IMPLEMENTACIÓN DE UN GOBIERNO AUTOGESTIONADO PARA EL REPOSITORIO DE SERVICIOS COMUNES DEL LABORATORIO DIGITAL DEL BANCO DE OCCIDENTE BAJO LAS PRÁCTICAS INNERSOURCE COMO ESTRATEGIA DE REUTILIZACIÓN DE SOFTWARE

JESÚS ALEXANDER ANDRADE SÁNCHEZ

Nota de Aceptación

Certificamos que el presente Trabajo de Grado Satisface, en alcances y calidad, todos los requisitos que demanda un Trabajo de Grado de Maestría.

JUAN CARLOS MARTÍNEZ ARIAS
Director

LUISA FERNANDA RINCÓN PÉREZ
Jurado 1

FELIPE MEJÍA VILLEGAS
Jurado 2

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister en Ingeniería de Software.

HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Santiago de Cali, 31 de marzo de 2022

## Acta de Correcciones al Documento de Trabajo de Grado

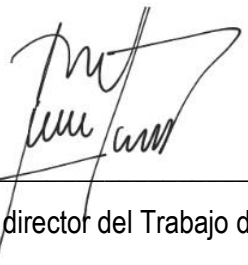
**Santiago de Cali, 31 de marzo de 2022**

**Autor: Jesús Alexander Andrade Sánchez**

**Título del Trabajo de Grado: "IMPLEMENTACIÓN DE UN GOBIERNO AUTOGESTIONADO PARA EL REPOSITORIO DE SERVICIOS COMUNES DEL LABORATORIO DIGITAL DEL BANCO DE OCCIDENTE BAJO LAS PRÁCTICAS INNERSOURCE COMO ESTRATEGIA DE REUTILIZACIÓN DE SOFTWARE"**

**Director: Juan Carlos Martínez Arias**

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.



Firma del director del Trabajo de Grado


# IMPLEMENTACIÓN DE UN GOBIERNO AUTOGESTIONADO PARA EL REPOSITORIO DE SERVICIOS COMUNES DEL LABORATORIO DIGITAL DEL BANCO DE OCCIDENTE BAJO LAS PRÁCTICAS INNERSOURCE COMO ESTRATEGIA DE REUTILIZACIÓN DE SOFTWARE

*Jesús Alexander Andrade Sánchez*
*Código. 00007939087*

*Trabajo de grado para optar al título de*
*Magister en Ingeniería de Software*

Director
JUAN CARLOS MARTÍNEZ ARIAS

FACULTAD DE INGENIERÍA Y CIENCIAS
MAESTRÍA EN INGENIERÍA DE SOFTWARE
SANTIAGO DE CALI, FEBRERO 28 DE 2022

FICHA RESUMEN
TRABAJO DE GRADO DE MAESTRÍA

TITULO: "IMPLEMENTACIÓN DE UN GOBIERNO AUTOGESTIONADO PARA EL REPOSITORIO DE SERVICIOS COMUNES DEL LABORATORIO DIGITAL DEL BANCO DE OCCIDENTE BAJO LAS PRÁCTICAS INNERSOURCE COMO ESTRATEGIA DE REUTILIZACIÓN DE SOFTWARE"

1. ÉNFASIS: Ingeniería de software.
2. ÁREA DE INVESTIGACIÓN: Reutilización de código de software.
3. ESTUDIANTE: Jesús Alexander Andrade Sánchez.
4. CORREO ELECTRÓNICO: alexander.andrade@gmail.com
5. DIRECTOR: Juan Carlos Martínez Arias.
6. CODIRECTOR(ES): N/A.
7. GRUPO QUE LO AVALA: N/A.
8. OTROS GRUPOS: N/A.
9. PALABRAS CLAVE: InnerSource, Ingeniería de Software, desarrollo de software basado en componentes reutilizables, reusabilidad, activo de software, reutilización de software, línea de productos.
10. CÓDIGOS UNESCO CIENCIA Y TECNOLOGÍA: 1203,1203.99.
11. FECHA DE INICIO: 01 de Jul de 2021 DURACIÓN ESTIMADA: 6 Meses.
12. RESUMEN (máximo una página).

**RESUMEN**

Por medio de este proyecto se diseñó a la medida un gobierno autogestionado para el Repositorio de Servicios Comunes (RSC) del Laboratorio Digital del Banco de Occidente, quienes han definido la reutilización de activos de software como la estrategia para acelerar su velocidad de desarrollo con el fin de reducir el tiempo requerido para sacar un producto digital al mercado (*time-to-market*), pero antes de este Gobierno pero carecían de estrategias que fomentaran la reutilización, mejora y creación de nuevos componentes. Se realizó una revisión bibliográfica para entender como empresas referentes de desarrollo de software resuelven esta situación, y se encontró que usan en

gran medida las prácticas InnerSource (ISS). Se diagnosticó la situación del Laboratorio frente a las ISS, se diseñó el gobierno a la medida, que luego se implementó y esta implementación se evaluó mediante una encuesta. El presente trabajo hace aportes sobre falencias encontradas en la revisión literaria de ISS y anexa una copia del Gobierno desarrollado que puede aplicarse en otras organizaciones, siendo esta monografía el primer trabajo académico redactado, a saber, en español, y en Latinoamérica, sobre ISS y tiene el potencial de convertirse en un referente para la aplicación e investigación de estas prácticas en el mundo de habla hispana.

**ABSTRACT**

Through this project, a personalized self-managed government was designed for the Common Services Repository (RSC) of Banco de Occidente's Digital Laboratory, who have defined the reuse of software assets as the strategy to accelerate their development speed to reduce the time needed to bring a digital product to market (time-to-market), but before this Government they lacked strategies that encouraged the reuse, improvement, and creation of new components. A literature review was conducted to understand how major software development companies deal with this situation and found that they heavily use InnerSource (ISS) practices. The situation of the Laboratory in relation to the ISS was diagnosed, the government was designed to measure, which was then implemented, and this implementation was evaluated through a survey. This work makes contributions on the shortcomings found in the literature review about ISS and attaches a copy of the developed Government that can be applied in other organizations, this monograph being the first written academic work, as is know at this time, in Spanish, and in Latin America, on ISS and It has the potential to become a referrer for the application and research of these practices in the Spanish-speaking world.

# TABLA DE CONTENIDO

# LISTA DE FIGURAS

# LISTA DE TABLAS

# GLOSARIO

**Activo (reutilizable) de software**: Es un producto diseñado expresamente para emplearse de forma recurrente en el desarrollo de muchos sistemas y aplicaciones. Ejemplos de activos reutilizables son: algoritmos, patrones de diseño, esquemas de base de datos, arquitecturas de software, especificaciones de requerimientos, de diseño y de prueba, entre otros (Montilva et al., 2003, pág. 2).

**API**: Una interfaz de programación de aplicaciones (API) es una conexión entre computadoras o entre programas de computadora. Es un tipo de interfaz de software que ofrece un servicio a otras piezas de software (Reddy, 2011, pág. 1).

**API Gateway**: De acuerdo con Amazon AWS, es un servicio totalmente administrado que facilita a los desarrolladores crear, publicar, mantener, monitorear y asegurar API a cualquier escala.

**Componente (reutilizable de software)**: De acuerdo con (Sommerville, 2011, pág. 453) los componentes son abstracciones de alto nivel en comparación con los objetos y se definen mediante sus interfaces. Por lo general, son más grandes que los objetos individuales y todos los detalles de implementación se ocultan a otros componentes. Los servicios o microservicios son un buen ejemplo de este término.

***Confluence***: De acuerdo Atlassian, su fabricante, *Confluence* es un software de colaboración en equipo donde se puede crear y compartir documentación y discusiones en un solo sitio. Para este trabajo de grado, es el sitio donde se documenta la información de los proyectos de la organización.

**ISS**: Las prácticas *InnerSource Software* (ISS) son formalmente definidas como "El uso de principios y prácticas de Desarrollo de Software Libre (OSSD: *Open Sourced Software Development*) dentro de los confines de una empresa" (ISC, 2021, pág. 3).

***Issues***: "propuestas", "solicitudes de características" (nuevas funcionalidades) o "reportes de *bugs"* (Mittman, 2018, pág. 17).

