Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Informática Aplicada

DevOps Management Dashboard

TRABAJO RECEPCIONAL para obtener el **GRADO** de
MAESTRO EN INFORMÁTICA APLICADA

Presenta: **GUILLERMO RUIZ VILLELA**

Tutor **LUIS FERNANDO GUTIÉRREZ PRECIADO**

Tlaquepaque, Jalisco. Diciembre de 2021.

AGRADECIMIENTOS

Agradezco a todos mis profesores de la Maestría en Informática Aplicada, aprendí mucho de ellos y me ayudaron a complementar mis conocimientos para enriquecer este proyecto, a mi asesor de este proyecto, Luis Fernando Gutiérrez Preciado, a mi coordinador de la maestría Ricardo Salas Mejía y a mi familia que siempre estuvo a mi lado para lograr este objetivo.

DEDICATORIA

Dedico esta tesis a mi amada familia, comenzando por mi amada esposa Adriana Gazca Chavarría, a mi adorada hija, Elizabeth Ruiz Gazca y a mis padres, Luis Octavio Ruiz Urbicio y Adelina Villela Valdez, siempre han estado a mi lado, son mi soporte y mi mayor inspiración en la vida.

RESUMEN

DevOps puede ser definido como un conjunto de herramientas y prácticas enfocadas en la colaboración, integración, automatización y comunicación para el desarrollo de software y las operaciones de TI en una empresa; ayuda a las organizaciones a ser altamente eficientes y productivas, eficientiza los flujos de trabajo y mejora la colaboración entre los diferentes equipos de TI.

A pesar de los grandes beneficios que DevOps puede entregar, las empresas se enfrentan a diferentes retos en la implementación de DevOps. La complejidad de la implementación, seguido de la falta de herramientas y soluciones estandarizadas han impedido un mayor crecimiento de este mercado.

*En este trabajo se propone que mediante un **Devops Management Dashboard**, las empresas podrán centralizar toda la aplicación de la metodología, conectado con una gran cantidad de herramientas de desarrollo, calidad, agile, monitoreo, análisis de código, defectos, etc., controlar y monitorear el ciclo de vida completo, desde que se registra una nueva funcionalidad, hasta que es liberada en ambientes productivos, además de consideran los diferentes roles involucrados, desde Desarrolladores de Software, Equipos Scrum, Equipos de Operaciones-Infraestructura, Soporte a Producción, Project Managers, Gerentes, etc.*

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ACRÓNIMOS, ANGLICISMOS Y ABREVIATURAS

DevOps	Development Operations
TI	Tecnologías de Información
SLA	Acuerdo de Nivel de Servicio *(Service Level Agreement)*
WWW	World Wide Web
KPI	Indicadores Clave de Rendimiento *(Key Performance Indicator)*
CI	Integración Continua (Continuous Integration)
CD	Entrega Continua (Continuous Delivery)
SQL	Structured Query Language
HTTP	Hyper Text Transfer Protocol
AWS	Amazon Web Services
JPA	Java Persistency API
REST	*Representational State Transfer*
SOAP	Simple Object Access Protocol
KPI	Key Performance Indicator
DOMD	DevOps Management Dashboard

1. INTRODUCCIÓN

En este capítulo, se describe el contexto y problemas que existen en la gestión de proyectos que implementan DevOps. También se describen los objetivos, entre los que se encuentran, eficientizar el proceso de desarrollo de software, la integración y comunicación con el equipo de Infraestructura/Operaciones. Además, se describe el objetivo buscado en el presente documento, que consiste en la implementación de una plataforma web, llamada ***DevOps Management Dashboard (DOMD)***, el cual tiene como objetivo centralizar, transparentar, planear, monitorear y controlar todas las actividades relacionadas con la implementación y el despliegue productivo de software en ambientes altamente productivos.

1.1. Antecedentes

El desarrollo de software a nivel empresarial siempre representa un reto, ya sea siguiendo una metodología formal, con modelo tradicional *waterfall* o metodologías ágiles, con equipos de trabajo multidisciplinarios, distribuidos en diferentes ubicaciones, considerando incluso diferentes países y husos horarios, entre mayor sea el número de integrantes que componen el equipo, ese reto se vuelve aún más grande.

Existen un sin número de razones por la cual los proyectos de software fallan o tienen retraso en sus tiempos de entrega, desde la falta de capacidades de los equipos de desarrollo de software, una mala definición de requerimientos por parte del cliente, mal análisis y diseño del proyecto, mala elección de tecnologías, una mala administración del proyecto, entre muchas otras razones.

Para este proyecto, se considera solamente aquellas empresas que tengan implementado, o que deseen implementar, un modelo de desarrollo de software orientado a *DevOps*.

Con base a mi conocimiento y experiencia profesional, se puede definir a *DevOps*, como un conjunto de herramientas y prácticas enfocadas en la colaboración, integración,

automatización y comunicación para el desarrollo de software y las operaciones de TI en una empresa; ayuda a las organizaciones a ser altamente eficientes y productivas, eficientiza los flujos de trabajo y mejora la colaboración entre los diferentes equipos de TI.

Dentro de DevOps, puede existir el rol del *Release Manager*, el cual desempeña una función muy importante de cohesión entre todos los equipos, es responsable de los entregables de cada iteración, de garantizar la entrega en tiempo y forma de los componentes de software, trabaja muy de cerca con todos los equipos de Desarrollo de Software y es una interfaz con los equipos de Pruebas, Soporte a Producción, Infraestructura, Gestión de los Programas y el cliente en caso de ser necesario, por lo tanto, la administración y gestión del ***DOMD*** es una actividad asociada a sus funciones con el objetivo de tener mayor control y eficiencia en todos los procesos previamente descritos.

1.2. Justificación

El propósito de este proyecto que denominaremos el ***DOMD***, es maximizar la posibilidad de éxito al centralizar y eficientizar la gestión de un proyecto de software que implemente metodologías ágiles y DevOps.

1.3. Problema

En cualquier organización que se dedique al desarrollo de software, aplicando metodologías agiles y *DevOps* y que no cuente con una plataforma centralizada, en este caso ***DOMD***, la administración, el seguimiento de todas las tareas, dependencias, controles y monitoreo a cada iteración, la interacción con los diferentes equipos, se vuelve una actividad muy complicada, con una gran posibilidad de errores, que desencadenan en lo siguiente:

- Retrasos en los tiempos de entrega
- Mala ejecución de Integración Continua, Despliegue Continuo y Entrega Continua
- Mala comunicación entre todos los equipos
- Mala gestión de los diferentes versiones o ramas del código fuente.
- Mala administración de los defectos
- Sanciones a la empresa por no cumplir los niveles de servicio esperados (SLA)

1.4. Objetivos

1.4.1. Objetivo General:

Diseñar el ***DOMD*** el cual tendrá las siguientes características:

Esta herramienta está orientada al Gerente de Liberación *(Release Manager)* principalmente, el cual cumple una función fundamental de enlace y comunicación entre los equipos de Desarrollo de Software y los demás equipos, Soporte a Producción, Infraestructura, Pruebas, Gestión de los Programas, pero su función no está acotada solamente a él, ya que será de utilidad para todos los equipos mencionados anteriormente.

Análisis y Diseño de una plataforma web mediante la cual se gestionen todos los recursos y eventos relacionados con las prácticas de *DevOps*.

Definición de los roles sensitivos que deban de tener acceso a la plataforma web para opciones de consulta o actualización de la plataforma.

Centralizar el acceso de los datos de las diferentes aplicaciones y procesos involucrados en los desarrollos de software de cada iteración, entre las que se consideran: fechas y eventos importantes (Inicio de Iteración, Días Festivos, etc.), estado de construcciones de código (*Build*), Entregables, Lista de defectos, Cambios o mezclas (*merge)* pendientes en el código, Cambios o mezclas (*merge)* pendientes en la base de datos, Lista de contactos de soporte, Scripts para ejecutar, Sesiones de entrenamiento a equipos de Soporte a Producción, Análisis de código estático, Defectos de Automatización.

Para una correcta gestión del proyecto, aplicando metodologías ágiles y *DevOps,* es fundamental que se consideren los siguientes puntos:

1. Administración de un cronograma o calendario donde se pueda editar la información con los eventos sensitivos o importantes de cada iteración, como inició de la iteración, fin de la iteración, congelamiento de código (del inglés, *code freeze*), inicio de pruebas internas, inicio de pruebas con el cliente, liberación de ambientes, liberación a producción.

2. Integración con la herramienta de gestión del proyecto, para la visualización de tareas, status, actualizaciones por cada iteración, ejemplo, integración con *Rally* o *Jira*.

3. Módulo de administración de defectos para la aprobación e integración con Despliegue Continuo.

4. Módulo de monitoreo de defectos de automatización, ejemplo, integración con *Quality Center*.

5. Módulo de monitoreo de creación diaria de paquetes de software.

6. Módulo de análisis estático de código, ejemplo, integración con *Sonar.*

7. Módulo de monitoreo de mezclas de código, de cambios de código y de base de datos.

8. Módulo del equipo de soporte a liberaciones en Producción con la información de contacto.

9. Dentro de los perfiles que se tienen identificados que podrán realizar actividades dentro del *Dashboard*, se encuentran los siguientes:

- Gerente de Liberación (*Release Manager*)
- Gerente de Desarrollo de Software
- *Scrum Master* o Líder de Equipo
- Desarrollador de Software
- Líder de pruebas
- *Tester*
- Manager de Programas *(Program Manager)*
- Gerente de Operaciones
- Invitado, solo puede acceder a ver la información del *Dashboard*, sin realizar ninguna acción.

10. Todos los módulos deben de tener la posibilidad de enviar notificaciones por correo electrónico de manera periódica o al momento.

1.4.2. Objetivos Específicos:

Dentro de los objetivos específicos del ***DOMD*** se considera la propuesta de la Arquitectura de Software, donde se especificará el Modelo General de la Aplicación, que tiene como

objetivo explicar claramente los actores y componentes que conforman dicha aplicación y a partir de ahí se esquematizaran diagramas partiendo de lo más general a aspectos más particulares de la Arquitectura propuesta; para cada una de las diferentes capas de la Arquitectura, se explicará de manera detallada los componentes e interacciones entre ellos, así como los roles involucrados.

Otro de los objetivos específicos es definir y explicar la Base de Datos Relacional que servirá como soporte medular a la aplicación para poder gestionar distintos aspectos, como los usuarios, sus roles, fechas del proyecto, inicios de iteraciones, mensajes, entre otros aspectos trascendentes de la aplicación.

Por último, se presentará el resultado con un prototipo funcional y que plasme las funcionalidades más relevantes del DOMD, con lo cual se pueda resaltar su utilidad y diferenciación con cualquier otro producto en el mercado.

2. ESTADO DEL ARTE O DE LA TÉCNICA

Después de realizar una extensa investigación en diferentes medios académicos, como la Biblioteca Online del ITESO, *EBSCO*, *Google Academics*, *Base Academic Search Engine*, *Dialnet*, *Scientific Electronic Library Online*, etc., no existe como tal un producto con el mismo alcance o visión al planteado en este estudio al que llamamos el ***Devops Management Dashboard (DOMD)***, lo que si existe es información teórica acerca de *DevOps*, fundamentos, objetivos, pilares, principios, etc., los cuales se detallaran más adelante en el Marco Teórico.

Debido a lo anterior se procedió a realizar una investigación en la *www*, acerca de aplicaciones existentes en el mercado que tengan algún tipo de relación con la propuesta actual, se encontraron las siguientes aplicaciones y se enlistan sus principales características.

DevOps Dashboards [8]

Entre sus características tenemos: Definir tipos de proyectos, Monitoreo de fases del proyecto con retraso, Estadística de tiempos activos e inactivos, Estadística de tiempos de carga, Administración del ciclo de vida de Defectos y Resoluciones *(Quality Assurance)*,

DevOps Dashboard and View [7]

Entre sus características tenemos: Monitoreo de actividades, Monitoreo de *KPIs* y métricas, Monitoreo de rendimiento usando "*Performance Analytics*" Acceso a detalle por líneas de trabajo, aplicaciones, etapas o equipos.

Azure DevOps Services [12]

Entre sus características tenemos: Customizable a través de *widgets,* Monitoreo de progreso y tendencias, Integración con productos de *Microsoft*.

