

En primer lugar, se instalan los requisitos y dependencias.

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release
```

A continuación de instala la *key* y se agrega el repositorio de Docker.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Se añade el repositorio:

```
echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Una vez realizada esta instalación previa, se procede a instalar los paquetes de Docker.

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Se añade el usuario al grupo de Docker:

```
sudo usermod -aG docker nombre_usuario
```

Se prueba Docker haciendo un *run* de “hello-world”:

```
sudo docker run hello-world
```

En la documentación oficial de Docker³³ se pueden encontrar otras opciones de configuración.

Imagen de Docker de SonarQube. Se descarga la imagen de Docker de SonarQube.

```
sudo docker pull sonarqube
```

Se inicia el contenedor de Sonar para acceder por el puerto 9000.

```
docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube
```

Para comprobar el funcionamiento correcto y la versión de SonarQube:

```
sonar-scanner -v
INFO: Scanner configuration file: /opt/sonar-scanner/conf/sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarQube Scanner 4.2.0.1873
INFO: Java 11.0.3 AdoptOpenJDK (64-bit)
INFO: Linux 5.11.0-27-generic amd64
```

Se accede a la instancia con:

```
localhost:9000
```

³² Docker SonarQube image: https://hub.docker.com/_/sonarqube/

³³ Install Docker Engine on Ubuntu. Docs Docker <https://docs.docker.com/engine/install/ubuntu/>

Y la siguiente información de acceso:

- Usuario: admin
- Pass: admin

En el primer acceso se solicita el cambio de *password*.

Además, SonarQube dispone de un fichero de propiedades donde se encuentran las opciones de configuración:

El fichero es **sonar-scanner.properties** se encuentra en:

`/opt/sonar-scanner/conf/sonar-scanner.properties`

En este fichero se puede editar, por ejemplo, la url de acceso:

```
sonar.host.url=http://localhost:9000
```

También es relevante el fichero **sonar-project.properties**, que se suele situar en el raíz.

Problemas y soluciones:

Problemas con la imagen de SonarQube:

```
Error response from daemon: Conflict. The container name "/sonarqube" is already in use by container "58f6cbd59652a6ca8eb1a8de9346bec792f95c5a832d824a3b97d2b8698b09a3".
```

Para solucionarlo se puede revisar el estado de los contenedores que están corriendo en Docker:

```
docker ps -a
```

Se puede forzar la eliminación del contenedor que está corriendo.

```
docker rm [OPTIONS] contenedor
```

8.4 Apéndice 4 – Guía de instalación y configuración de SonarQube Runner

Resumen:

Para realizar análisis de código estático con SonarQube se pueden utilizar herramientas como Ant o Maven. Una alternativa es utilizar SonarQube Runner. A continuación, se detalla cómo realizar su configuración, paso a paso.

Requisitos previos:

- ✓ Se ha utilizado una máquina virtual con la versión de Ubuntu Ubuntu 20.04.3 LTS.
- ✓ SonarQube versión 3.7 o superior.
- ✓ Java 6 o superior.

Guía de instalación:

En primer lugar, se descarga el paquete de Sonar Runner en el directorio deseado:

La instalación tiene un directorio /conf. En ese directorio está el fichero de configuración de SonarQube Runner **sonar-runner.properties**.

Se edita el fichero de configuración:

```
#----- Default SonarQube server
sonar.host.url=http://localhost:9000

#----- MySQL
sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8
sonar.jdbc.driver=com.mysql.jdbc.Driver

#----- Global database settings
#sonar.jdbc.username=admin
#sonar.jdbc.password=vic12345
```

Esta configuración se puede hacer para otros sistemas de gestión de base de datos:

```
#----- Oracle
#sonar.jdbc.url=jdbc:oracle:thin:@localhost/XE
#----- Microsoft SQLServer
#sonar.jdbc.url=jdbc:jtds:sqlserver://localhost/sonar;SelectMethod=Cursor
#----- PostgreSQL
#sonar.jdbc.url=jdbc:postgresql://localhost/sonar
```

A continuación, se crea una variable de entorno:

```
SONAR_RUNNER_HOME
```

Se añade el directorio al path añadiendo SONAR_RUNNER_HOME/bin.

```
export SONAR_RUNNER_HOME=/opt/sonar-runner-v2.4
export PATH=$PATH:${SONAR_RUNNER_HOME}/bin
```

Después, se realiza la configuración de fichero de propiedades del runner.

Se edita en: SONAR_RUNNER_HOME/conf/sonar-runner.properties

Para comprobar su correcta instalación se ejecuta:

```
Sonar-runner -h
```

Para analizar un proyecto, se crear un fichero de sonar-runner properties dentro del directorio.

```
sudo vi sonar-DVWA-master.properties
```

Se añade la información del proyecto:

```
# must be unique in a given SonarQube instance
```

```
sonar.projectKey=my:project
# --- optional properties ---

# defaults to project key
#sonar.projectName=My project
# defaults to 'not provided'
#sonar.projectVersion=1.0

# Path is relative to the sonar-project.properties file. Defaults to .
#sonar.sources=.

# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
```

Ampliación de información:

SonarQube ofrece varios ejemplos en su repositorio de sonar scanning examples³⁴, según el tipo de análisis que se quiera realizar. En la documentación oficial de SonarQube³⁵ se puede encontrar información sobre la configuración de SonarQube Runner.