***Mock***: Objetos o servicios simulados.

**Open API**: La especificación OpenAPI (OAS: *Open* API *Specification*) se define a sí misma como una interfaz estándar independiente del lenguaje para las API de tipo RESTful que

permite que tanto los humanos como las computadoras descubran y comprendan las capacidades del servicio sin acceso al código fuente, documentación o mediante la inspección del tráfico de la red. Cuando se define correctamente, un consumidor puede comprender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación.

**RSC**: Repositorio de Servicios Comunes.

Swagger: También conocido como “Open API”. La especificación OpenAPI (OAS: *Open* API *Specification*) se define a sí misma como una interfaz estándar independiente del lenguaje para las API de tipo RESTful que permite que tanto los humanos como las computadoras descubran y comprendan las capacidades del servicio sin acceso al código fuente, documentación o mediante la inspección del tráfico de la red. Cuando se define correctamente, un consumidor puede comprender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación.

# INTRODUCCIÓN

Por medio de este proyecto se diseñó a la medida un gobierno autogestionado para el Repositorio de Servicios Comunes (RSC) del Laboratorio Digital del Banco de Occidente, quienes han definido la reutilización de activos de software como la estrategia para acelerar su velocidad de desarrollo con el fin de reducir el tiempo requerido para sacar un producto digital al mercado (*time-to-market*). Sin embargo, como muchas organizaciones, esta empresa ha aplicado muy bien las técnicas de reutilización disponibles, pero carecía de estrategias culturales que le permitieran fomentaran la reutilización de los componentes existentes y la participación de todo el personal para la mejora y creación de nuevos componentes.

Este trabajo se inició buscando la solución a este problema con base a dos premisas determinadas por la organización: 1) que fuera autogestionada por cada integrante del Laboratorio y 2) que se basara en técnicas utilizadas por referentes del desarrollo de Software. Se realizó una revisión bibliográfica encontrando que las Prácticas InnerSource (ISS) cumplían a cabalidad con lo requerido. Luego se diagnosticó la situación a través de entrevistas y una encuesta que revelaron realidades desconocidas hasta el momento por los Administradores del Laboratorio, como que la reutilización estaba muy limitada dado que el RSC, así como su concepto, era desconocido para la mayoría de los colaboradores; y que, a pesar de ser reconocidos por su cultura de apertura, hay mucho hermetismo dentro de las células. Posterior a esto, se diseñó el gobierno a la medida, que luego se implementó en todos los activos del RSC y cuya implementación se evaluó mediante una nueva encuesta.

Como principales conclusiones podemos encontrar que la reutilización de software, y la consecuente aceleración del desarrollo, sigue siendo una actividad que muy pocas empresas logran aprovechar, ya que históricamente solo se han implementado las cuestiones técnicas y poco las culturales necesarias para lograr esta ventaja. Aunque también existen muchas herramientas de trabajo colaborativo, estas de por sí solas no garantizan el nivel de colaboración necesario para crear una cultura abierta que fomente la reutilización. La literatura más referenciada sobre ISS poco habla de las actividades previas a la implementación, además de que, aunque considera el trabajo remoto, el equipo de implementación y los decisores siempre se han encontrado co-localizados; este trabajo realiza aportes a estas situaciones al

documentar como se resolvieron y presentar todo el Gobierno en un anexo que se puede aplicar en otras organizaciones.

El trabajo de grado culmina con la redacción de la presente monografía que es el primer texto académico redactado, a saber, en español, y en Latinoamérica, sobre ISS y tiene el potencial de convertirse en un referente para la aplicación e investigación de estas prácticas en el mundo de habla hispana.

# 1 DEFINICIÓN DEL PROBLEMA

## 1.1 PLANTEAMIENTO DEL PROBLEMA

Uno de los principales desafíos que enfrentan las empresas es aumentar la velocidad del desarrollo de software al encontrar formas más eficientes para que los desarrolladores colaboren entre sí, compartan conocimientos, reutilicen el código desarrollado y reduzcan los comportamientos “tribales”. En la última década, la Transformación Digital ha creado un sentido de urgencia para que todo tipo de empresas desarrollen estas habilidades. Sumado a esto, la irrupción de la COVID-19 y los confinamientos impuestos a nivel mundial durante los últimos años, han llevado a las empresas a acelerar décadas de planes de digitalización para dar respuesta a esta nueva realidad de interactuar con los clientes sin contacto físico y han surgido nuevas necesidades en la dinámica de los equipos de desarrollo de software en entornos no co-localizados.

Para transformar su negocio, Banco de Occidente cuenta desde 2019 con un Laboratorio Digital que desarrolla software como experimentos para resolver las necesidades de sus clientes o incluso intentar anticiparlas. Este laboratorio está compuesto por diferentes "células" de desarrolladores, cada una a cargo de un producto que responde a necesidades únicas. Desde 2020, el Laboratorio trabaja de forma 100% remota, lo que añade una nueva capa de complejidad a sus procesos y cultura que se diseñaron para el trabajo en sitio.

Todos los desarrollos del Laboratorio se llevan a cabo bajo el enfoque de microservicios, como componentes reutilizables para lograr los objetivos de un producto en particular. Esto ha sido visto por las directivas como una oportunidad para reutilizar software bajo el paradigma de Desarrollo Basado en Componentes que, según (Sommerville, 2011, pág. 426) desde principios de la década de 2000 ha sido la respuesta a las crecientes demandas para reducir los costos de producción y mantenimiento del software, y aumentar la calidad y velocidad de entrega de los sistemas. Para ello, el Laboratorio ha habilitado un repositorio de códigos denominado de Servicios Comunes (RSC) o “Blue Bank”, donde las células pueden localizar desarrollos que puedan ser de utilidad para otros equipos de trabajo. Sin embargo, el Laboratorio, al igual que el 59 % de los encuestados por (Capilla et al., 2019, pág. 11), no tiene prácticas de reutilización definidas, por lo que el RSC no promovía ni aseguraba, entre otros, la confianza en el código depositado debido a situaciones como la escasa o nula documentación, poca claridad sobre responsabilidades o la baja trazabilidad de impactos que pueden haber en otros productos al hacer contribuciones al código; lo que se traducía en esfuerzos aislados para hacer soluciones que ya existían (“reinventar la rueda”) con su consecuente afectación a los tiempos de salida al mercado (*time-to-market*).

Esta situación, conocida por las directivas del Laboratorio, dio comienzo a este trabajo de grado y de cuyo primer acercamiento surgieron una serie de interrogantes que orientaron la búsqueda de una solución:

- ¿Cómo **garantizar que los desarrolladores contribuyan** al repositorio de Servicios Comunes (RSC)?
- ¿Cuál era la mejor manera de **estandarizar y obtener consistencia** en la documentación y códigos almacenados en este repositorio?
- ¿Cómo se aseguraría de que el RSC **genere un alto nivel de confianza** para que las células aprovechen estas soluciones en lugar de emprender esfuerzos individuales?
- ¿Cómo se **facilitaría el aporte de mejoras** al RSC por parte de cualquier desarrollador del Laboratorio independientemente de la célula a la que perteneciera?
- ¿Cómo se aseguraría que quienes aportaran tuvieran la **visión sobre el impacto que tendrán sus mejoras en otros productos**?
- ¿Cómo se lograría que todas las Células se **coordinaran y auditaran el repositorio de forma autónoma**, es decir, sin que existiera una estructura de administración distinta a los propios desarrolladores?
- ¿Qué **buenas prácticas de coordinación se podrían extraer de proyectos con miles de colaboradores** como Linux?
- Y finalmente, ¿**cómo resolvían estas inquietudes los grandes desarrolladores de software** como Google, Microsoft, entre otros?

El Laboratorio restringió la búsqueda a soluciones utilizadas en referentes de la industria y no se equivocaba en querer imitar a otros, ya que la imitación de métodos que han demostrado ser exitosos es una práctica de mejora común entre las empresas que desarrollan software (Stol et al., 2011, pág. 1). En este orden de ideas, la revisión bibliográfica determinó que los referentes citados resuelven estas situaciones a través de las prácticas InnerSource (ISS). Estas serán la base sobre la cual el Laboratorio puede establecer una gobernanza sobre el RSC que fomente de manera autogestionada el intercambio no solo de microservicios (componentes de software), sino también de los activos de software relacionados con estos como el código, arquitecturas, especificaciones y diseños, y contribuyendo a la distribución de los conocimientos adquiridos durante la investigación previa al desarrollo, de las decisiones técnicas tomadas y de las buenas prácticas entre toda la plantilla de desarrolladores.