Datadog [4]

Entre sus características tenemos: Integración de *Azure DevOps Builds*, Seguimiento de liberaciones *(Releases)*, Seguimiento de Items de trabajo *(Work Items)*, Seguimiento de

eventos de código, Monitoreo en tiempo real para resolución de problemas, Monitoreo de Flujos de Trabajo/Automatización *(Pipelines).*

De las 4 herramientas previas, es muy complicado realizar un análisis más exhaustivo acerca de sus ventajas y desventajas, ya que se requieren licencias con el costo asociado para poder ejecutarlas, solo se hace énfasis en las características que se describen en sus sitios principales y de primera vista podemos observar que ninguna de ellas tiene el alcance que si contempla el ***DOMD.***

Dentro de las características fundamentales del ***DOMD*** y que no existe como tal una herramienta o plataforma que las englobe en su totalidad, se encuentran las siguientes:

1) Lista de Tareas de Nuevas Funcionalidades
2) Defectos de automatización de pruebas *(Automation Defects)*
3) Transferencia de conocimiento, Scripts, Pruebas de aceptación, Jobs en ejecución, documentación, etc. *(Production Readiness)*
4) Lista de Defectos *(Defects List)*
5) Resolución a los Defectos *(Hot Fixes)*
6) Integración y mezcla de cambios de código *(Pending Code Merge)*
7) Integración y mezcla de cambios de base de datos *(Pending Database Merge)*
8) Herramientas de análisis de código estático *(Code Static Analyzer)*
9) Estatus de Construcción y Despliegue *(Build & Deployment Status)*
10) Cronograma y calendario de eventos *(Timeline)*

3. MARCO TEÓRICO/CONCEPTUAL

En este capítulo tiene el objetivo de presentar los conceptos más importantes relacionados a *DevOps*; iniciando con algo de historia que nos muestre de donde surge este concepto y como se relaciona con las metodologías agiles. Posteriormente se describe cómo ha evolucionado y finalmente se detallarán conceptos como integración y entrega continua. De forma transversal se presentan los fundamentos, objetivos, pilares y principios de los DevOps.

3.1. Historia

En el año 2007, Patrick Debois, Administrador de Proyectos y practicante de *Agile*, tuvo una asignación con el gobierno de Bélgica para ayudar en la migración de centros de datos. Su rol era certificar y preparar las pruebas, sus responsabilidades requerían manejar las relaciones entre los equipos de Desarrollo de Software y los equipos de Operaciones (servidores, bases de datos, redes) [11].
Su experiencia y frustraciones acerca de las diferencias y la falta de cohesión entre los procedimientos de Desarrollo y de Operaciones, plantearon una semilla de descontento para Debois, sin saber que pronto alguien iba a lograr éxito en esta integración.

En agosto de 2008, en la conferencia de *Agile* en Toronto, Andrew Schafer publica una oferta para moderar una conferencia para discutir el tema "Infraestructura Agile", solo una persona se presentó, Patrick Debois, Discutieron y compartieron sus ideas con otros conceptos avanzados de "Administración de Sistemas Agiles", en el mismo año, Debois y Schafer formaron un grupo de Administración de Sistemas Agiles en *Google*, pero tuvo poco éxito [13].

En junio de 2009, en la conferencia de *O'Reilly Velocity*, 2 empleados de *Flicker*, John Allspaw, vicepresidente senior de operaciones técnicas y Paul Hammond, Director de Ingeniería, expusieron la presentación "*10+ Deploys per-Day: Dev and Ops Cooperation at Flickr*". [13]

Tanto Allspaw y Hammond, ejecutaron un rol de liderazgo y administración sumamente importante dentro de sus divisiones de desarrollo y operaciones, en lugar de culparse y señalarse acerca de quién era culpable o responsable de X o Y actividad, como ocurre en gran cantidad de empresas de software, lograron una sinergia, integración y transparencia entre ambas divisiones para poder trabajar en armonía, con orden y control.

Los ejes fundamentales de la ponencia de Allspaw y Hammond para una correcta integración y trabajo entre Desarrollo y Operaciones son las herramientas y la cultura, que se describen a continuación.

Herramientas

En este apartado, se consideran como herramientas, a las aplicaciones de software que se utilizan de manera cotidiana para que los diferentes equipos puedan desempeñar su trabajo de una manera más ordenada, profesional y eficiente, a continuación, se describen los diferentes tipos.

Infraestructura Automatizada

> Se refiere a herramientas que permitan automatizar operaciones o procesos a nivel sistema operativo o nivel aplicativo, que puedan ser configurables y se puedan parametrizar, calendarizar, etc.

Control de Versiones de Código Compartidas

> Se refiere a la centralización o uso de un repositorio único para los equipos de desarrollo y operaciones, simplificando acceso.

Construcción de un solo paso y despliegue *(One step build and deploy)*

> Se refiere a la posibilidad de compilar, localizar y desplegar los artefactos o paquetes de software, comúnmente asociado a Integración Continua y Entrega Continua.

Banderas de Funcionalidad *(Feature Flag)*

> *Aka "branching code"*, se refiere a la estrategia de manejo de *ramas (branches)*, mediante la cual solo importa la versión que se encuentra en Producción y las versiones anteriores ya no importan. Esto también considera la habilitación-inhabilitación de banderas dentro del código para mostrar o compartir funcionalidades de la aplicación.

Métricas Compartidas

Se refiere a compartir todos aquellos valores asociados a la aplicación, ya sea desde el punto de vista aplicativo, de sistema operativo, de procesos, de base de datos, etc.

IRC and IM robots

Se refiere a contar con herramientas de comunicación en tiempo real para la interacción de todos los miembros del equipo

Cultura

En el tema de cultura, se refiere a aquellos atributos, actitudes o comportamientos que deben de existir entre todos los miembros de los diferentes equipos y que contribuyan a un mejor ambiente laboral y organizacional, se enlistan los principales a continuación.

Respeto

Se refiere a tratar a los miembros del equipo con educación y respeto, todos juegan un papel importante y su labor es valiosa para todo el proceso.

Confianza

Se refiere a la confianza y transparencia que debe haber entre todos los miembros del equipo, saber comunicar cualquier situación o problema asociado al proyecto.

Actitud saludable acerca de los fallos *(Healthy attitude about failure)*

Se refiere a buscar soluciones expeditas, realizar análisis de causas de origen y lecciones aprendidas para evitar que vuelva a ocurrir

Evitar culpar

Se refiere a evitar buscar culpables, entender los procesos como tal y con base a las lecciones aprendidas, evitar que vuelvan a ocurrir.

Debois observó la presentación de Allspaw y Hammong por video conferencia. Estaba inspirado al punto de formar su propia conferencia llamada *"DevopsDays"* en Ghent, Bélgica, la cual abrió sus puertas el 30 de octubre con una impresionante asistencia de desarrolladores, administradores de sistemas, etc. Cuando la conferencia terminó, la discusión se movió a *Twitter*, para crear el memorable hashtag. Debois acortó el nombre a *#DevOps* y en este punto de la historia se puede decir que surge el término *"DevOps"*, que sería una contracción de *"Development"* y *"Operations"* [10].

3.2. Conceptos y Evolución

DevOps se ha vuelto un tema importante en la industria de software en los años recientes, hay cientos de artículos en internet, la mayoría de ellos son muy positivos, *Ambler* define *DevOps* como la colaboración efectiva entre los equipos de Desarrollo y Operaciones para la incorporación de sus procesos, *DeGrandiss* [2] sugiere que *DevOps* significa aplicar procesos agiles a las actividades de operación, *Jabbari* define *DevOps* como una metodología de desarrollo enfocada en cerrar la brecha entre Desarrollo y Operaciones, enfatizando la comunicación y colaboración, integración continua, medición de calidad y entrega con *despliegues* automatizados utilizando un conjunto de prácticas de desarrollo.

En general podemos acotar *DevOps* como la combinación de los equipos de Desarrollo y equipos de Operaciones, utilizando las mejores prácticas en colaboración y comunicación, con el propósito de cerrar la brecha de ambos equipos.

La metodología *Agile* [3] es actualmente una de las más populares para el desarrollo de software, es adoptada por una gran cantidad de compañías, y la parte de operaciones, relacionado con el despliegue y el mantenimiento, por lo general está asociada a empresas de subcontratación de servicios *(outsourcing)*, utilizar *Agile* en la parte de desarrollo de Software, sin considerar el despliegue y las operaciones puede causar graves problemas e incluso impedir que nuevas funcionalidades sean presentadas a los usuarios lo más rápido posible. Las compañías están considerando adoptar *DevOps* con Agile [1] para mejorar sus procesos y desarrollo de software.

A pesar de que *DevOps* es muy prometedor para eliminar la brecha entre los equipos de Desarrollo y Operaciones, pasar a un estado de sincronización y eficiencia en los procesos (automatizados y no automatizados), tareas y comunicación, no es trivial. A pesar de ello, debido a la naturaleza, tamaño, infraestructura y automatización de cada compañía, son diferentes los retos que enfrentan. Sin embargo, se puede obtener inspiración y aprendizaje de la experiencia de los predecesores y con el conocimiento actual, estar listos para manejar problemas, planear y prepararse por adelantado.

Muchas personas están preocupadas acerca de combinar *Agile* con *DevOps* y muchas compañías en sus procesos de desarrollo de software tienden a usar *Agile*, pero con el desarrollo continuo de la industria del software, en periodos de tiempo corto para liberar

productos al mercado, mientras que mantener la calidad, las expectativas del cliente y el uso de nuevas tecnologías se ha vuelto la norma. En realidad, Agile habilita entregar software de manera rápida, pero no puede ser *desplegada* y operada tan pronto como sea posible.
En los equipos Agile, hay principalmente 3 roles, *Product Owner*, *Scrum Master* y Equipo de Desarrollo. En un escenario común, el equipo de desarrollo es responsable de la codificación y de las pruebas y para poder enviar las nuevas funcionalidades o modificar parcialmente algunos para el cliente lo más pronto posible, se necesitan un equipo especial de operaciones.

"Los conceptos Continuous Integration, Continuous Delivery y Continuous Deployment son frecuentemente mal utilizados o intercambiados, (incluso también se confunde con el termino Continuous Release). La gente dice Continuous Integration cuando ellos quieren decir Continuous Deployment, o dicen Continuous Deployment cuando quiere decir Delivery y viceversa. Para hacerlo más complejo, algunas personas usan la palabra "DevOps" cuando quieren hablar acerca de los diferentes tipos de "Continuous". DevOps, por lo tanto, es más que solo Continuous Integration, Delivery y/o Deployment." [5]

Integración Continua (*Continuous Integration*)

El primer paso para entregar software consistente y de alta calidad es Integración Continua (CI), todo es acerca de asegurar que el software este en un estado de desplegable todo el tiempo, esto significa que el código compile y que la calidad del código pueda asumirse que sea razonablemente buena.

Entre sus componentes principales se encuentran:

- Control de Código Fuente

  Repositorio centralizado para manejo de código
- Servidor de Integración Continua

  Las construcciones de software (*builds)* [14] deben ser un proceso automatizado utilizando alguna herramienta de CI, algunas de las más comunes son *Jenkins*, *TFS*, *Cruise Control* y *Bamboo*.
- Calidad de Software

Se refiere a garantizar ciertos niveles de calidad en el software realizado, se pueden utilizar Pruebas Unitarias, Pruebas de Integración, los cuales se pueden clasificar en *Big Bang Testing* o Pruebas Incrementales, Pruebas de aceptación, *Smoke Tests* y Analizadores estáticos de código

- Automatización
  Se refiere al proceso de eliminar interacción humana con la programación de tareas o procesos y su calendarización o ejecución con determinados eventos.
- Trabajo en Equipo
  Todo esto involucra un esfuerzo colaborativo, generar una disciplina de trabajo, tener claros los procesos y seguirlos al pie de la letra para tener control y orden en el uso de los recursos.

Entrega Continua (*Continuous Delivery*)

El siguiente paso hacia un exitoso despliegue de software es Entrega Continua (el cual no tiene una abreviación y se puede confundir con Despliegue Continuo). Con Entrega Continua, los artefactos producidos por el Servidor de *CI* pueden ser desplegados en un servidor de Producción con el simple clic de un botón. Integración Continua es un prerrequisito para una exitoso Entrega Continua.

Un principio fundamental para las actividades de *Entrega Continua* es el *Deployment Pipeline o Flujo de Despliegue*, como se observa en la Figura 1, este modelo puede ser usado para mapear actividades manuales y automatizadas. El *Deployment Pipeline* es un flujo clave de valor para el proyecto, desde la integración de cambios de código hasta la liberación, describe los procesos, conexiones manuales y automatizados y las respuestas a los eventos de integración de cambios de código.

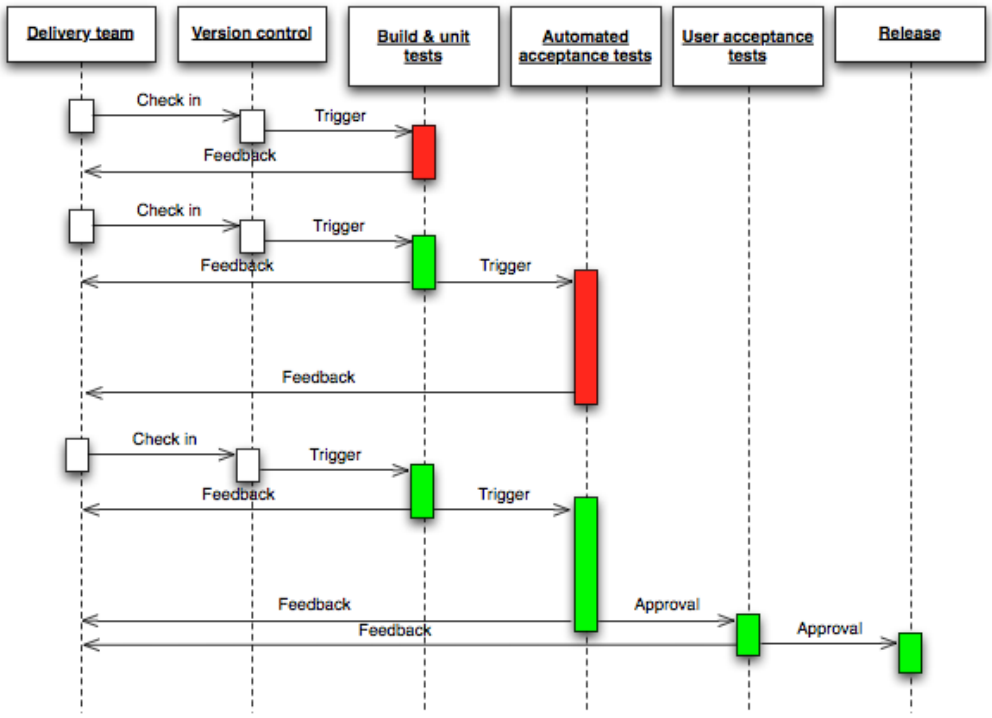

Figura 1 Flujo de Despliegue (Deployment Pipeline)

DevOps incluye cambios organizacionales, de infraestructura y de cultura en adición a *Despliegue Continuo***,** fuerte dependencia en el uso secuencial de herramientas de automatización a lo largo del ciclo de vida es uno de sus principales rasgos distinguibles [6]. Durante todo el proceso se consideran diferentes herramientas o programas de software que soportan el ciclo entero, la Figura 2, muestra la relación entre *DevOps* y Entrega Continua.