³⁴ SonarSource Continuous Code Quality github.com/SonarSource/sonar-scanning-examples

³⁵ SonarQube Runner. Installing and Configuring
<https://docs.sonarqube.org/display/SONARQUBE45/Installing+and+Configuring+SonarQube+Runner>

8.5 Apéndice 5 - Características de los servidores

HPE ProLiant MicroServer Gen10 Plus

Procesador

Intel Xeon E-2200 Series / 9th Gen Pentium G						
Model	CPU Frequency	Cores	L3 Cache	Power	DDR4	SGX
Xeon E-2224	3.4 GHz	4	8 MB	71W	2666 MT/s	No
Pentium G5420	3.8 GHz	2	4 MB	54W	2400 MT/s	No

Tipo de memoria Memoria estándar HPE DDR4

Tipo de fuente de alimentación Adaptador de alimentación externo de 180 W

Ranuras de expansión 1 ranura PCIe Gen3 x16

Controlador de red Controlador Ethernet 4 puertos por controlador

Controlador de almacenamiento 1 HPE Smart Array S100i

Formato MicroServer

Peso (imperial) 9.33 lb

Peso (métrico) 4,23 kg

Dimensiones del producto (imperial) (H x B x T) 4.58 x 9.65 x 9.65 in

Dimensiones del producto (métrico) (H x B x T) 11,6 x 24,5 x 24,5 cm

Descripción detallada del producto Servidor HPE ProLiant MicroServer Gen10 Plus con un procesador Intel Xeon E-2224, 16 GB de memoria, un HDD de 1 TB SATA, cuatro bahías para unidades de factor de forma grande sin conexión *hot-plug* y una fuente de alimentación externa de 180 W

Más información sobre las características del producto en su página oficial ³⁶

³⁶ Servidor HPE ProLiant MicroServer Gen10 Plus. HP Enterprise Online Shop. <https://bit.ly/3twWC3b>

Servidor HPE ProLiant DL380 Gen10
--

Tipo de procesador Intel

Familia del procesador Procesador escalable Intel® Xeon® de la serie 8100/8200, procesador escalable Intel® Xeon® de la serie 6100/6200, procesador escalable Intel® Xeon® de la serie 5100/5200, procesador escalable Intel® Xeon® de la serie 4100/4200, procesador escalable Intel® Xeon® de la serie 3100/3200

Núcleo de procesador disponible De 4 a 28 núcleos, según el modelo

Caché de procesador De 8,25 a 38,50 MB L3, según el modelo del procesador

Memoria máxima 3,0 TB con DDR4 de 128 GB, en función del modelo de procesador; 6,0 TB con kit de memoria persistente HPE 2666 de 512 GB, en función del modelo de procesador

Ranuras de memoria 24 ranuras DIMM

Tipo de memoria DDR4 HPE Smart Memory con memoria persistente Intel Optane opcional para HPE, según el modelo de procesador seleccionado.

Tipo de NVDIMM HPE NVDIMM-N *Disponible solo en procesadores escalables Intel® Xeon® de 1.ª generación

Rango de NVDIMM Rango único

Capacidad de NVDIMM 16 GB

Controlador de red Adaptador Ethernet HPE 331i de 1 Gb y 4 puertos por controladora o HPE FlexibleLOM opcional, según el modelo

Características de los ventiladores del sistema Ventiladores redundantes *hot-plug*, estándar

Ranuras de expansión 8, para descripciones detalladas consulte las especificaciones rápidas

Controlador de almacenamiento 1 HPE Smart Array S100i y/o 1 HPE Smart Array P408i-a y/o 1 HPE Smart Array P816i-a y/o 1 HPE Smart Array E208i-a, según el modelo

Nombre del procesador Intel

Número del procesador 1 ó 2

Velocidad del procesador 3,9 GHz, máximo según el procesador

Memoria, estándar LRDIMM de 3,0 TB (24 X 128 GB); Memoria persistente HPE de 6,0 TB (12 X 512 GB)

Seguridad Kit de bisel con cierre opcional, kit de detección de intrusión y HPE TPM 2.0

Formato 2 U

Peso (imperial) 32.6 lb

Peso (métrico) 14,76 kg

Dimensiones del producto (imperial) (H x B x T) 17,54 x 28,75 x 3,44 pulg.

Dimensiones del producto (métrico) (H x B x T) 44,55 x 73,03 x 8,74 cm

Más información sobre las características del producto en su página oficial³⁷

³⁷ Servidor HPE ProLiant DL380 Gen10. HP Enterprise Online Shop <https://bit.ly/3E1MN1Q>