Por todo lo anterior, se estableció un gobierno para el repositorio de Servicios Comunes bajo las prácticas ISS como estrategia de reutilización, que se adaptó a las particularidades del Laboratorio Digital del Banco de Occidente; siendo éste el primer trabajo académico, a saber, en español, y en Latinoamérica, realizado sobre este tema y que tiene el potencial de llegar a convertirse en un referente para la aplicación e investigación de estas prácticas en el mundo de habla hispana.

## 1.2 FORMULACIÓN DEL PROBLEMA

De acuerdo con el planteamiento del problema, se realizó la siguiente formulación:

> ¿Cómo implementar un gobierno autogestionado para el repositorio de Servicios Comunes del Laboratorio Digital del Banco de Occidente bajo las prácticas InnerSource como estrategia de Reutilización de Software?

# 2 OBJETIVOS DEL PROYECTO

## 2.1 OBJETIVO GENERAL

Implementar un gobierno autogestionado para el repositorio de Servicios Comunes (RSC) del Laboratorio Digital del Banco de Occidente bajo las prácticas InnerSource (ISS) como estrategia de Reutilización de Software.

## 2.2 OBJETIVOS ESPECÍFICOS

1. Diagnosticar los inconvenientes actuales relacionados con la gestión del RSC que limitan la reutilización de los componentes de software contenidos en este repositorio.
2. Diseñar un gobierno autogestionado para el RSC bajo las prácticas ISS que resuelva los problemas diagnosticados con respecto a la gestión de este repositorio con el fin de reutilizar los componentes de software que contiene.
3. Implementar el Gobierno ISS diseñado en todos los componentes de software del RSC.
4. Validar que la implementación del Gobierno ISS resuelva los problemas diagnosticados en materia de gestión del RSC que limitan la reutilización de los componentes de software contenidos en este repositorio.

# 3 RESULTADOS ESPERADOS

Los resultados, en términos de entregables, se enumeran a continuación:

- Para el Laboratorio Digital del Banco de Occidente:
  - Sitio InnerSource (ISS) que mantiene un inventario de los componentes de software migrados al Gobierno ISS.
  - Migración al Gobierno ISS de todos los componentes de software del repositorio de Servicios Comunes (RSC) que existan en el momento de la realización de este proyecto.
  - Instructivos editables del Gobierno ISS.
- Para la Universidad Pontificia Javeriana y comunidad académica:
  - Monografía con documentación del proceso llevado a cabo para realizar el trabajo de grado. Este contiene copias de los documentos entregados a la organización donde se ofuscará la información privada de la empresa, así como recomendaciones para futuros trabajos de investigación sobre el marco teórico de esta monografía o del proyecto implementado en la organización.

# 4 ALCANCE

El alcance del presente trabajo de grado, que está delimitado por los OBJETIVOS DEL PROYECTO y los RESULTADOS ESPERADOS, fue implementar un gobierno autogestionado para el repositorio de Servicios Comunes del Laboratorio Digital del Banco de Occidente bajo las prácticas InnerSource (ISS) como estrategia de Reutilización de Software. Como delimitaciones (alcance) se tiene que:

- La solución no incurrió en costo alguno, ni requirió la adquisición o desarrollo de Software, ya que aprovechó las herramientas existentes en el Laboratorio.
- La evaluación del proyecto se realizó mediante encuestas dirigidas a todos los desarrolladores del Laboratorio para identificar si se resolvieron los problemas del repositorio de Servicios Comunes (RSC) que limitaban la reutilización de los componentes de software depositados en este repositorio. Dentro de la evaluación no se contabilizaron el número de aportes o aumento de contribuyentes tras la implementación del Gobierno ISS, ya que esto no podía ser influenciado por este trabajo de grado, pues dependía de la necesidad de reutilización o del surgimiento de nuevas iniciativas comunes.

## 4.1 EXCLUSIONES

Quedaron fuera del alcance de este trabajo de grado:

- Implementaciones fuera del contexto específico del trabajo de grado. No obstante, durante el desarrollo de este proyecto o después de su finalización, el Gobierno ISS puede implementarse fuera del contexto original sin que ello implique participación, responsabilidad, notificación a, o soporte de este trabajo de grado.
- El soporte de la solución, su continuidad o evolución, tras la entrega del trabajo de grado.
- El desarrollo de automatizaciones como informes de uso del repositorio de Servicios Comunes (RSC) o buenas prácticas utilizadas en general, por proyecto o por desarrollador.
- La gestión o gobierno de activos de software que se encontraban en el RSC y que no fue posible establecer una célula responsable o usuaria.
- Cualquier otra actividad, tema o elemento no mencionado en el alcance.

# 5 MARCO TEÓRICO DE REFERENCIA Y ANTECEDENTES

A lo largo de esta sección se muestra los conceptos básicos de la Reutilización de Software como estrategia para aumentar la velocidad de desarrollo y reducir costos. También se encontró que es común entre muchos referentes del desarrollo de software el uso de las prácticas InnerSource (ISS) para crear el ambiente propicio para favorecer la reutilización de software.

Tras establecer este marco teórico, se realizó un recuento de los trabajos relacionados que aportan información relevante para el presente trabajo. También se analizó el estado del arte, es decir, cuál es la situación actual de la Reutilización de Software y de las prácticas ISS.

## 5.1 REUTILIZACIÓN DE SOFTWARE

De acuerdo a la IEEE, "la reutilización de software implica capitalizar el software y los sistemas existentes para crear nuevos productos" (IEEE, 2010). El término Reutilización de Software se acuñó en la primera Conferencia Internacional sobre Ingeniería de Software (ICSE) en 1968 (Capilla et al., 2019, pág. 1). Desde entonces, academia e industria han estudiado la reutilización de software para aumentar la productividad y reducir costos (Barros-Justo et al., 2019, pág. 1), aumentando cada año el número de publicaciones por lo que se considera que sigue siendo un tema de interés en la Ingeniería de Software. Además, la reutilización de software puede describir tanto el uso de componentes preexistentes para desarrollar nuevas aplicaciones como la mejora de un producto antiguo (sistemas heredados o legados / *legacy*) (Barros-Justo et al., 2019, pág. 2).

La Tabla 1 relaciona una serie de términos, enfoques y ámbitos de aplicación de la reutilización de software, de acuerdo a la revisión de literatura realizada por (Barros-Justo et al., 2019, págs. 1 y 2).

Tabla 1. Clasificación de la Reutilización de Software

| | |
|---|---|
| Términos relacionados | 1. Desarrollo basado en componentes, *Component Based Development* (CBD),<br>2. Líneas de productos de software, *Software Product Lines* (SPL),<br>3. Desarrollo dirigido por modelos, *Model Driven Development* (MDD),<br>4. Ingeniería de dominio (o análisis de dominio),<br>5. Producto listo-para-usar, *Commercial-Of-The-Shelf* (COTS),<br>6. Entre otros. |
| Enfoques principales | 7. Con reutilización: desarrollo utilizando componentes preexistentes.<br>8. Para la reutilización: desarrollo de componentes reutilizables. |
| Ámbito de aplicación | 9. Reutilización vertical: reutilización de software en un dominio de aplicación determinado.<br>10. Reutilización horizontal: reutilización de componentes en varios dominios de aplicación. |

Fuente: Propia, basada en (Barros-Justo et al., 2019, págs. 1 y 2)

## 5.2 PRÁCTICAS INNERSOURCE

Las prácticas InnerSource (ISS) son formalmente definidas como "El uso de principios y prácticas de Desarrollo de Software Libre (OSSD: *Open Sourced Software Development*) dentro de los confines de una empresa" (ISC, 2021, pág. 3). *Inner Sourced* Software (ISS) promete resolver los problemas del desarrollo de software tradicional facilitando la reutilización del software y permitiendo que las partes dentro de una organización colaboren más allá de los límites internos de la organización (Capraro, 2020 pág. V). Dentro de los casos de éxito de esta práctica se encuentran Google, Uber, SAP, PayPal, Nike, American Airlines, BBC, Europace, Robert Bosch GmbH, DB Systel, Elbit Systems, Entelgy, Zylk, Bitergia, Hewlett-Packard, Alcatel-Lucent, Philips Healthcare, IBM, Nokia o el Laboratorio de Propulsión Jet (JPL) de la NASA (ISC, 2021) (Edison et al., 2018, pág. 1) (Stol & Fitzgerald, 2014, pág. 1) (Mittman, 2018).