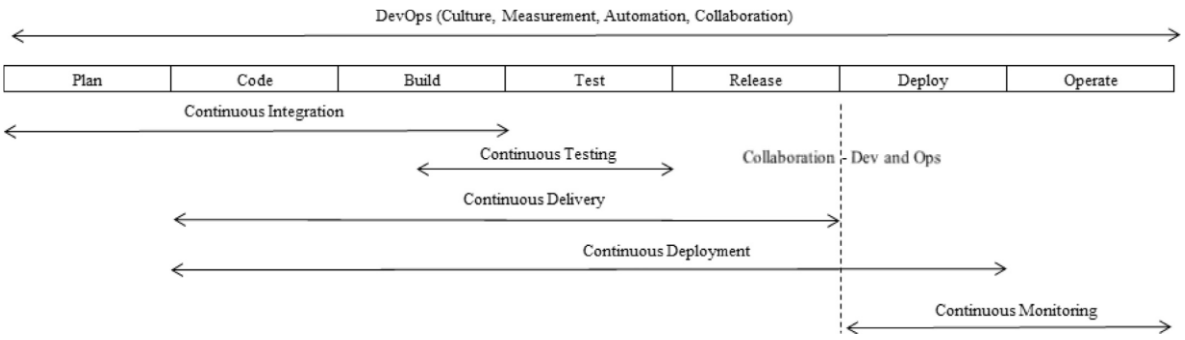

Figura 2 Relación Entre DevOps y Entrega Continua

DevOps frecuentemente está mal definido, los atributos clave, descritos por *V. Gupta*, deben ser usados para caracterizar y evaluar las implementaciones de *DevOps*. Estos pueden ser agrupados en 4 categorías, donde la automatización y el Control del Código son atributos clave independientes que afectan la calidad de la implementación de *DevOps*:

1. Automatización: De procesos tediosos o repetitivos y el aprovisionamiento de recursos de configuración y archivos de definición (Infraestructura como código)
2. Control de Código: Administración del cambio y facilitación de Despliegues
3. Cohesión de Equipos: Opuesto al modelo tradicional de equipos aislados por función
4. Continuous Delivery: Mejores prácticas

En la Tabla 1 se presentan los atributos de *DevOps*, de acuerdo con el modelo de *V. Gupta*:

Automatización	Control de Código Fuente	Cohesión de Equipos	Entrega Continua
Revisiones Automáticas de Código	Patrones en las Ramas	Involucramiento de Operaciones en Desarrollo *(OpsInDev)*	Retroalimentación Continua
Pruebas Automatizadas	Cambios en las Ramas	Involucramiento de Desarrollo en Operaciones *(DevInOps)*	Cambios de Código Diario
Despliegues Automatizados	Dispersión en las Ramas		Desarrollo Iterativo
Herramientas Automatizadas para Monitoreo	Profundidad en las Ramas		Mantenibilidad de Código
Infraestructura como Código	Alternancia de Funciones		Monitoreo del Sistema
			Pruebas Pronto y Frecuentes

Tabla 1 Atributos de DevOps

A continuación, se explica de manera breve cada atributo clave:

Automatización

Revisiones Automáticas de Código: Se refiere a predefinir las reglas de calidad del código y las mejores prácticas para automáticamente detectar defectos durante el *check-in* de

Continuous Integration. Este atributo está relacionado a mitigar el riesgo y reducir el tiempo-costo, está demostrado su eficiencia para mejorar la calidad de software.
Pruebas Automatizadas: Ejecución automática de test cases para verificar que el código que se ha ingresado cumpla los requerimientos.

Despliegues Automatizados: *Deployment* de la aplicación automatizados a ambientes productivos. Reduce los riesgos y los tiempos de liberación de nuevas funcionalidades, mejoras o correcciones de las aplicaciones, proporcionando cambios más rápidos y en el mercado.

Herramientas Automatizadas para Monitoreo: Monitorear ambientes productivos, se puede conseguir un nivel de calidad más alto, monitoreo continuo requiere herramientas automatizadas y reduce los riesgos cuando se incrementa la confianza en el producto.

Infraestructura como Código: Administrar la infraestructura con interacciones de scripts con el entorno, permite cambios más rápidos para reducir el riesgo y la frecuencia de despliegues de aplicación bajo demanda y mejorar la calidad debido a la rápida recuperación en caso de fallo.

Control de Código Fuente

Patrones en las Ramas *(Branching pattern)*: Las ramas (o branches en inglés) son usados en control de código para aislar cambios de código. El patrón para aislar trabajo individual y subsecuentes estrategias de merges, afecta directamente la calidad del código y su frecuencia de liberación.

Cambios en las Ramas *(Branching Changes)*: Tareas de Branching comparadas con tareas de desarrollo representa la complejidad de los esfuerzos de integración de código. Valores altos son prohibitivos para cambios rápidos y frecuentes en los *releases*.

Dispersión en las Ramas *(Branching Scatter)*: El número de branches que contienen a un componente de software, valores altos indicar tasas de fallas altas y redicen la calidad del código.

Profundidad en las Ramas *(Branching Depth)*: Cambios de código en los branches que no son descendientes directos del Branch maestro no son indicativos de calidad en el código.

Fue demostrado que tienen de muy bajo a ningún impacto, este atributo puede considerarse invalido.

Alternancia de Funciones *(Feature Toggle)*: Funcionalidades “prendidas” o “apagadas”, permiten deshabilitar bajo las fases de desarrollo y pruebas. Este enfoque simplifica la estrategia de branches y releases sin tiempo muerto.

Cohesión de Equipos

Involucramiento de Operaciones en Desarrollo *(OpsInDev)*: La colaboración a través de información y responsabilidad compartida. Los equipos de operación proporcionan visión en la infraestructura y los requerimientos de los ambientes a los equipos de desarrollo, logrando mayor productividad y calidad en el software.

Involucramiento de Desarrollo en Operaciones *(DevInOps)*: La colaboración a través de información y responsabilidad compartida. El equipo de desarrollo proporciona los requerimientos de implementación y la visión del diseño, logrando mayor productividad y calidad del software.

Entrega Continua

Retroalimentación Continua: Los sistemas son diseñados para exponer información relevante. La eficiencia en los procesos de desarrollo es una medida de acuerdo con las métricas de rendimiento. Actividades de retroalimentación son compartidas con los *Stakeholders* en tiempo real, proporcionando una mejor calidad en el servicio debido a la rapidez en la resolución de problemas y reducción de los ciclos, incrementando la eficiencia en los procesos.

Cambios de Código Diario: Requiere que los desarrollados hagan check-in del código de manera diaria. Es un principio fundamental de Continuous Integration. Reducir la integración y los problemas de Merge, incrementando la calidad del software.

Mantenibilidad de Código: La habilidad para fácilmente acomodar cambios futuros cuando sean requeridos. Incrementando la calidad del software y cambios más rápidos.

Monitoreo del Sistema: Monitoreo del software y la infraestructura en los ambientes de producción permite la medición en la eficiencia de procesos y mejor calidad de software a través de los datos.

Pruebas Pronto y Frecuentes: Las pruebas pueden ser usadas para proporcionar retroalimentación para las actividades de desarrollo y deben ser usadas estratégicamente. Pruebas tempranas son usadas para mejorar procesos e incrementar productividad, menores costos e incrementar la calidad del software.

Desarrollo Iterativo: Implementar tareas grandes en pequeños fragmentos, mejora la calidad del código, reduce riesgos y habilidad integración y liberación continua, lo cual se traduce en releases más rápidos y despliegues de aplicaciones más frecuentes [9].

3.2.1. Despliegue Continuo (*Continuous Deployment*)

La etapa final de la automatización del proceso de desarrollo de software es el Despliegue Continuo, que implica que cada actualización de código que pasa las pruebas de software, es desplegado en el ambiente de producción (o similar) como una construcción exitosa. El razonamiento detrás de esto es, que se va a desplegar o "*deployar*" un cambio a producción, tarde que temprano. Entre más pronto se haga, es mejor la probabilidad de que se puedan solucionar defectos más rápido. Es más fácil de recordar que se hizo el día de ayer que puedo haber causado un defecto que recordar lo que se hizo hace 2 meses que pudo ocasionarlo.

Considerando un escenario donde hay un cambio de código, que es sometido a producción y se tiene el mensaje de error de producción 5 minutos más tarde, será más fácil encontrar y arreglar el defecto y 5 minutos después el ambiente de producción estará funcionando y libre de defectos. Desafortunadamente, la mayoría de los gerentes y dueños del software se ponen muy nerviosos en el tema de despliegues automatizados, esto debido a que tienen una noción o sentimiento de que hay menor control en el proceso, en lugar de que lo ejecute un integrante del equipo manualmente, cuando en la práctica es todo lo contrario, ya que se podría reaccionar de una manera más ordenada o estructurada si ocurre alguna eventualidad.

En conclusión y como parte importante en la implementación de *DevOps*, las diferencias entre estos 3 conceptos, Integración Continua *(Continuous Integration)*, Entrega Continua

(Continuous Delivery) y Despliegue Continuo *(Continuous Deployment)* pueden ser hasta cierto punto ambiguas o confusas. Para dar mayor claridad en esto, se presenta en la Tabla 2, las etapas del ciclo de vida del software indicando donde estos conceptos inician y terminan.

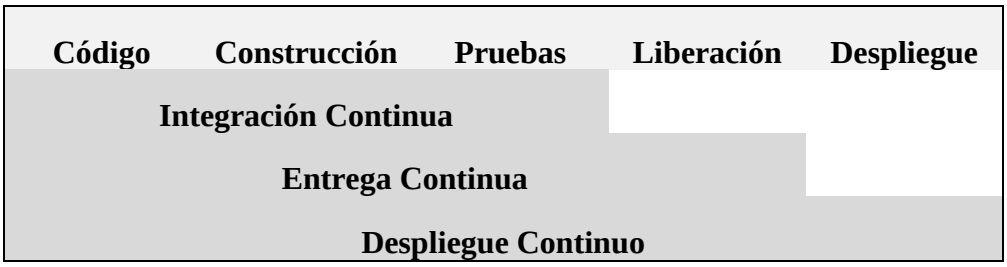

Tabla 2 Los diferentes tipos de "*Continuous*" y su relación con las diferentes etapas

4. Modelo y Arquitectura del Dashboard

Con todo lo analizado y descrito previamente respecto a *DevOps*, su cultura, prácticas y herramientas, se presenta el modelo general en la Figura 3, que describe el ***DOMD***, el cual consolida herramientas estándar en la Industria de Tecnologías de Información, define los roles de interacción y presenta los resultados del Sistema de Información.

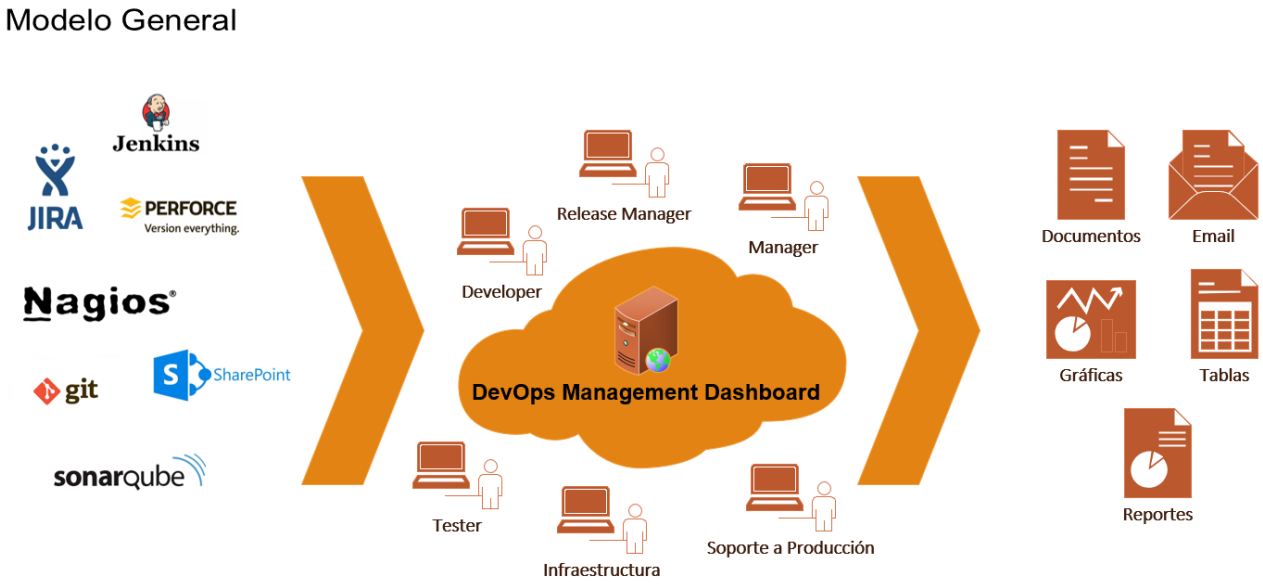

Figura 3 Modelo General de DevOps Management Dashboard

Vale la pena recapitular conceptos básicos retomados de capítulos anteriores, para después continuar con los detalles de Diseño y Arquitectura del ***DOMD***.