Para llevar acabo un diagnóstico del estado del uso de ISS se analizaron artículos, tesis y libros sobre el tema, teniendo en cuenta criterios como su relevancia en cuanto a citaciones, así como la relación de casos de éxito en empresas reconocidas. A partir de este análisis, se determinó el impacto de las ISS en las organizaciones, sus factores de éxito o fracaso, de tal forma que permitan plantear estrategias para adaptar el marco dentro una empresa local. Este análisis también sirve como un punto de partida para nuevas investigaciones y sus posibilidades de adaptación a otras empresas.

Para realizar la investigación se hicieron búsquedas inicialmente en Google sobre prácticas de colaboración de software en empresas como Google o Microsoft. Pronto, las

búsquedas dirigieron hacia las prácticas ISS y con esta información se procedió a buscar en Google Scholar con los términos "*Inner Source*" y "Software", filtrando resultados entre 2016 y 2021. El filtro por idioma español no arrojó resultados. Luego se hizo esta misma búsqueda en la editorial tecnológica O'Reilly donde se encontraron varios libros al respecto. De los resultados arrojados por la búsqueda, se eligieron 5 artículos, 1 tesis de doctorado y 3 libros, que resultan ser los mas referenciados sobre el tema.

### 5.2.1 Aspectos generales

En las empresas que desarrollan software se presentan diferentes problemas a la hora de coordinar el conocimiento adquirido por las células de desarrolladores, así como la reutilización del software desarrollado para apalancar otros desarrollos. Las ISS se presentan como una forma de crear comunidades de desarrolladores, como las de OSSD, dentro de las instalaciones de la organización habilitando que cada desarrollador pueda contribuir, retroalimentar y más, a los repositorios de los proyectos de la organización (Mittman, 2018, págs. 5-6).

De acuerdo a Tim O'Reilly[1] "El éxito del Software Libre que tiene más que ver con sus capacidades de colaborar que con el simple hecho de programar" (O'Reilly, 2000) ha hecho que las empresas se interesen en adoptar estas prácticas para sus procesos de software; fenómeno que O'Reilly acuñó como *Inner Source* (ISS) (O'Reilly, 2000).

La implementación de ISS cuenta con el amplio apoyo de las investigaciones académicas, casos de éxitos en grandes compañías que desarrollan software para uso interno y externo, y la comunidad Inner Source Commons (ISC), fundada por PayPal en 2015 como un foro abierto a la discusión y mejora de las prácticas (Mittman, 2018, pág. 37).

### 5.2.2 Las ventajas del OSSD deseadas por las empresas

Intentando averiguar sobre las buenas prácticas para el desarrollo de software común en las organizaciones, se hizo evidente que era necesario antes entender las buenas prácticas en el desarrollo de los proyectos de código abierto (OSS: *Open-Sourced Software*), entre las que se encuentran:

- La colaboración abierta fomenta una mayor contribución y por ende la innovación (Mittman, 2018, pág. 8). El acceso universal a todos los artefactos de desarrollo, como el

[1] Tim O'Reilly, fundador de la editorial tecnológica O'Reilly.

código y la documentación, permite que cualquiera pueda inspeccionar y enviar contribuciones (Stol & Fitzgerald, 2014, pág. 1) mientras el proyecto y la organización se benefician de la creatividad y diversidad de perspectivas de toda la plantilla de desarrolladores (Oram, 2021, pág. 8).

- Aumenta la calidad de los desarrollos mediante la revisión por pares independientes de las contribuciones de otros en la "comunidad" de desarrolladores (Mittman, 2018, pág. 8) (Stol & Fitzgerald, 2014, pág. 1). Establecer un sistema formal y objetivo para probar cada contribución, basados en técnicas de pruebas continuas (*continuous testing*), fomenta la confianza entre los desarrolladores y permite determinar cuales de ellos pueden recibir mayores responsabilidades respecto al proyecto (Oram, 2021, págs. 10-11).
- Documentación completa depositada en el repositorio del código (Oram, 2021, pág. 17).
- Comentarios tempranos y lanzamientos frecuentes para mantener vivo un proyecto y mejorarlo rápidamente (Stol & Fitzgerald, 2014, pág. 1).
- Las personas que se encuentran a grandes distancias geográficas, en momentos distintos, pueden trabajar en el mismo código o contribuir con diferentes archivos de código al mismo proyecto (Oram, 2021, pág. 7).
- La comunicación tiende a escribirse y ubicarse en sitios públicos en lugar de compartirse informalmente de boca en boca, lo que proporciona una historia del proyecto, así como oportunidades de aprendizaje para los nuevos miembros del proyecto (Oram, 2021, págs. 7-8) y (Stol & Fitzgerald, 2014, pág. 1).
- Autoselección de tareas para permitir a los desarrolladores identificar partes del software que creen que pueden mejorar o defectos que pueden corregir (Oram, 2021, págs. 7-8) y (Stol & Fitzgerald, 2014, pág. 1).

### 5.2.3 Ventajas de las ISS

Grandes compañías como Google, Uber, SAP, PayPal, Nike, American Airlines, BBC, Europace, Robert Bosch GmbH, DB Systel, Elbit Systems, Entelgy, Zylk, Bitergia, Hewlett-Packard, Alcatel-Lucent, Philips Healthcare, IBM, Nokia o el Laboratorio de Propulsión Jet (JPL) de la NASA (ISC, 2021) (Edison et al., 2018, pág. 1) (Stol & Fitzgerald, 2014, pág. 1) (Mittman, 2018) han implementado el marco ISS con el ánimo de obtener las ventajas de OSS dentro de los desarrollos internos (del inglés "*inner*") y lograr otras ventajas en el ámbito corporativo como:

- Aumento de productividad, eficiencia y efectividad (Capraro, 2020, pág. 162) ya que no siempre los desarrolladores tendrán que iniciar desde cero (Mittman, 2018, pág. 9). La velocidad de desarrollo es potencialmente tan grande como todo el personal de desarrollo de la organización, lo que puede resultar en un tiempo de comercialización más rápido (*time to market*) (Stol & Fitzgerald, 2014, pág. 2) y (Capraro, 2020, pág. 162). El desarrollo se vuelve más rápido a medida que los programadores aprenden a usar pruebas unitarias, cobertura de código e integración continua para eliminar errores en una etapa temprana del desarrollo. Los comentarios escritos intercambiados entre los miembros del equipo,

aunque toman algo de tiempo, se pagan sobradamente al ayudar a los nuevos programadores a aprender el sistema más rápido (Oram, 2021, pág. 8) y ayuda a mover historias que están muy en el futuro del *backlog* (Bonewald, 2017, pág. 33).

- Reducción de costos de desarrollo al aumentar nivel de reutilización de software (Capraro, 2020, pág. 162) a través de productos y componentes de software que están disponibles como proyectos ISS para que los use cualquier persona dentro de una organización (Stol & Fitzgerald, 2014, pág. 1) (Oram, 2021, pág. 8) reduciendo la duplicación y desperdicio de esfuerzos (Mittman, 2018, pág. 9). Esto también logra una reducción de dependencia entre el equipo desarrollador original y sus contribuidores (Capraro, 2020, pág. 162).
- Calidad mejorada mediante el aprovechamiento de la Ley de Linus[2]: "*Dado un número suficientemente elevado de ojos, todos los errores se vuelven obvios"* (Stol & Fitzgerald, 2014, pág. 1). Además, el explicar un tema complejo a varias personas que quieran contribuir permite reconocer posibles mejoras arquitecturales (del software) (Oram, 2021, pág. 19).
- Mayor innovación en el desarrollo de software (Capraro, 2020, pág. 162).
- Mejora en la documentación del código, tanto formalmente (como comentarios y documentación en el código) como informalmente (en listas de discusión). La documentación proporciona un historial del proyecto y ayuda a personas externas a comprenderlo, y comprender las decisiones tras los cambios, lo que aumenta la contribución (Oram, 2021, págs. 8-9).
- Colaboración entre equipos relativamente fluida. El código contribuido rara vez tiene que ser reescrito por el equipo que lo recibe, y no se requieren discusiones a un alto nivel gerencial (Oram, 2021, pág. 8). El análisis del código contribuido por otros ayuda a crear nuevas y sofisticadas habilidades dentro del equipo (Oram, 2021, pág. 20). Los equipos distantes geográficamente se ven mayormente beneficiados (Capraro, 2020 pág. 122).
- Mayor movilidad del personal como efecto de que los desarrolladores se familiaricen con varios proyectos de software, así como con las herramientas utilizadas en un repositorio de plataforma central (Stol & Fitzgerald, 2014, pág. 2).
- Acelera la incorporación (*On-boarding*) de nuevos desarrolladores (Mittman, 2018, pág. 13) al reducir tiempos y costos relacionados (Oram, 2021, págs. 14-15).
- Fortalecimiento de los procesos, la confianza y la alineación mediante la toma de decisiones transparente (Mittman, 2018, pág. 7).
- Rompimiento de silos (Capraro, 2020, pág. 162) como resultado de una cultura de la colaboración, apertura y transparencia (Mittman, 2018, pág. 13), logrando distribuir los costos y riesgos en toda la organización y mayor intercambio de información (Capraro, 2020 pág. 162).
- Apoya a la mejora del clima laboral: aumenta la satisfacción laboral al permitirles a todos aportar, crea sentido de comunidad, ayuda a mejorar las habilidades de todos los contribuidores, a crear reputación, a fomentar la creación de tutores o mentores, a que cada desarrollador persiga sus intereses, a compartir ideas en igualdad de condiciones y

[2] Linus Torvalds, iniciador y líder *kernel* (núcleo) del Sistema Operativo Linux.

aprender más fácilmente de sus compañeros (Mittman, 2018, pág. 11 y 14) y (Capraro, 2020 págs. 21 y 162).

### 5.2.4 Anatomía de un proyecto ISS

#### 5.2.4.1 Elementos claves de las ISS

Son cuatro elementos clave que constituyen ISS (Capraro, 2020, pág. 3):

1. Cultura de colaboración abierta: equidad, meritocracia y autoorganización.
2. Entorno de desarrollo abierto.
3. Comunidades alrededor del software.
4. Al menos un escenario de ISS específico:
    - Reutilización participativa: participar del desarrollo y mantenimiento del desarrollo reutilizado.
    - Autoselección de tareas: desarrollo realizado durante el horario laboral.
    - Voluntariado: desarrollo realizado fuera del horario laboral.
    - Proyectos de desarrollo colaborativo.

#### 5.2.4.2 Modelos de ISS

Se encuentran dos modelos para implementar ISS (Stol et al., 2011, pág. 7):

- Basado en infraestructura: donde la organización pone a disposición las herramientas comunicativas y de gestión de código.
- Basado en proyectos: donde la organización delega la responsabilidad de los “recursos compartidos” a un equipo.

La Tabla 2 muestra las principales diferencias entre ambos modelos:

Tabla 2. Diferencias claves de los modelos de ISS

| Característica | Basado en infraestructura | Basado en proyectos |
|---|---|---|
| Reutilización | Oportunista, ad-hoc. Maximice el número de proyectos a compartir dentro de la organización. | Planificado estratégicamente. Optimización de la reutilización de activos críticos. |
| Soporte | Opcional, difiere según el proyecto y depende del propietario / encargado del mantenimiento y de la actividad de la "comunidad". | El esencial para el éxito de las ISS. |
| Propietario | Creador / propietario individual del proyecto. | Equipo central. |
| Tipos de paquetes de software | Paquetes de software diversos (por ejemplo, utilidades, herramientas, compiladores, *shells*). | Activos críticos (por ejemplo, plataforma de una línea de productos de software). Tecnología primaria, en lugar de herramientas y utilidades. |

Fuente: (Stol et al., 2011, pág. 7).

Tras analizar los modelos de ISS relacionados en la Tabla 2, se puede determinar que estos no son aplicables a todas las situaciones y por eso es necesario que cada implementador evalúe cual se adapta mejor a la organización y los recursos disponibles.

#### 5.2.4.3 Estructura

Muchos proyectos OSSD siguen una estructura organizativa similar, una que muchas empresas pueden tener en cuenta al configurar equipos multifuncionales para los proyectos de ISS (Mittman, 2018, pág. 15):

- Mantenedores: Colaboradores que son responsables de impulsar la visión y gestionar los aspectos organizativos del proyecto. No son necesariamente los propietarios o autores originales del código. Entre estos se encuentran: Desarrolladores, gerentes de productos y otros tomadores de decisiones clave (Mittman, 2018, pág. 15).
- Colaboradores: todos los que han contribuido con algo al proyecto. Entre estos se encuentran: Desarrolladores, DevOps, UX/UI, Analítica y otros roles (Mittman, 2018, pág. 15).
- Miembros de la comunidad: personas que utilizan el proyecto. Pueden participar activamente en conversaciones o expresar su opinión sobre la dirección del proyecto (Mittman, 2018, pág. 15).

#### 5.2.4.4 Herramientas básicas

Entre las herramientas básicas para la adopción de ISS se encuentran, sin que sean las únicas:

- Un sistema de control de versiones de fuentes, como GitHub, que permita documentar los cambios en el código, así como la solicitud de revisiones para integrar estos cambios (Mittman, 2018, pág. 17). El sistema elegido debe tener la capacidad de gestionar "*issues* (propuestas)", "solicitudes de características" (nuevas funcionalidades) y "reportes de *bugs*", a manera de registrar las discusiones para consulta posterior (Mittman, 2018, pág. 17).
- Preferiblemente un sistema de chat empresarial que persista las conversaciones y estas se puedan consultar por otros, como sucede en Slack (Mittman, 2018, pág. 17).

#### 5.2.4.5 Gestión de contribuciones

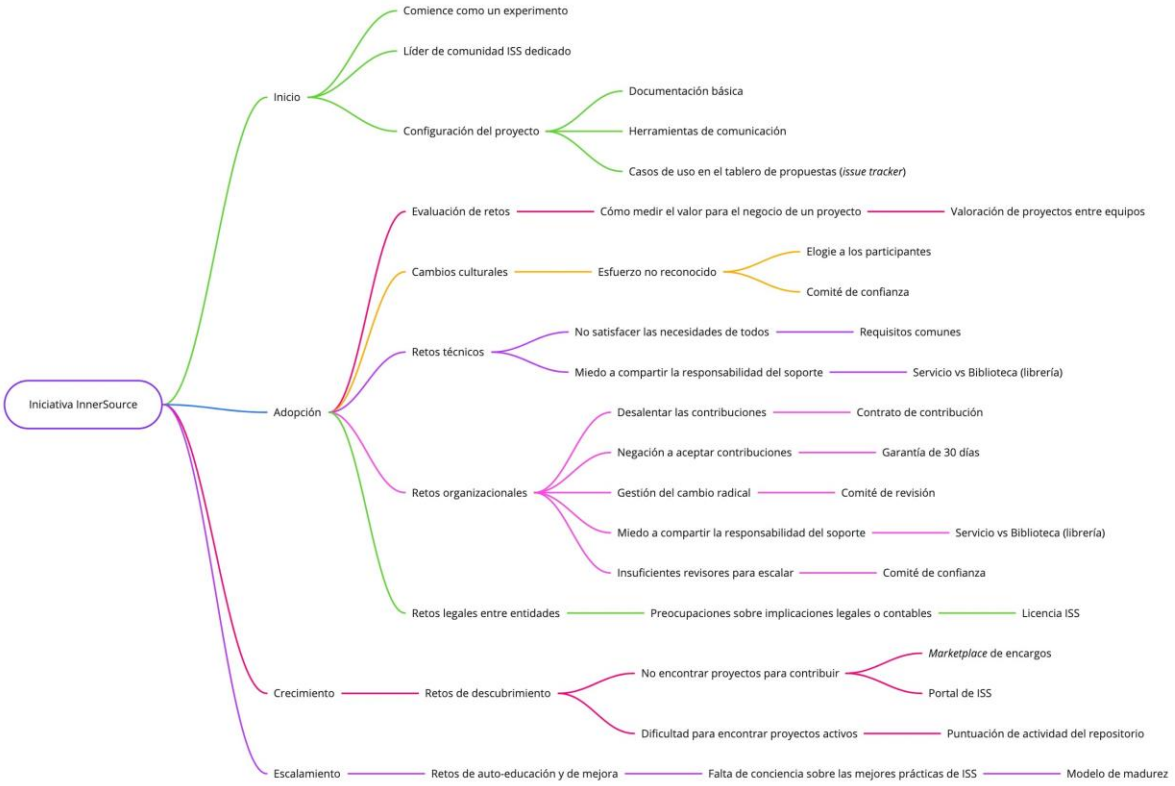
Para que las contribuciones sean realmente efectivas se deben establecer y seguir una serie de normas:

- Establecer reglas sobre las características que debe tener una contribución para ser aceptada, así como el tamaño máximo que debe tener una *pull-request* (Mittman, 2018, pág. 31).
- Hacer que el proyecto ISS sea fácil de encontrar (Mittman, 2018, pág. 34):
    - Breve descripción del proyecto: título y *tagline*.
    - Enlace al sitio web del producto.
    - Uso de herramientas de GitHub® como *issues* (*Bug*, Pregunta o Solicitud de función) y *pull-requests*.
    - Uso de temas o tópicos para promover la visibilidad.
    - Código de conducta, pautas de contribución y un archivo de licencia.
- Todos los proyectos deben estar abiertos a todos los desarrolladores para su revisión y posible aporte (Mittman, 2018, pág. 35).