Recapitulando conceptos vistos en la sección 3.2**,** *DevOps*, es un conjunto de herramientas y prácticas enfocadas en la colaboración, integración, estandarización, automatización y comunicación para el desarrollo de software y las operaciones de TI en una empresa; ayuda a las organizaciones a ser altamente eficientes y productivas, eficientiza los flujos de trabajo y mejora la colaboración entre los diferentes equipos de TI.

Otro concepto fundamental para entender el ***DOMD*** son las ***Metodologías Ágiles*** [1]en el Desarrollo de Software, su objetivo es generar valor, en unidades de software pequeñas y funcionales clave, para satisfacer los requerimientos del cliente, dichas metodologías utilizan enfoques flexibles y un trabajo en equipo constante para mejorar todo el tiempo, estas metodologías por lo general constan de equipos pequeños auto organizados, compuesto por el cliente, un dueño de aplicación, *Scrum Master* y equipo de desarrollo; también se favorece un enfoque sencillo y compacto en la documentación del software y está totalmente adaptada a los cambios que pudieran ocurrir en las diferentes etapas del ciclo de vida, siendo muy flexible.

Por lo tanto, el ***DOMD*** es una plataforma web, cuyo propósito fundamental es centralizar y permitir el acceso de los datos, de todas las herramientas o aplicaciones que están alineadas a los diferentes procesos y roles (dueños de la aplicación, scrum masters, gerentes, *testers*, equipo de desarrollo, equipo de soporte a producción, infraestructura/operaciones, etc.) que participan en el ciclo de vida del software, utilizando el estándar de DevOps y Agile. La aplicación desplegará tablas consolidadas, gráficas, estadísticas, envió de correos electrónicos, actividades a realizar, calendarización y notificación eventos importantes futuras, etc.; al ser una aplicación web estándar con tecnologías populares en la industria (Java[2] y Node.js[3]), su despliegue será muy simple en entornos propietarios o *cloud.*

4.1. Arquitectura

La Arquitectura es un tema muy relevante, definirla en el contexto del software puede ser un tema complejo, en caso de no realizarse de manera adecuada, generaría problemas a futuro en el funcionamiento y rendimiento de la aplicación, por lo tanto, es importante destacar los siguientes principios al definir una arquitectura.

Múltiples Stakeholders, explicados más a detalle posteriormente: esto implica que diferentes roles utilizarán la aplicación, por lo cual esta se tiene que concebir y desarrollar pensando en todas las actividades que realizaran en el día a día como parte de los procesos.

[1] https://www.redhat.com/es/devops/what-is-agile-methodology
[2] https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)
[3] https://es.wikipedia.org/wiki/Node.js

Separación de Responsabilidades: al ser una aplicación web, se va a requerir de un navegador web para su ejecución, existen una gran cantidad de tecnologías que permiten el desarrollo de aplicaciones web, por lo cual, definir claramente las diferentes capas de la aplicación, su estructura e interacción, es parte fundamental del ***DOMD***

*Enfoque en la Calidad***:** al definir una arquitectura de software, la calidad y la operación de la aplicación son aspectos altamente relacionados, conceptos como extensibilidad, confiabilidad, disponibilidad y seguridad son clave para el ***DOMD,*** al utilizar tecnologías estándar en la industria (*Java* y *NodeJs*), estos aspectos se aplican directa e indirectamente.

Al considerar todos estos principios en la Arquitectura del ***DOMD,*** se obtendrían los siguientes beneficios: mayor escalabilidad de la aplicación para futuras actualizaciones, un rendimiento estable en la ejecución de la aplicación con un mínimo de errores, el código será mucho más fácil de mantener, actualizar y escalar.

A continuación, se profundizará en la arquitectura empleada para la creación del ***DOMD***

En la primera capa de la Arquitectura, representada en la Figura 4, se observan los componentes desde una perspectiva más general, algunos de los roles de interacción con la aplicación, entre los que se encuentran el Desarrollador, el Tester, el Gerente, el Release Manager, que protocolo de comunicación de red se utiliza para que los clientes y las aplicaciones se comuniquen, en este caso es HTTPS[4], el cual es uno de los protocolos de Red estándar más utilizados en el mundo y que se encarga de transferir contenido en Internet; a su vez, este protocolo también nos ayudara con la comunicación de los sistemas de terceros o "*Third Party*" que proporcionan datos al ***DOMD.***

[4] https://en.wikipedia.org/wiki/HTTPS

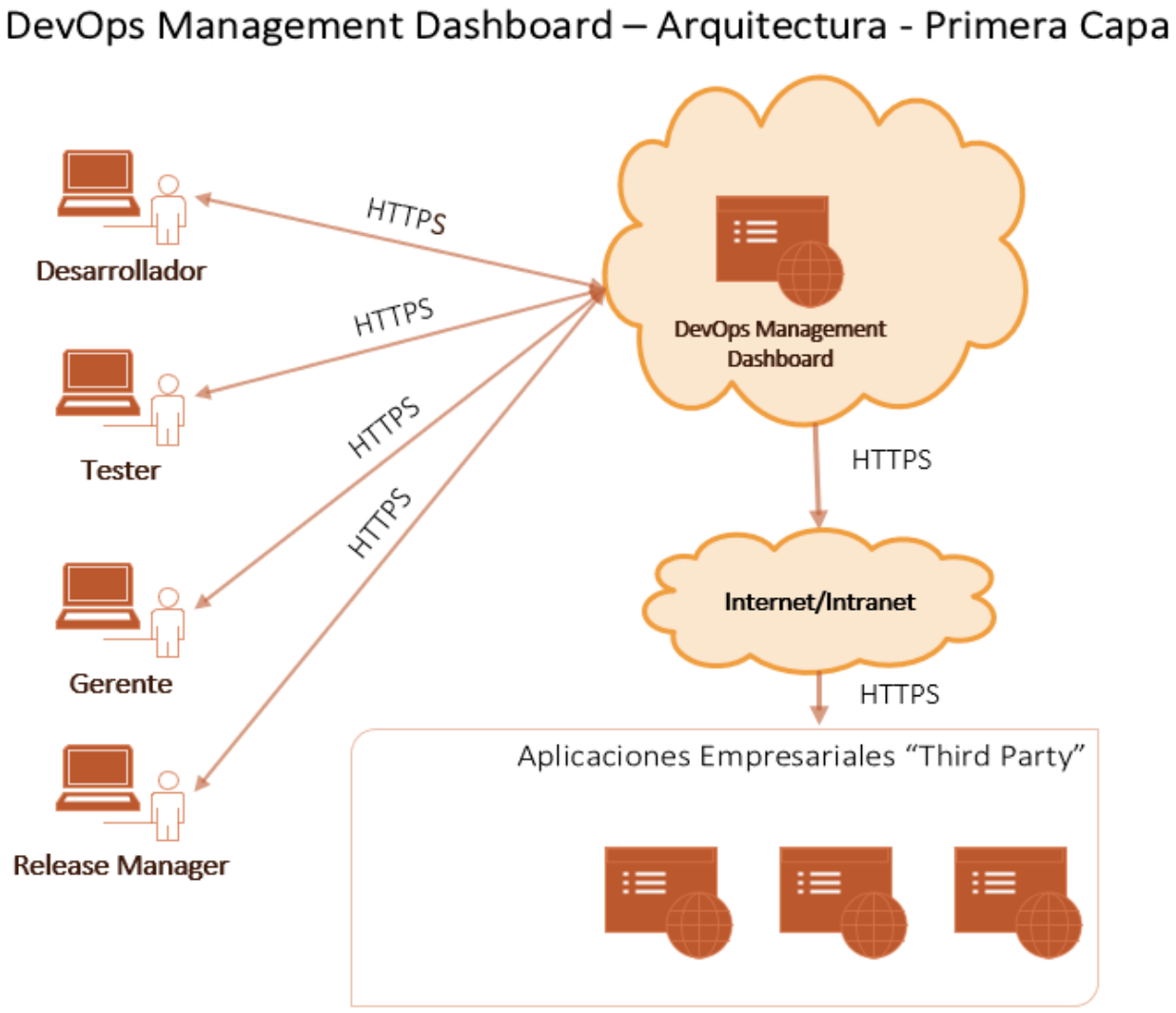

Figura 4 Primera Capa de Arquitectura

En la Figura 5 descendemos un nivel y se proporciona más detalle respecto a los tipos de servidores involucrados, al ser una aplicación web tradicional para los estándares actuales, se requerirá de un *Servidor de Aplicaciones* [5] para desplegarla, existen una gran variedad, entre los que sen encuentran WebLogic, Websphere, Wildfly, Glassfish, etc. La aplicación tendrá la flexibilidad de ser "*deployado*" en cualquiera de ellos con un mínimo de configuración, además la aplicación se conectará a una *Base de Datos Relacional*[6], entre las

[5] https://www.oracle.com/java/technologies/compatibility-jsp.html
[6] https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/

más populares tenemos, Oracle, MySQL, SQL Server, MariaDB, etc., al igual que el Servidor de Aplicaciones, la aplicación tendrá la flexibilidad de ser conectada en cualquiera de las bases de datos previamente listadas con un mínimo de configuración.

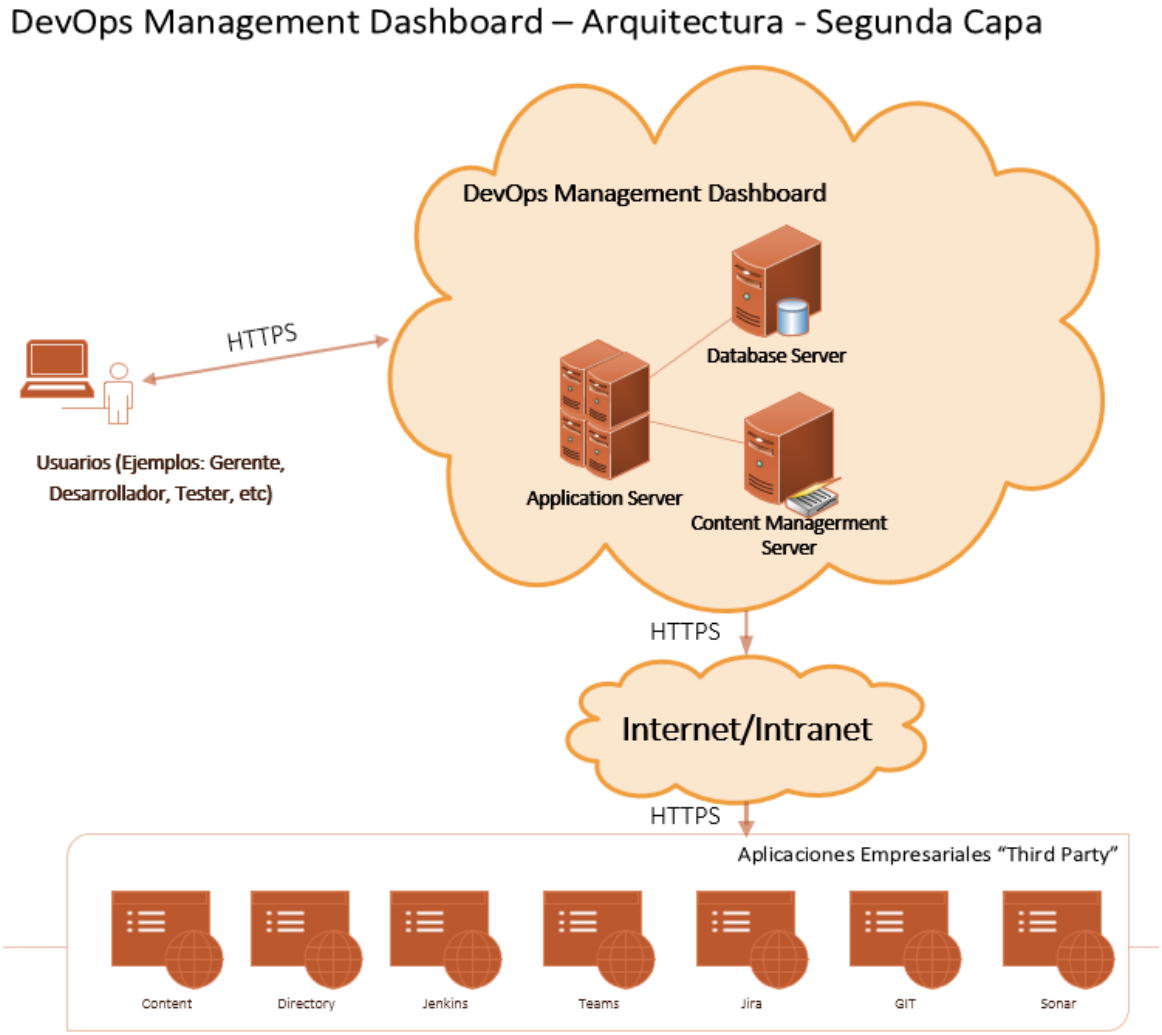

Figura 5 Segunda Capa de Arquitectura

El ***DOMD*** también soportara conectividad con un Servidor de Administración de Contenidos[7], en caso de querer guardar u obtener referencias de archivos, imágenes, etc. Y en la parte inferior de la *Figura 5*, se observa la interacción que tendrá el ***DOMD,*** con

[7] https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_contenidos

aplicaciones de terceros o “Third Party” para extraer contenido, ejemplos de estas son Sonar, GitHub, Jira, Teams, etc.