### 5.2.5 Retos de las ISS

Estas atractivas ventajas han hecho que un buen número de empresas, sobre todo grandes, reporten éxitos relacionados con el uso de esta práctica, pero la misma supone retos importantes a la hora de integrar los desarrollos comunes dentro de los productos (Stol et al., 2011, pág. 1). Los retos mas comunes han sido resueltos de forma comprobada y repetible, y documentados por la comunidad Inner Source Commons (ISC) en forma de patrones (ISC, 2021, págs. 3-4). La Figura 1 muestra el mapa mental de relaciones entre los principales retos de ISS y los patrones recopilados por la comunidad ISC.

Figura 1: Mapa mental de los principales retos de ISS y sus patrones.




Fuente: Traducción propia basada en (ISC, 2021, pág. 7).

Estos patrones deben usarse con cuidado, no se pueden aplicar indiscriminadamente, en la mayoría de los casos, deberá adaptar la solución dada a su situación particular (ISC, 2021, pág. 4). En la Tabla 3 se presenta un resumen de los patrones documentados por la ISC hasta la fecha de redacción de este trabajo de grado, contrastando el problema que resuelve, la solución y las empresas que han reportado estos casos de éxito.

Tabla 3. Breve resumen de los patrones ISC

| Patrón | Problema | Solución | Caso |
|---|---|---|---|
| Garantía de 30 días | Existe renuencia natural a aceptar contribuciones. Algunos aducirán que es mejor desarrollar la solución antes que analizar el código externo (Oram, 2021, pág. 20); o que el nuevo código comprometerá el futuro de la solución (Stol et al., 2011) citado por (Oram, 2021, pág. 20). | El contribuidor se comprometerá a proporcionar correcciones de errores al equipo receptor, lo que aumentará el nivel de confianza entre ambos y aumentará la probabilidad de aceptación de las contribuciones. | PayPal |
| Requisitos comunes | El código común no satisface las necesidades de todos los equipos de proyecto que desean usarlo. | Alinear y refactorizar requisitos. | PayPal |
| Herramientas de comunicación | Los equipos usuarios tienen problemas para obtener ayuda y ponerse en contacto con el equipo del proyecto. | Configurar y documentar herramientas que permitan que las discusiones sean visibles, almacenadas y fáciles de encontrar. | Europace, PayPal |
| Contrato de contribución | La gerencia a la que pertenece el desarrollador le desalienta a que contribuya a ISS. | Realizar contratos y acuerdos formales donde deje claro, entre otras, cuanto porcentaje de tiempo del contribuidor necesita la organización para ISS. | Robert Bosch GmbH |
| Valoración de proyectos entre equipos | Es difícil vender el valor de los proyectos de ISS entre equipos que no tienen un impacto directo en los ingresos de la empresa. | Crear valoraciones basadas en datos para visibilizar los aportes. | Nike |
| Líder dedicado a la comunidad ISS | ¿Cómo se asegura de que una nueva iniciativa de ISS aumente su impacto en su comunidad de desarrolladores? | Seleccionar personas con habilidades técnicas y de comunicación para liderar y garantizar el éxito en el inicio de una iniciativa de ISS. | Robert Bosch GmbH |
| *Marketplace* de encargos | Los contribuidores no saben como participar en un proyecto ISS puede beneficiarle; tampoco saben como indicarle a su gerencia el compromiso con un proyecto. Los gerentes no tienen como realizar seguimiento y recompensar a quienes aporten a un proyecto ISS. | Listar las necesidades de la organización a manera de "encargos" (*gigs*), con requisitos explícitos de tiempo y habilidades. Esto dará a los gerentes una visión global de las necesidades de la empresa. | SAP |
| Licencia ISS | Dos entidades legales que pertenecen a la misma organización desean compartir ISS entre sí, pero les preocupan las implicaciones legales o contables. | Crear licencias de uso de software y código ISS. | DB Systel |

| Patrón | Problema | Solución | Caso |
|---|---|---|---|
| Portal de ISS | Los proyectos no atraen contribuciones. Los contribuidores no tienen una manera fácil de descubrirlos. | Crear un sitio web interno que indexe la información disponible de los proyectos ISS para facilitar que los que atraigan contribuidores. | SAP, Elbit Systems, American Airlines |
| Casos de uso en el Tablero de Propuestas (*Issue Tracker*) | El proyecto ISS no logra que los planes y el progreso sean transparentes, tampoco el contexto para los cambios. | Aumentar los casos de uso para que el Tablero de Propuestas (*Issue Tracker)* del proyecto también sirva para la lluvia de ideas, la discusión de la implementación y el diseño de funciones. | Europace |
| Modelo de madurez | La organización ha comenzado a adoptar ISS. La práctica se está extendiendo a varios proyectos. Sin embargo, no se comprende bien lo que constituye un proyecto ISS. | Proporcionar un modelo de madurez que permita a los equipos realizar una autocomprobación y descubrir patrones y prácticas que aún no conocen. | Entelgy, Zylk, Bitergia |
| Elogiar a los participantes | Olvidar agradecer a un contribuidor puede afectar su compromiso y disuadir a que otras personas aporten. | Agradecer al colaborador por su tiempo y esfuerzo, al final de cada contribución. | Nike |
| Puntuación de actividad del repositorio | Los contribuidores potenciales desean encontrar proyectos activos de ISS que necesiten su ayuda. | Calcular un puntaje de actividad del repositorio para cada proyecto, por ejemplo, en el Portal de ISS, para facilitar al contribuidor la decisión del proyecto al que desea contribuir. | SAP |
| Comité de revisión | El modelo de trabajo de ISS es una desviación radical de los enfoques más tradicionales, tanto para desarrolladores como para gerentes. | Establecer un comité como interfaz entre la iniciativa ISS y los altos directivos que participan en ella. Esto los familiarizará con la iniciativa y hará que la apoyen, ya que les otorga un cierto nivel de supervisión y control sin fomentar la microgestión (*micromanagement*). | Robert Bosch GmbH |
| Servicio vs. Biblioteca (librería) | Los equipos pueden ser reacios a trabajar en bases de código comunes debido a la ambigüedad sobre quién será responsable de responder ante una “caída” de ese servicio. | Validar la posibilidad de implementar la misma solución como servicio en entornos independientes con escalamiento de soporte separados o compartir códigos en bibliotecas. | Europace |
| Documentación básica | Los nuevos colaboradores de un proyecto de ISS tienen dificultades para determinar quién mantiene el proyecto, en qué trabajar y cómo contribuir. | Proporcionar documentación en archivos estándar como README.md/CONTRIBUTING.md permite un proceso de autoservicio para encontrar respuestas a las preguntas más comunes. | Europace, PayPal |

| Patrón | Problema | Solución | Caso |
|---|---|---|---|
| Comenzar ISS como un experimento | Se considera una iniciativa de ISS, pero no se inicia porque la gerencia no está segura de su resultado y, por lo tanto, no está dispuesta a comprometerse con una inversión. | Comenzar la iniciativa de ISS como un experimento de tiempo limitado para facilitar el apoyo y respaldo de los gerentes. | Robert Bosch GmbH |
| Toma de decisiones transparente entre equipos mediante RFC | ¿Cómo dar mas transparencia a los procesos, atraer contribuidores desde instancias tempranas del proyecto y que estos aporten desde el inicio pudiendo encontrar errores de diseño? | Publicar los documentos internos de RFC (*Request For Comments*: Solicitud de Propuesta Formal) para habilitar el debate en las primeras etapas del proceso de diseño y aumenta las posibilidades de crear soluciones con un alto grado de compromiso de todas las partes involucradas. | Google, Uber, BBC, Europace |
| Comité de confianza | Muchos proyectos de ISS se encontrarán en una situación en la que recibirán constantemente comentarios, funciones y correcciones de errores de los colaboradores. | Reconocer y recompensar el trabajo de los colaboradores más activos promoviéndolos al comité distribuye el control haciendo que las revisiones y aprobaciones de código sean más ágiles. | Nike, PayPal |

Fuente: Propia, basada en (ISC, 2021).

El establecimiento de esta gobernanza no ahuyentará la colaboración, al contrario, la investigación de (Capraro, 2020, pág. 132) demuestra que la aumenta en comparación a los proyectos ISS con pocas o nulas restricciones.

A pesar de la lista plasmada en la Tabla 3, las ISS también presentan unos retos de alto nivel, que son propios a cualquier cambio cultural, y que requerirán vigilar constantemente como:

1. Algunas empresas ya implementan prácticas de ISS aunque no lo llaman de esta forma (Capraro, 2020, pág. 63), lo cual reduce el número de resultados durante las búsquedas bibliográficas.
2. A diferencia del OSSD las ISS no son actividades de voluntariado (Mittman, 2018, pág. 32).
3. ISS no es una metodología definida y cada implementación es hecha a la medida de cada organización (Stol & Fitzgerald, 2014, págs. 1-2).
4. No hay métodos formales para medir el impacto de las ISS (Capraro, 2020, pág. 2).
5. La implementación de las ISS dependen de que la empresa tenga un esquema de integración y despliegue continuo (CI/CD, DevOps) (Mittman, 2018, pág. 20).
6. Renuencia para contribuir: Contribuir a un activo compartido es naturalmente más complejo. Los desarrolladores generalmente no están acostumbrados a pensar y diseñar soluciones que fueran más generales que su propia línea de producto (Stol et al., 2011, págs. 31-32).
7. Cambio cultural a nivel organizacional: la empresa debe realizar acciones que fomenten el

intercambio de conocimientos y aliente la recepción de contribuciones de toda la plantilla de desarrolladores (Mittman, 2018, pág. 11) (Stol et al., 2011, pág. 33).

### 5.2.6 Conclusiones de la revisión de literatura

La documentación consultada resulta ser la mas referenciada acerca de ISS y entrega información importante sobre los aspectos positivos y retos que tiene implementar ISS en una organización. Las empresas pueden utilizar ISS no solo por las ventajas relacionadas con mejora de calidad y reutilización tanto del código como del conocimiento dentro de su plantilla de programadores, sino también, ahora por el advenimiento del trabajo remoto como respuesta a la COVID-19, por las ventajas de comunicación en entornos remotos heredada de los esquemas de OSSD y que de acuerdo a (Capraro, 2020, págs. v y 122) era una de las características mas buscadas por la industria antes de la pandemia.

Las prácticas ISS contribuyen a la organización agilizando su velocidad de respuesta a las necesidades del mercado. Los proyectos podrán contar con múltiples contribuidores que ayuden a identificar y resolver *bugs* o que aporten nuevas características. Con ISS las nuevas características identificadas por "agentes externos" al proyecto pueden ser desarrolladas por ellos mismos sin pasar por el cuello de botella de esperar a que el equipo "dueño" del proyecto tenga el tiempo suficiente para priorizar la tarea en su *backlog*, refinarla y finalmente desarrollarla, lo que podría tardar hasta meses. Por otro lado, ISS puede ayudar a desarrollar en los programadores habilidades profesionales de alto valor relacionadas con cultura de la colaboración; sofisticación en la documentación y la forma de comunicar, tanto para justificar a otros sus decisiones como para rechazar contribuciones a sus proyectos.

Es importante resaltar que ISS puede ser el vehículo para lograr cambios de gran impacto en las organizaciones, ya que de acuerdo con (Oram, 2021, pág. 21) "Los programadores modernos aprenden a prosperar con el cambio. Aparte de las nuevas tecnologías y herramientas, un cambio cultural puede ayudarles a mantenerse ágiles y a actualizar sus habilidades. ISS es un paso hacia todos estos logros". Estos cambios suponen también una oportunidad para descubrir talentos dentro de la misma plantilla de desarrolladores, pues estos ya no tendrían que quedarse únicamente en el rol, producto o la tecnología para la que se han contratado, sino que pueden demostrar su valía o desarrollar sus habilidades para aportar a la organización desde otros puntos.

Las ISS no son unas prácticas perfectas que resuelven todos los problemas, además de los retos identificados en las referencias, la revisión de literatura también generó algunas recomendaciones que, si bien no están escritas de forma explícita, se detallan a continuación y se tuvieron en cuenta durante la ejecución de este trabajo de grado:

- Hacer un código abierto (no confundir con OSSD), mejor desarrollado, mejor documentado,

con muchos revisores, es mas complejo y de forma natural demora un poco más. Las empresas deben ser capaces de ver que esta demora es menos costosa que la reinvención de soluciones en cada equipo.

- La apertura del proyecto no indica que todo su código debe estar listo para ser reutilizado y contar con una documentación detallada, pero si exige que desde las fases de diseño se piense que componentes estarán abiertos y hacer las gestiones para que sea posible su aprovechamiento por la comunidad de desarrollo de la organización.
- Cada organización debe ser capaz de redactar un manual que permita identificar que proyectos, o partes de estos, deben tratarse como ISS o no.
- Las organizaciones deben tener claro que no es obligatorio abrir los proyectos al público; que este es un ideal de las prácticas ISS.

Los hallazgos de esta revisión bibliográfica proveen valiosas pistas para que las empresas interesadas en adoptar ISS lo hagan conociendo de ante mano las ventajas y retos y esto les permita elaborar un plan aterrizado de adopción. Puesto que sus ventajas en lo concerniente a trabajo colaborativo, reutilización de software y aumento de calidad y velocidad de entrega de productos es mayor a cualquier reto, y los retos de implementar ISS son naturales a cualquier cambio organizacional, (Stol et al., 2011, pág. 12) recomienda imitar las experiencias de HP, Alcatel o Phillips Healthcare: enfocarse en aquello que funciona antes que en los retos. ISS es una práctica en constante evolución, que cada empresa debe ir adecuando en el camino, pero en este camino no está sola, pues, además de la academia, cuenta con la comunidad Inner Source Commons que documenta de forma rigurosa las experiencias y casos de éxito de ISS provenientes de todas las latitudes, donde se destacan, entre sus contribuidores, actores relevantes de la industria del software.

## 5.3 TRABAJOS RELACIONADOS

### 5.3.1 Oportunidades para la reutilización de software en un mundo incierto: del pasado a las tendencias emergentes (Capilla et al., 2019)

Título original: *Opportunities for software reuse in an uncertain world: From past to emerging trends*.

Resumen: Este *artículo* hace una revisión bibliográfica sobre la historia de la reutilización de software, sus motivaciones, la situación actual y recopila tendencias futuras sobre el tema en cuestión.

### 5.3.2 Implementación de un repositorio para el catalogo, búsqueda y uso de componentes software reutilizables en el desarrollo de aplicaciones web (Vargas-Fandiño et al., 2020)

Resumen: Este *artículo* muestra el proceso de creación de un repositorio de elementos reutilizables en una organización, en este caso la Universidad Francisco de Paula Santander (Cúcuta, Colombia), y la posterior evaluación. Este trabajo ha servido como instrucción e inspiración para entender que la reutilización no solo es un problema de la industria, sino que también se presenta como un interesante tema de investigación académica.

### 5.3.3 Inicie con InnerSource, las claves para la colaboración y productividad dentro de su compañía (Oram, 2021)

Título original: *Getting started with InnerSource, keys to collaboration and productivity inside your company*.

Resumen: este *booklet* de la editorial O'Relly hace una serie de entrevistas a personas que hicieron parte del proceso de adopción de ISS en la empresa financiera PayPal.