Para concluir con la Arquitectura del ***DOMD*** se presenta la Figura 6, que es el nivel más bajo de detalle presentado en este documento, se dividió la aplicación en una Arquitectura basada en Servicios, los cuales son de tipo *REST*[8], se divide la aplicación en Front End y Back End. En la parte de Front End se utilizará JavaScript/ES6 [9] con el soporte de Node.js y el Framework React[10] para generar todos los componentes visuales que se presentaran en el navegador web y son responsables de mostrar información en formato HTML[11], así como formularios de entrada para que el usuario pueda interactuar con la aplicación, dicho componente generará peticiones HTTP hacia el *Back End* usando diferentes métodos [12] de acuerdo con la necesidad del servicio.

Una vez que las peticiones HTTPS en el Front End son enviadas, el Back End procesará dichas peticiones utilizando el Estándar Enterprise de Java[13] con lo que podrá identificar el destino de la petición y direccionarla al proceso adecuado dependiendo el caso de uso; entre las actividades más comunes se encuentran inserciones y consultas a Base de Datos, con la ayuda del Framework Spring [14]y JPA[15], otras tareas fundamentales serán la consulta a aplicaciones Third Party las cuales también expondrán servicios ya sea por *REST* o por SOAP y que pueden ser fácilmente invocados en el Back End del ***DOMD*** para su posterior presentación en la vista del Usuario.

[8] https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional
[9] https://www.w3schools.com/js/js_es6.asp
[10] https://es.wikipedia.org/wiki/React
[11] https://es.wikipedia.org/wiki/HTML
[12] https://www.w3schools.com/tags/ref_httpmethods.asp
[13] https://es.wikipedia.org/wiki/Java_EE
[14] https://es.wikipedia.org/wiki/Spring_Framework
[15] https://es.wikipedia.org/wiki/Java_Persistence_API

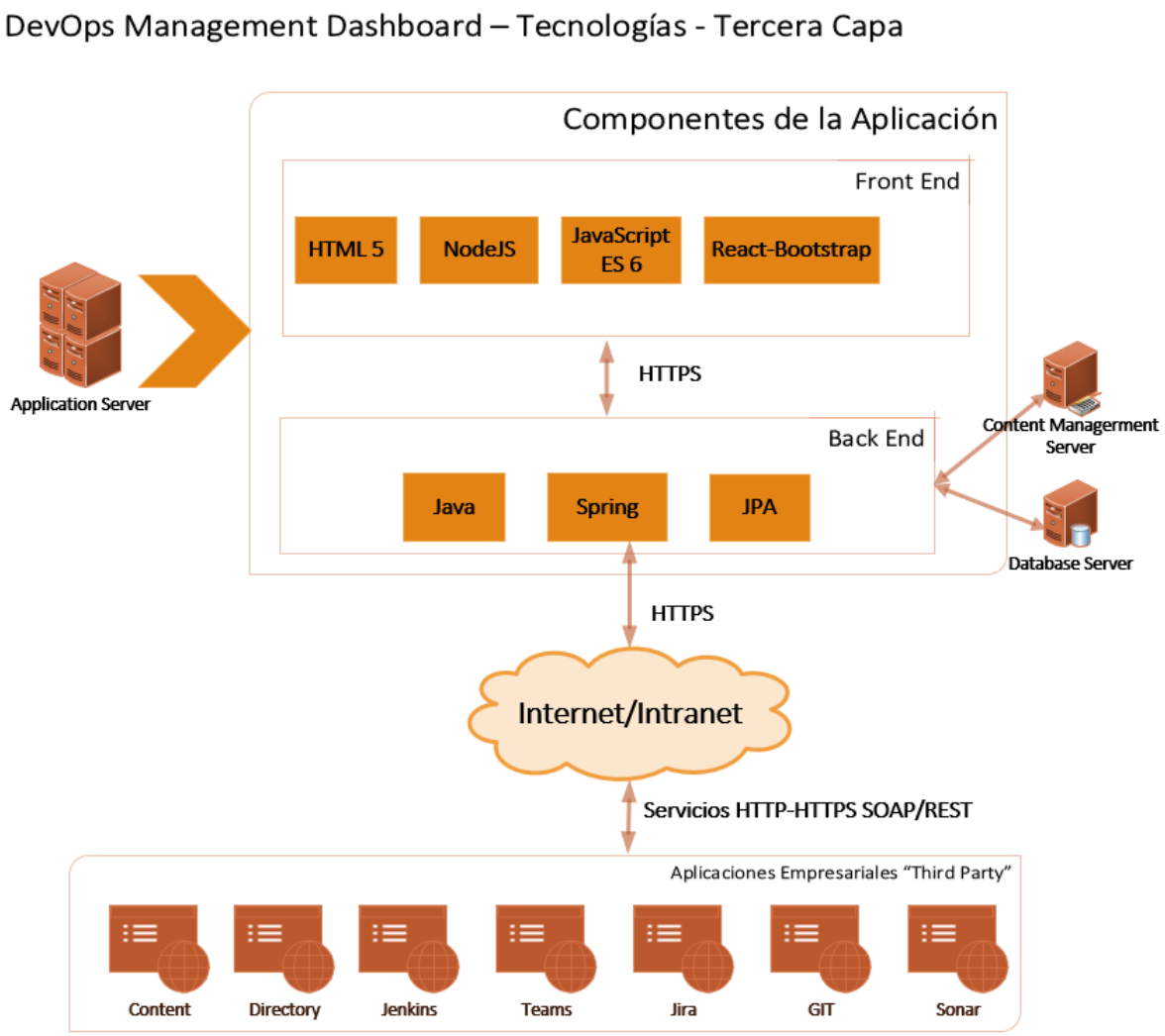

Figura 6 Tercera Capa de Arquitectura

4.2. Modelo Entidad Relación

Como parte de la propuesta de valor, el ***DOMD***, contará con una Base de Datos Relacional [16] Interna para administrar los aspectos clave de un proyecto de Desarrollo de Software con metodología Ágil y que considera los roles e interacciones que se describieron previamente.

[16] https://www.oracle.com/mx/database/what-is-a-relational-database/

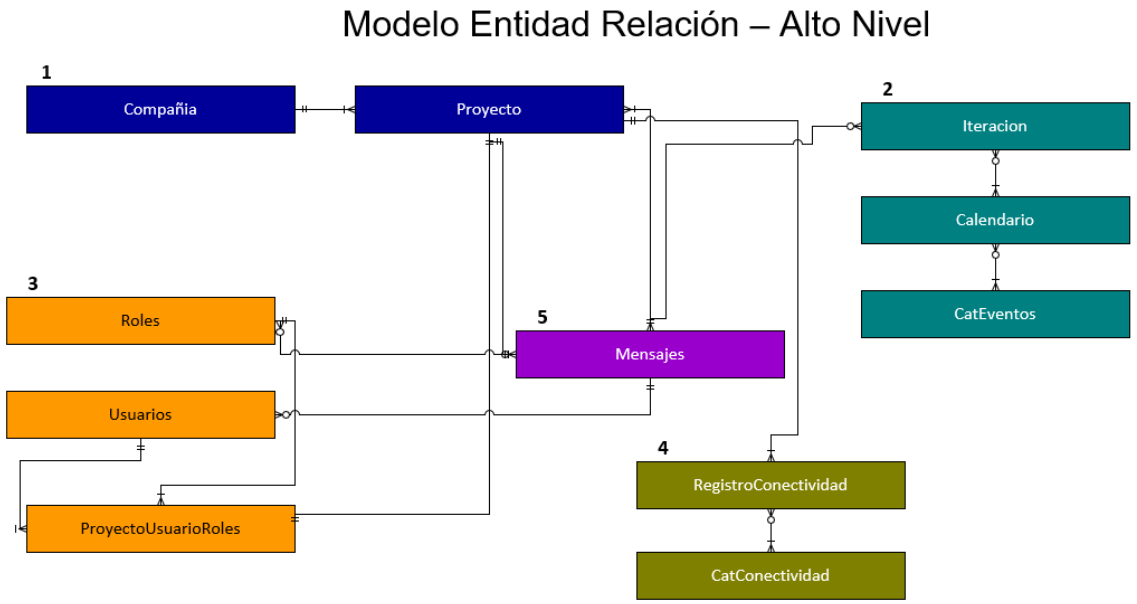

Figura 7 Modelo Entidad Relación – Alto Nivel

Para un mejor entendimiento de la funcionalidad y la estructura de la Base de Datos se dividió en 5 secciones y por colores, ver *Figura 7*. En la sección uno se considera la compañía y la capacidad de tener múltiples proyectos, en la sección dos, cada proyecto, que tiene una fecha de inicio y de fin, puede tener múltiples iteraciones y cada iteración administra un calendario de eventos, en la sección tres, se administran los roles y los usuarios que puede contener un proyecto, en la sección cuatro se administran las relaciones que existen entre los proyectos y las aplicaciones *Third Party* y por último, al centro tenemos la tabla que administra todos los mensajes y que se compone de todas las llaves foráneas[17] de todas las demás entidades, con esto se concretan las expectativas de las funcionalidades definidas previamente como parte de los diferenciadores del ***DOMD***

[17] https://es.wikipedia.org/wiki/Clave_for%C3%A1nea

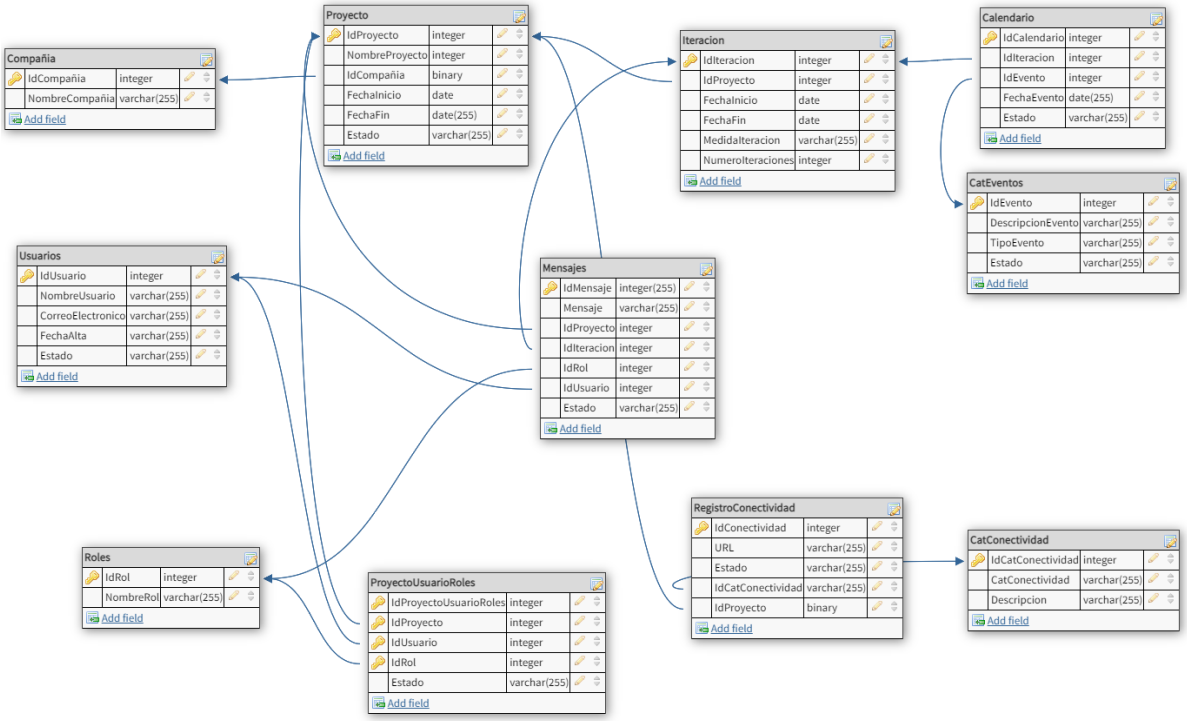

Figura 8 Detalle del Modelo Entidad Relación

Para profundizar en el aparato de la base de datos, también se presenta la *Figura 8*, que muestra las entidades de la base de datos, sus relaciones entre ellas, así como los atributos, los respectivos tipos de datos y restricciones de llaves primarias y foráneas, esto ayuda a tener una visión más completa de como el ***DOMD*** gestionará los datos fundamentales para su operación, que son parte del valor añadido de esta plataforma.

4.3. Diferenciadores y Atributos de DevOps Management Dashboard

A continuación, se enlistan los principales diferenciadores de la propuesta presentada en este documento respecto de los *Dashboards* existentes.

Configuración de Timeline e Iteraciones

La plataforma permitirá planear y definir eventos importantes en el calendario, así como fragmentar por iteraciones el proyecto para una mejor administración de este.

Una Iteración, en el contexto Ágil, es un tiempo definido durante el cual se realiza el Desarrollo de Software, la duración presenta las siguientes características:

- Puede variar de proyecto en proyecto, usualmente entre una y cuatro semanas
- En la mayoría de los casos las iteraciones están acotadas a la duración del Proyecto

Un aspecto clave, es que, desde el enfoque de vista Ágil, el proyecto consiste exclusivamente en una secuencia de iteraciones, generalmente con una excepción de una breve etapa de visión y planeación previa al desarrollo de software y una etapa de cierre a la conclusión de este.

En general las iteraciones están alineadas con los calendarios de la semana, usualmente empezando en lunes y terminando en viernes, esto es más una cuestión de conveniencia y cada proyecto o equipo podría adoptar diferentes convenciones.

La longitud fija de las Iteraciones ayudará a madurar a los equipos de trabajo, ya que entre más pasa el tiempo, más se irán acoplando los elementos y sus interacciones, el rendimiento mejorará, con lo cual se observará una mayor resolución de tareas y calidad en periodos de tiempo iguales.

En la Figura 9, se genera una representación de una Iteración de 4 semanas en la cual se marcan las fechas de inicio y de fin de cada semana, al término de esta iteración se espera la generación de un entregable útil para el cliente.

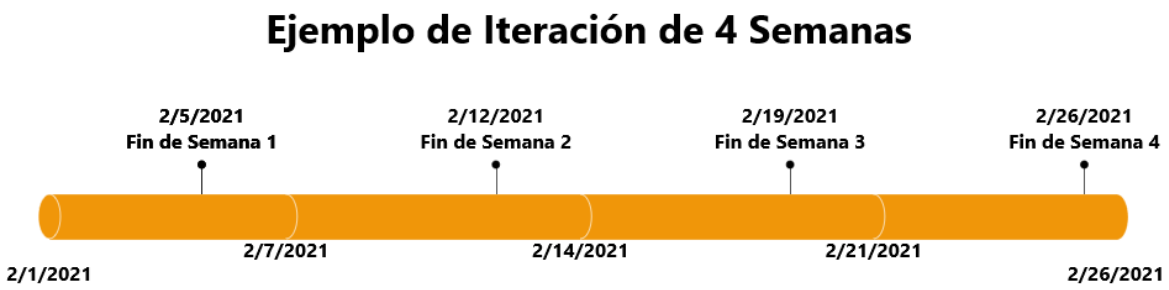

Figura 9 Ejemplo de Iteración

En el ***DOMD*** será posible definir la fecha inicial y final del proyecto y el número de semanas por Iteración; todo esto mediante una interfaz de configuración y almacenada en una base de datos relacional.

Otro aspecto importante en cualquier proyecto es la adición de eventos o fechas importantes durante el proyecto, esto genera mayor comunicación y visibilidad a todos los miembros del equipo, un ejemplo de esto puede ser el "Despliegue a Ambientes de Pruebas de determinada Iteración", con lo cual los equipos de Desarrollo, Operaciones y Pruebas saben con anticipación y pueden realizar una mejor planeación de sus actividades. Esta característica también podrá ser configurada dentro de la aplicación y será almacenada en la base de datos relacional.

Bandeja de Entrada y Mensajes

La aplicación contará con una Bandeja de Entrada con actividades pendientes por realizar para ese miembro en particular, como por ejemplo algún *"Merge"* de código pendiente, algún defecto por completar, alguna tarea abierta de la iteración.

También contará con un sistema de mensajes para comunicar eventos o situaciones particulares en los siguientes niveles: proyecto, iteración, rol de usuario y usuario en particular.

Conectividad con Plataformas *Third Party*

El ***DOMD*** podrá conectarse con aplicaciones de terceros. En la Tabla 1 se hace referencia a las APIS y su documentación estándar, se consideran aplicaciones como Teams, Jira, Git, Sonar, Jenkins, SoapUI, Microsoft Active Directory, etc.

Nombre	Propósito	URL
Teams	Trabajo colaborativo	**https://docs.microsoft.com/en-us/graph/api/resources/teams-api-overview?view=graph-rest-1.0**
Jira	Administración de Agile	**https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/#version**
Git	Versionador de Código	**https://docs.github.com/es/rest/overview/resources-in-the-rest-api**
Sonar	Análisis de Código Estático	**https://docs.sonarqube.org/latest/extend/web-api/**
Jenkins	Servidor de Automatización	**https://www.jenkins.io/doc/book/using/remote-access-api/**

Tabla 3 Conectividad Third Party

Plataforma Web y Cloud

Como parte de una característica fundamental para una aplicación empresarial, esta será compatible con los navegadores web más utilizados (Chrome, Edge, Safari), utilizando el protocolo HTTP-HTTPS y además podrá ser alojada en servidores propietarios o en una plataforma Cloud, ejemplo AWS[18] o Azure[19], con lo cual su administración será más simple y estandarizada.

Base de Datos Centralizada

Al ser una aplicación empresarial que soportará procesos especializados dentro de la organización, se creará una Base de Datos Central, la cual servirá para definir los proyectos dentro de la organización, los usuarios relacionados a dicho proyecto, al estar enfocado en DevOps y Agile, también contendrá los datos de las Iteraciones, así como los eventos importantes en el calendario; también contará con la información de conectividad con los sistemas *Third Party*.

Conectividad con Directorio de Identidad y Definición de Roles

La mayoría de las organizaciones modernas cuenta con un directorio de identidad, por lo cual la integración con esta herramienta es fundamental, permitiendo agregar al subgrupo de empleados que se podrán autenticar en el Sistema y por consecuencia definir sus roles en el proyecto (desarrollo, soporte a producción, alcance, pruebas, programa, infraestructura).

Notificaciones Automáticas por Correo Electrónico

La comunicación es un aspecto fundamental para cualquier proyecto, por lo que la notificación automática de información será implementada en esta herramienta, informes de ejecuciones de pruebas, fallas en el build, cambios pendientes de código, tareas sin completar, defectos abiertos, son solo algunos ejemplos de la información que será comunicada de manera recurrente.

En conclusión, después de listar y explicar de manera concreta los diferenciadores y atributos del ***DOMD*** se puede observar claramente el valor agregado de la aplicación, su enfoque a la

[18] https://es.wikipedia.org/wiki/Amazon_Web_Services
[19] https://es.wikipedia.org/wiki/Microsoft_Azure

centralización, consolidación y simplificación de procesos y datos es de gran relevancia, para que cualquier organización, que actualmente ejecute DevOps con Agile, pueda ser aún más eficiente, ordenada y precisa en sus procesos de desarrollo de software y liberación a ambientes productivos.

4.4. Roles e Interacciones en el DevOps Management Dashboard

Un aspecto muy importante en el ***DOMD*** es la integración de todos los roles que podrían laborar en un proyecto de Desarrollo de Software, modelar sus funciones e interacciones y alinearlo con procesos estándar que puedan ser adoptados por cualquier organización que utilice DevOps y Agile.

A continuación, se describe brevemente los roles considerados en la aplicación:

Equipo de Alcance

Su función es trabajar muy de cerca con el cliente, los gerentes y el equipo de desarrollo, poseen un conocimiento de la lógica del negocio, así como conocimiento técnico básico que ayuda a identificar cambios, requerimientos, factibilidad, riesgos y generación de documentación para que los equipos de desarrollo de software los analicen y se le dé continuidad al mantenimiento, creación y actualización del software.

Equipo de Administración

Su función es estratégica para el desarrollo del proyecto, se encargan de planear, ejecutar, monitorear, controlar y comunicar el progreso del proyecto, también participan en la definición o actualización de nuevos procesos que agreguen valor, ayudan a validar alcance, riesgos y funcionalidades, sirven como punto de enlace con los diferentes equipos y definen y administran métricas. Entre este rol se pueden encontrar figuras particulares, como son los gerentes técnicos, arquitectos, de proyecto, de *release*, de programa, etc.

Equipo de Desarrollo

Su función es implementar las actualizaciones de software, trabajan de cerca con el equipo de alcance, de administración y de pruebas para validar el alcance de los nuevos requerimientos o actualizaciones, deben de tener un conocimiento técnico y profundo en lenguajes de programación, bases de datos, sistemas operativos, scripting, etc. y también

conocimiento básico del negocio, una vez entregado el software también trabajan en la corrección de defectos que pudieran aparecer en etapas de prueba o utilización del cliente.

Equipo de Soporte a Producción

Su función, como su nombre lo indica, es estar al pendiente de la versión productiva de la aplicación, esto significa la versión que se ejecuta todos los días con el cliente en particular y que ejecuta los diferentes procesos de negocio, se encargan de monitorear y corregir defectos en el software, también trabajan de cerca con el equipo de Desarrollo para transferir y entrenarse en nuevas funcionalidades o cambios que puedan ser liberados a Producción.

Equipo de Pruebas

Su función es la de planear, ejecutar y monitorear casos de prueba en las diferentes aplicaciones, ya sea a nuevas funcionalidades, a funcionalidades previas (regresión), pruebas unitarias, pruebas de sanidad, pruebas de rendimiento, pruebas de seguridad entre otros, también se encarga de levantar y validar defectos para que sean atendidos por los equipos de Soporte a Producción y Desarrollo.

Equipo de Infraestructura/Operaciones

Su función es la de monitorear el funcionamiento del hardware y el software, en caso de situaciones de bajo performance o error, deben de actuar de manera expedita para brindar el servicio con la calidad y disponibilidad acordada con el cliente, también cumplen la función de actualizar los diferentes ambientes de preproducción y producción, de actualizar el software de manera periódica para estar en cumplimento con las distintitas regulaciones y cuestiones de seguridad, trabajan de cerca con los equipos de Administración y Desarrollo.

Para clarificar más en las posibles interacciones de los distintos roles en un proyecto con DevOps y Agile, a continuación, se presenta la Tabla 4, Matriz de Interacción de Roles, donde se clasifica en 3 categorías (Alta, Media y Baja), la intensidad o periodicidad de dichas interacciones.

	Administración	Alcance	Soporte a Producción	Infraestructura /Operaciones	Desarrollo	Pruebas
Administración	X	Alta	Alta	Alta	Alta	Alta
Alcance	Alta	X	Baja	Baja	Alta	Media
Soporte a Producción	Alta	Baja	X	Baja	Media	Media
Infraestructura /Operaciones	Alta	Baja	Media	X	Media	Baja
Desarrollo	Alta	Alta	Media	Media	X	Media
Pruebas	Alta	Media	Media	Baja	Media	X

Tabla 4 Matriz de interacción entre los distintos roles

4.5. Herramientas *Third Party*

Las herramientas de trabajo auxilian y soportan muchas actividades en diferentes oficios e industrias, desde un martillo para un carpintero, un automóvil para un piloto de fórmula 1, un microscopio para un biólogo, un cuchillo para un carnicero, etc.

En el mundo del software ocurre lo mismo, existen una gran cantidad de herramientas que ayudan en diferentes áreas del ciclo de desarrollo de software, sus principales propósitos son almacenar, estructurar y ordenar datos, automatizar procesos, procesar información, comunicar equipos, almacenar cambios de código, escanear código en buscar de vulnerabilidades, etc.

Es por ello, que el ***DOMD*** clasifica las diferentes herramientas que pueden ser utilizadas en proyectos de desarrollo de software con DevOps y Ágil y ayuda a presentar, organizar, estructurar y utilizar los datos e información de dichas herramientas en un entorno integrado y centralizado.

Dentro de las clasificaciones que el ***DOMD*** realiza, acotamos las siguientes:

Colaboración: Se refiere a aquellas herramientas que sirvan para comunicarse de manera síncrona o asíncrona con individuos, foros, equipos, etc., que formen parte del proyecto e

incluso externos, así como para compartir recursos: archivos de texto, hojas de cálculo, presentaciones o archivos de cualquier propósito.

Ciclo de Vida del Proyecto (Ágil): Este tipo de software es más especializado, sirve para administrar proyectos de software, permite la jerarquización de componentes de trabajo, desde lo general a lo muy particular, asignación de tareas, tiempos, características, etc., este tipo de aplicaciones y su utilización son críticas para una mejor administración de los proyectos de software.

Administración de Defectos: En esta categoría se encuentran aquellas aplicaciones que permiten registrar y administrar los defectos asociados al software, con lo cual su seguimiento y control es más eficiente, desde que es asignado hasta que es corregido, desplegado, probado y resuelto.

Versionador de Código: Se encuentran las herramientas que permiten administrar el código fuente de los proyectos de software, por el tipo de concurrencia y cambios continuos, este software es fundamental para que el código se mantenga integro y siempre disponible, otro apartado importante son las ramificaciones que pidieran existir del código principal y las estrategias de combinación o mezcla del código para mantener todos los cambios actualizados.

Analizador de Código Estático: Este tipo de herramientas cumplen la función de escanear el código fuente de los proyectos de software con el propósito de identificar malas prácticas, mal uso de la sintaxis o sentencias del lenguaje de programación, así como identificar posibles riesgos de seguridad.

Integración Continua: El propósito de esta categoría y tipo de software es la automatización de rutinas de trabajo, teniendo la capacidad de agendar eventos de ejecución o identificar detonantes que ejecuten otras rutinas, un ejemplo típico es la invocación de *pipelines* con el objetivo de compilar la última versión del código y colocar los artefactos generados en algún repositorio para su libre acceso.

Entrega Continua: En esta categoría se encuentras aplicaciones enfocadas en la automatización y el despliegue de componentes de software, esto requiere un gran dominio y control de todos los procesos que involucran una actualización de software.

Pruebas Unitarias: En este tipo de herramientas encontramos *frameworks* ligados a algún lenguaje de programación en particular o herramientas independientes capaces de aislar las pruebas de software y ejecutarlas de manera coordina, sincronizada o calendarizadas, proporcionando estadísticas de ejecución de éxito, fallo, errores, etc.

Medición de KPI: Este tipo de herramientas, sirven para registrar, gestionar y también graficar los indicadores clave de rendimiento del proyecto de software, es una utilidad más enfocada para los equipos de administración y sirve para de manera más simplificada generar información y conocimiento de las actividades realizadas durante todo el proyecto, un ejemplo típico son las gráficas donde se muestran los defectos abiertos por equipos o por las diferentes aplicaciones, donde rápidamente se pueden tomar mejores decisiones para corregir problemas o vicios en los proyectos de software.

En la Tabla 5 a continuación, se presentan las herramientas *Third Party* más utilizadas en la industria de Tecnologías de Información, clasificadas de acuerdo con su utilización y que tienen relación con DevOps y metodologías ágiles, las cuales se integrarían al ***DOMD***

Colaboración	**Ciclo de Vida del Proyecto (Ágil)**	**Administración de Defectos**	**Versionador de Código**	**Analizador de Código Estático**	**Integración Continua**	**Entrega Continua**	**Pruebas Unitarias**	**Medición de KPI**
Teams	Jira	Jira	Github	SonarQube	Jenkins	Docker	SoapUI	Geckoboard
G Suite	Microsoft TFS	Microsoft TFS	Git	Veracode	CircleCI	Kubernettes	Junit	Salesforce
Slack	Smartsheet	Bugzilla	Microsoft TFS	Checkmarx	CloudBees CodeShip		Cucumber	Tableau
Confluence		Github Issues	CVS	Codacy				SimpleKPI
Wordpress		Backlog	BitBucket					Asana

Drupal		Spira						
Dropbox		Rational ClearQuest						

Tabla 5 Matriz de Interacción de Roles

5. RESULTADOS Y DISCUSIÓN

A continuación, se presentarán imágenes de un prototipo funcional de la aplicación web, el cual implementa todos los principios y características previamente descritos en este proyecto de grado, que en conjunto conforman el ***DOMD.***

En la Figura 10, se presenta una pantalla de *login*, con el respectivo usuario y contraseña, su propósito es la de autenticar a un usuario y dar los controles de acceso necesarios en la aplicación dependiendo del Rol que desempeñe, recordemos que existen roles de Gerente, Desarrollador, Tester, Infraestructura, etc. En caso de no tener acceso a la plataforma, se agrega una opción para solicitar acceso, con lo que el Release Manager de la aplicación recibirá un correo y validará el perfil de la persona para otorgarle o negarle el acceso.

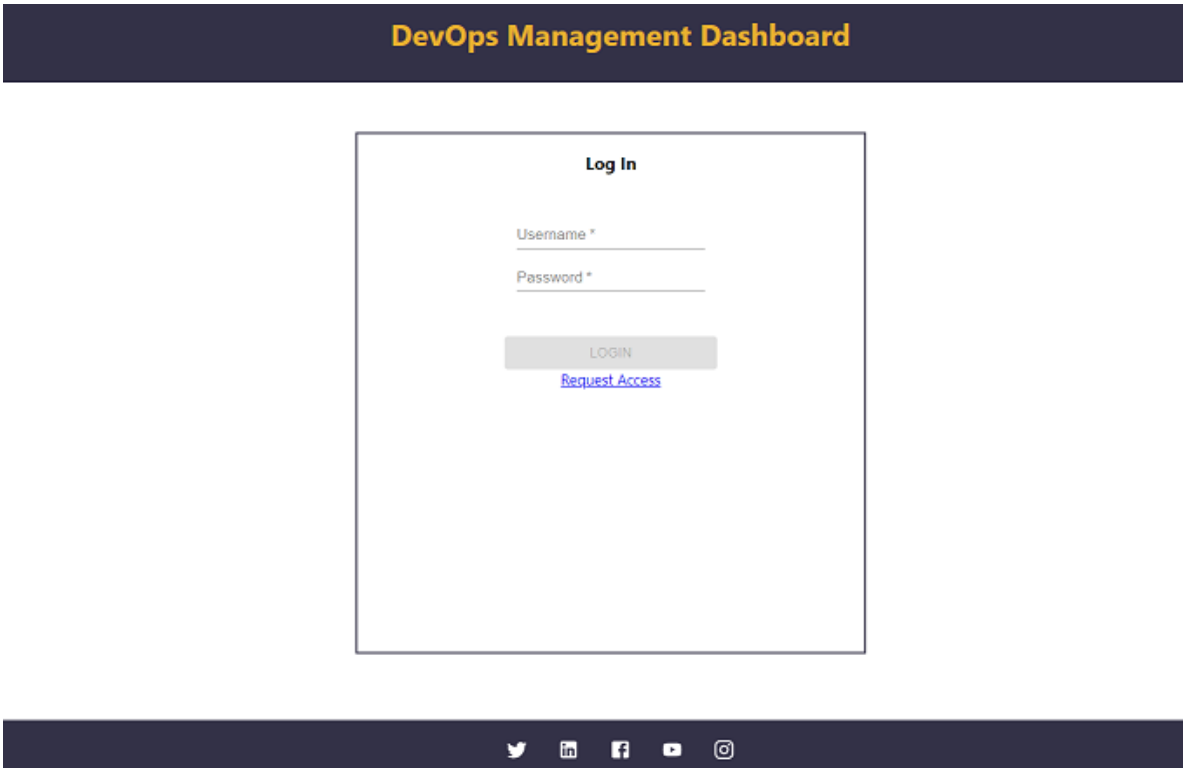

Figura 10 Pantalla de *Login*

En la Figura 11, se presenta el Menú principal de aplicación, como se puede observar, está clasificado por las diferentes funcionalidades previamente descritas, desde la administración

de las correcciones de código, ejecución de automatizaciones, cambios pendientes de código, seguridad del código, nuevos cambios de funcionalidades, preparación a liberación a producción, perfil del usuario, etc., cada categoría puede o no contener más secciones internamente para complementar los procesos en el Ciclo de Vida del Desarrollo de Software utilizando DevOps y Agile

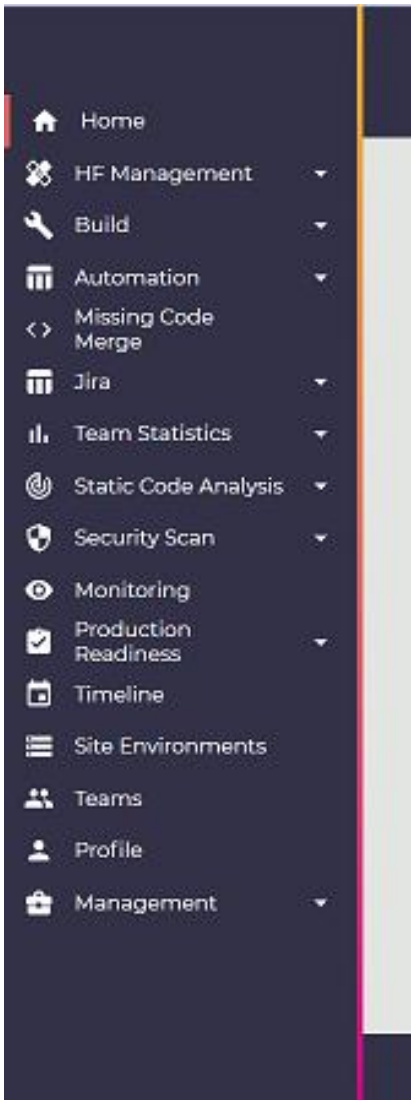

Figura 11 Menú Principal

El ***DOMD*** es una aplicación muy intuitiva y fácil de utilizar de acuerdo con la estructura y organización planteada, en el encabezado, en la zona izquierda, Figura 12, se observa claramente información acerca de las diferentes iteraciones que se encuentra actualmente en

Producción, Pruebas y Desarrollo, esto da una visión inmediata al integrante del equipo acerca del estado actual del proyecto.

En el mismo encabezado de la aplicación, solo que, de la parte derecha (Figura 13), se puede observar la opción de selección de iteraciones (el cual se muestra en ciertas condiciones), esto permite de manera ágil y precisa, poder consultar estados previos del proyecto, con lo cual se puede clarificar muy rápidamente, datos relevantes a los diferentes *stakeholders* en el proyecto.

Figura 12 Encabezado izquierdo con detalle de Iteraciones en progreso

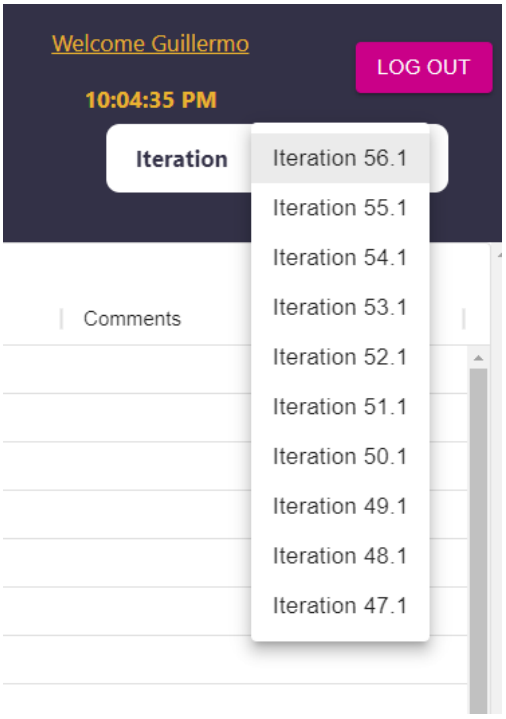

Figura 13 Encabezado derecho con histórico de Iteraciones

En la Figura 14, se presenta el diseño completo de la aplicación, se pueden observar algunos de los elementos previamente descritos en esta sección, el menú principal izquierdo con todas las secciones, el encabezado con la información relevante del proyecto y el estado actual, también se puede una sección al centro de “anuncios”, mediante el cual, el release manager, gerentes o líderes de equipo puede comunicar información a todos integrantes de los equipos que tengan acceso al ***DOMD,*** un ejemplo de esto es, cuando sería la fecha de la próxima liberación a un ambiente productivo.

Figura 14 Pantalla de Bienvenida

En la Figura 15, se presenta otra sección del ***DOMD***, *Cambios Pendientes de Código*, como se documentó y describió previamente, una de las características esenciales de esta aplicación, es la posibilidad de interactuar con aplicaciones de terceros, en este caso en particular, hacemos referencia a Tabla 5 en la columna de *Versionador de Código*, aquí podemos observar y validar por aplicación y por usuario los cambios pendientes, con lo cual se puede notificar mediante correo electrónico y de manera diaria para que el usuario tome una acción concreta y no olvide mezclar el código en las diferentes iteraciones.

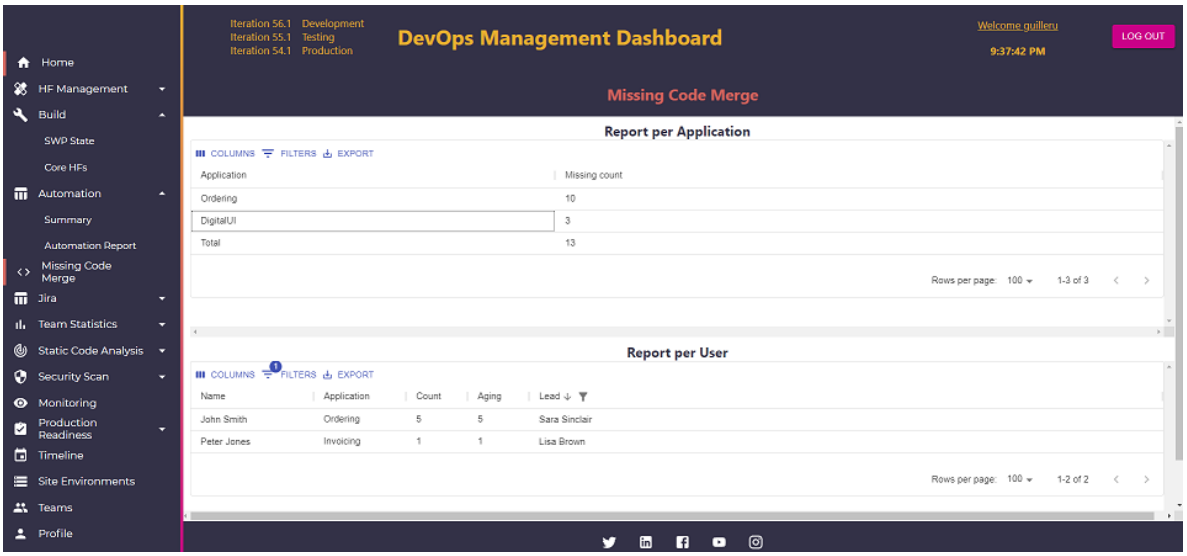

Figura 15 Sección de cambios pendientes de Código

Continuado con la Figura 16, se presenta otra sección, la cual retoma el tema conectividad con aplicaciones de terceros y haciendo referencia a la Tabla 1, la columna de Analizador de Código Estático, en esta sección en concreto, se conecta con el software SonarQube[20], que permite visualizar por aplicación los últimos cambios en el código fuente de las aplicaciones y clasificar por severidad cuando no se cumple con ciertas reglas y estándares en determinado contexto, tecnología o lenguaje de programación, aquí también brinda un acceso directo a la aplicación de SonarQube para que el involucrado puede ver más detalle y poder corregir el incidente, con lo cual la calidad se incrementa y se reducen riesgos potenciales en las distintas aplicaciones.

[20] https://www.sonarqube.org/

Figura 16 Sección de Integración con Sonar

Continuando con la integración de aplicaciones de terceros, en la *Figura 17*, se presenta integración con una aplicación de Ciclo de Vida del Proyecto (Ágil), en referencia a la *Tabla 5*, en este caso concreto, de Jira[21] y se puede observar un listado de *unidades de trabajo*, desde una perspectiva ágil, las cuales pertenecen a una iteración en particular y se tienen que completar en dicha iteración, otro aspecto importante a considerar en la aplicación y que es expuesto en esta imagen, es la presentación de datos, como se resaltan datos relevantes en la aplicación, como la fecha de entrega; también a nivel de usabilidad en la interfase web, se presenta la posibilidad de ocultar columnas, aplicar filtros, paginación, búsquedas, ordenamientos de acuerdo a las columnas, así como exportar a un formato *csv*[22] para una manipulación más a detalle en otras herramientas de software como Excel

[21] https://www.atlassian.com/es/software/jira
[22] https://es.wikipedia.org/wiki/Valores_separados_por_comas

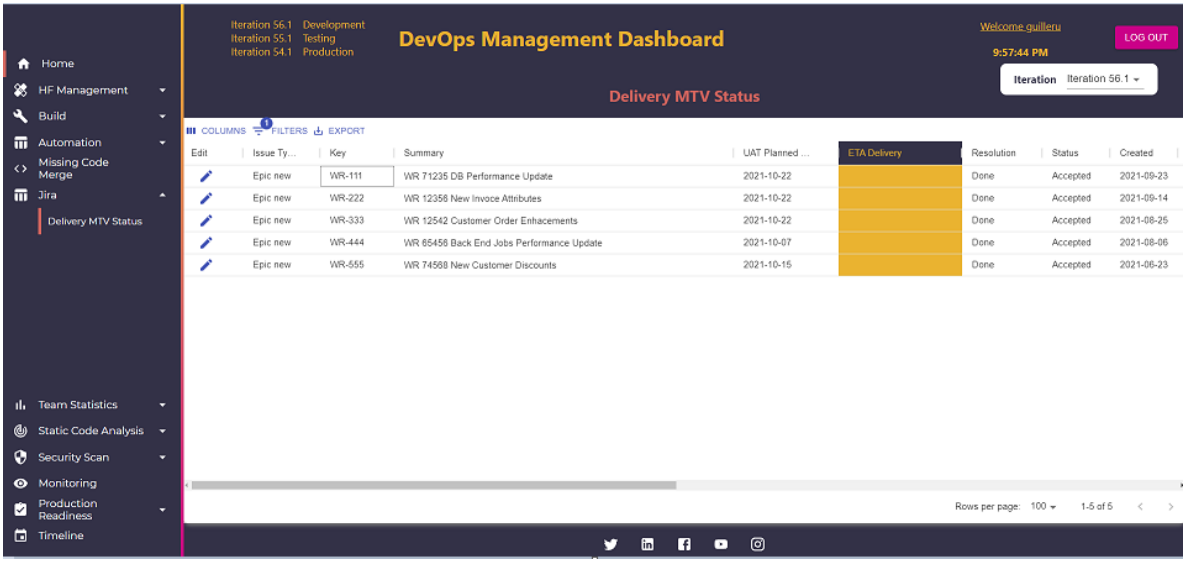

Figura 17 Sección de Integración con Jira

Y para concluir con la presentación de este prototipo funcional del ***DOMD*** se presenta una sección de administración, *Figura 18*, aquí solo roles privilegiados podrán acceder, su propósito es mantener diferentes aspectos de la aplicación ligadas a DevOps y paradigmas ágiles, como la definición de iteraciones, agregar usuarios (que potencialmente se podrían conectar a través de Directorios Activos), la administración de Catálogos como los roles de usuarios, notificaciones por correo electrónico, calendarización de eventos, etc.

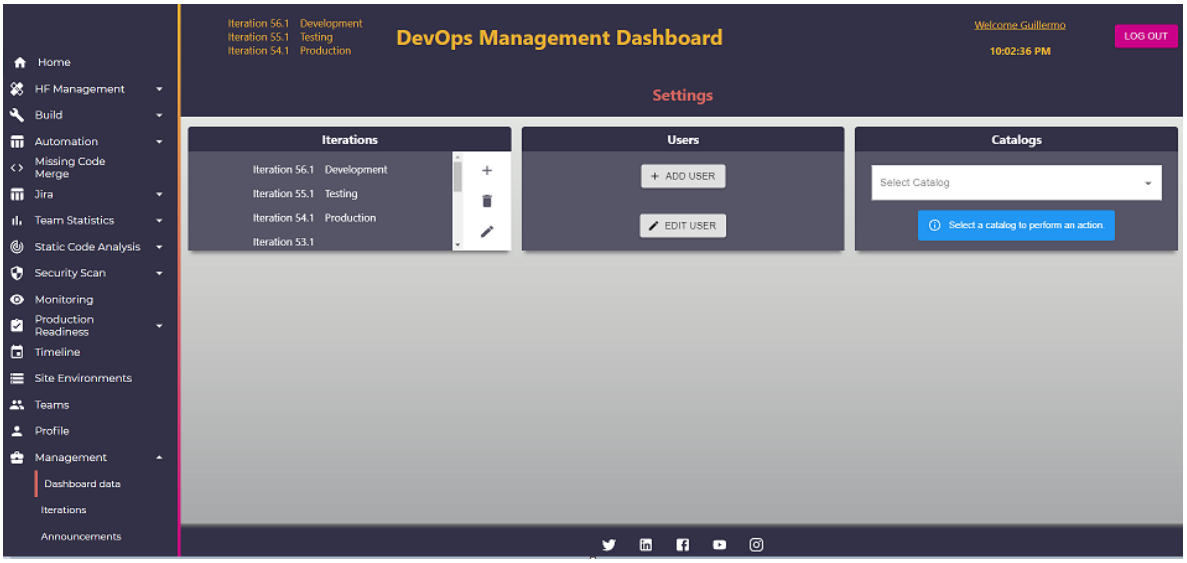

Figura 18 Sección de Administración del DevOps Management Dashboard

6. CONCLUSIONES

6.1. Conclusiones

El alcance de este proyecto consistía en analizar, documentar, modelar y definir la arquitectura de un sistema de información con el cual se pudieran gestionar proyectos de desarrollo de software que apliquen DevOps y metodologías ágiles en el día a día; considerando todos los retos de integración de herramientas, procesos, mecanismos de comunicación, entre otros rubros ya descritos puntualmente en este documento. Se logró completar el objetivo, se plasmó el diseño, se modeló la base de datos, se acotaron los roles de los stakeholders, los casos de uso y se definió el modelo general junto con la arquitectura de la aplicación. Todo esto, sienta las bases para implementar y extender este proyecto, incluso se logró crear un prototipo funcional para ejemplificar, de forma clara, la interfaz de la plataforma y un vistazo general en su utilización.

Se espera que este trabajo sirva a otros profesionistas del área de software a enriquecer su conocimiento en DevOps y Agile, utilizar y entender el potencial de este tipo de herramientas, las cuales utilizadas de manera correcta pueden contribuir a que las organizaciones se vuelvan más eficientes, desarrollen sus capacidades, se integren los equipos y aplicaciones, generando una sinergia y calidad en el software entregado.

A nivel personal, con base a mi experiencia profesional y al rol de *Release Development Manager* que tengo en la actualidad, puedo aseverar que el ***DOMD*** funciona, ayuda a transparentar procesos y herramientas de software, las cuales son utilizadas todos los días por equipos grandes y complejos con diferentes roles.

El conocimiento adquirido en el área de DevOps y metodologías ágiles fue muy relevante y enriquecedor y sumado con las revisiones y retroalimentaciones por parte de mi asesor de grado, completa este aprendizaje, el cual me enriquece para ser una mejor persona y profesionista y continúe aportando a la sociedad con respeto, honestidad y compromiso.

6.2. Trabajo Futuro

Con la primera versión del ***DOMD***, se cumple el objetivo de integrar diferentes aplicaciones, centralizar procesos y datos, pero siempre hay aspectos de mejora a ser considerados.

En un futuro convendría hacer una versión con microservicios, desacoplando más cada servicio y las interacciones entre ellos, permitiendo mayor autonomía y escalabilidad, esto se adaptaría a la arquitectura de la aplicación y el propósito de integrar de la mayor cantidad de aplicaciones *Third Party* con menos complejidad y reduciendo los tiempos de implementación.

Otro aspecto, que podría considerarse, es la implementación de monitoreo de herramientas de *Despliegue Continuo*, con lo cual poder cerrar aún más la brecha del flujo completo de *DevOps* y tener más datos en tiempo real que sirvan como mecanismos de monitoreo y control en cualquier proyecto.

La retroalimentación por parte de los usuarios es muy importante, conviene enfocarse en analizar y entender aún más procesos que pudieran ser integrados en la herramienta, ayudando así a más personas a desempeñar su trabajo de una manera más clara, precisa y eficiente.

Por último, pero no menos importante, convendría implementar algún componente o módulo que tome ventaja de Big Data, para analizar los grandes volúmenes de datos y generar información valiosa que ayude en la toma de decisiones y estrategia en los proyectos con DevOps y metodologías ágiles.

BIBLIOGRAFÍA

[1] “Agile Project Success Rates 2X Higher Than Waterfall”, *Vitality Chicago Inc.*, Apr. 01, 2019. https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/

[2] C. Wang y C. Liu, *Adopting DevOps in Agile Challenges and Solutions*, 2018.

[3] “Agile Methodology: An Overview”, *Zenkit*, mar. 02, 2018. https://zenkit.com/en/blog/agile-methodology-an-overview/

[4] Datadog, “Azure DevOps Monitoring | Datadog”, *Azure DevOps Monitoring*, 2021 https://www.datadoghq.com/dg/monitor/azure-devops-monitoring/

[5] S. Rossel, *Continuous Integration, Delivery, and Deployment*. Packt Publishing, 2017.

[6] K. Bacha, *DevOps compliant guidelines for project inception elaboration Phases*, 2019.

[7] “DevOps dashboard and view | ServiceNow Docs”. https://docs.servicenow.com/bundle/orlando-devops/page/product/enterprise-dev-ops/concept/insights-dashboard-dev-ops.html , Nov 14, 2019

[8] “DevOps Dashboards”, *Sisense*. https://www.sisense.com/dashboard-examples/devops/ , 2021.

[9] “Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch”, *InfoQ*. https://www.infoq.com/articles/standish-chaos-2015/

[10] F. Paul y pwpadmin, “The Incredible True Story of How DevOps Got Its Name”, *New Relic Blog*, may 16, 2014. https://blog.newrelic.com/engineering/devops-name/

[11] “The Origins of DevOps: What’s in a Name?”, *DevOps.com*, ene. 25, 2018. https://devops.com/the-origins-of-devops-whats-in-a-name/

[12] KathrynEE, “Understand dashboards, charts, reports & widgets - Azure DevOps”. https://docs.microsoft.com/en-us/azure/devops/report/dashboards/overview , 2021.

[13] 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr. https://www.youtube.com/watch?v=LdOe18KhtT4, Jun 29, 2009.

[14] “What is Build Automation / Automated Build? | Agile Alliance”. https://www.agilealliance.org/glossary/automated-build/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_glossary~'aa_organizations~'aa_research_paper~'aa_video)~tags~(~'automated*20build))~searchTerm~'~sort~false~sortDirection~'asc~page~1) , 2021