### 5.3.4 Entendiendo la lista de chequeo de InnerSource, como catapultar la colaboración en su empresa (Bonewald, 2017)

Título original: *Understanding the InnerSource Checklist, How to Launch Collaboration Within Your Enterprise*.

Resumen: Este libro de la editorial O'Relly trae a la directora de ISS en PayPal compartiendo un análisis del proceso de montaje y la idea de implementar una lista de chequeo que permita a otras organizaciones dar el salto.

### 5.3.5 Adoptando InnerSource, principio y casos de estudio (Cooper et al., 2018)

Título original: *Adopting InnerSource, Principles and Case Studies.*

Resumen: En este libro de la editorial O'Relly se presenta Denise Cooper quien fue la

encargada del montaje de ISS en PayPal y fue fundadora de ISC, y al académico Klass-Jan Stol que ha conducido investigaciones sobre OSS e ISS en los últimos 10 años. En este libro se hace un estudio concienzudo sobre los procesos de OSS, la filosofía o estilo de OSS de Apache (*The Apache Way*) y los procesos de implementación de ISS en Bell Laboratories, Bosch, PayPal, Europace y Ericsson; entregando no solo el relato de la estrategia llevada acabo, sino también lecciones aprendidas y consejos. Como parte de sus conclusiones se indica que ISS es un tema de cultura más que de herramientas, una situación similar a la que ocurre en el Laboratorio ya que tienen todas las herramientas necesarias, pero no logra impulsar dentro de los desarrolladores la reutilización.

En conclusión, los anteriores trabajos listados en esta sección permitieron obtener un contexto general de lo que es la reutilización de software, como se han utilizado estas técnicas para desarrollar proyectos académicos y como empresas referentes en el desarrollo de software resuelven mediante ISS las complejidades de las dinámicas sociales alrededor de la construcción y reutilización de software.

# 6 DESARROLLO

El siguiente capítulo, relaciona las actividades realizadas para diseñar e implementar un gobierno autogestionado para el repositorio de Servicios Comunes (RSC) del Laboratorio Digital del Banco de Occidente bajo las prácticas InnerSource (ISS) como estrategia de Reutilización de Software. Este trabajo se abordó en cuatro fases correspondientes al diagnóstico de la situación, diseño de una solución personalizada (a la medida), implementación de dicha solución y finalmente una evaluación para confirmar si esta solución resolvió las situaciones diagnosticadas en la primera fase.

## 6.1 DIAGNÓSTICO DE LA SITUACIÓN ACTUAL

En el diagnóstico se realizaron las acciones necesarias para identificar los problemas respecto a la gestión del repositorio de Servicios Comunes (RSC) que limitaban la reutilización de los componentes de software contenidos en este repositorio. Las actividades realizadas en esta fase son detalladas a continuación.

### 6.1.1 Recopilación de información de los problemas de reutilización del repositorio

Esta recopilación de información tuvo tres tareas principales:

1. Diseñar una encuesta sobre la reutilización de software y su percepción acerca de los problemas que limitan la reutilización (ver **Encuesta de diagnóstico y análisis de resultados**).
2. Encuestar a todos los desarrolladores del Laboratorio.
3. Análisis de los resultados de la encuesta (ver **Encuesta de diagnóstico y análisis de resultados**).

La encuesta mencionada (ver apartado **Encuesta de diagnóstico y análisis de resultados**) se realizó a toda la población del Laboratorio Digital en el mes de septiembre de 2021. Para esto se diseñó un formulario que permitió obtener información de todos los roles sin importar su grado de experiencia o conocimiento y poder capturar el “estado del

arte" de la situación e incluso contrastar las respuestas entre los roles administrativos del laboratorio, de negocio de los proyectos y técnicos (los desarrolladores), siendo estos últimos el foco principal. La Tabla 4 relaciona la ficha técnica de la encuesta, para la cual el Laboratorio proveyó una base de colaboradores con 141 personas, de las cuales se eliminaron las vacantes, las personas en vacaciones y cargos como los de áreas de soporte como Talento Humano.

Tabla 4. Ficha técnica de la encuesta de diagnóstico

| **Población total** | |
|---|---|
| Población | 127 |
| Nivel de confianza | 95 % |
| Margen de error | 5 % |
| Tamaño de la muestra | 97 (76 %) |
| **Desarrolladores (foco)** | |
| Población | 55 |
| Tamaño de la muestra | 42 (76 %) |

En la Tabla 5 se relaciona la clasificación de cargos que se hizo para simplificar el análisis de la información. En este caso, los *Scrum Masters* se clasificaron como administrativos, pues su acercamiento al RSC está más alineado a los administrativos que a los otros roles.

Tabla 5. Clasificación cargos para la encuesta

| **Administrativo** | **Desarrollador** | **Negocio** |
|---|---|---|
| • Administrativo<br>• *Scrum Master* | • Arquitecto(a)<br>• Desarrollador(a)<br>• QA | • Navegador(a)<br>• PO / Analista PO |

La encuesta se cerró con un total de 90 respuestas (93 %), de las cuales, 43 correspondieron a los desarrolladores, obteniendo un 102 % de alcance en este grupo. Aunque la encuesta fue anónima, se solicitó un rango de periodos de tiempo de contratación de la persona, para que cuando se realizara la encuesta de evaluación al final de este proyecto se pudiera gestionar las situaciones relacionadas a la rotación del personal.

En la sección **Encuesta de diagnóstico y análisis de resultados** se puede ver en detalle cada una de las preguntas y las respuestas obtenidas. Como resultados generales de la encuesta, y algunas entrevistas realizadas, se pudo encontrar que:

1. Los Administradores tienen una visión totalmente optimista sobre el uso del RSC y desconocían que se presentaban situaciones que afectaban la reutilización de sus componentes.
2. Los Arquitectos en general y los otros roles de las células que han usado el RSC tienen una visión diferente que la de los administrativos; considera que se requiere mejora y

organización.

3. Aunque el 84 % de los desarrolladores afirma haber escuchado del RSC, solo el 31 % afirma saber como ubicar sus activos. De este 31 %, que dice saber como ubicar la información, solo el 36 % (el 9 % del total de desarrolladores) conocía la forma real (pre-ISS) de buscar esta información: ir preguntado. La pregunta también ofrecía opciones que no existían en el momento y que se implementarían con el Gobierno como es el caso de listas organizadas y tableros (Jira) centralizados. En términos numéricos, solo 5 de 43 desarrolladores conocían la forma real de ubicar la información del RSC.

### 6.1.2 Análisis y priorización de los problemas detectados

Para el diagnostico de los inconvenientes que se presentaban con respecto a la gestión del Repositorio de Servicios Comunes (RSC), limitando la reutilización de los componentes de software contenidos en este, se llevaron a cabo conversaciones con las directivas del Laboratorio y con los Arquitectos de dos células que recientemente habían utilizado el RSC. Estas conversaciones evidenciaron diferentes problemas:

1. Para el Arquitecto #1, que tuvo contacto con un servicio del RSC en diciembre de 2020, este presentaba problemas de rendimiento en las consultas a la base de datos, sumado a lo anterior era difícil el soporte de la célula que desarrolló el activo de software; no hubo documentación pues el desarrollador se fue del Laboratorio y tampoco se pudo obtener acceso al código fuente. Esto llevó a que su célula implementara un servicio complementario propio para suplir los inconvenientes encontrados con respecto al desempeño del servicio del RSC.
2. El Arquitecto #2, que usó el RSC en junio de 2021, confirma que utilizó el mismo componente (servicio) que el Arquitecto #1, encontró exactamente la misma situación y desde su célula implementaron una nueva solución complementaria particular, pues desconocían que la célula #1 ya había pasado por esta situación.
3. Durante la entrevista al Arquitecto #2 se determinó que, para resolver el tema de rendimiento, utilizaron una solución simple de recortar el número de caracteres que se envía al servicio del RSC. Informa que dado lo sencillo de la solución, intentó aplicarla directamente al activo del RSC, pero no logró obtener acceso al código fuente. Se confirma que la célula del Arquitecto #1 desconoce esta solución y la ha controlado de una forma mucho mas compleja y menos efectiva.

4. Ambas células tuvieron que desarrollar sobre *mocks*[3] construidos a partir del *swagger*[4] del servicio dado que los ambientes bajos (desarrollo y pruebas) no funcionaban. Además, las soluciones realizadas por ambas células se aplicaron después de una salida "en falso" a producción, donde descubrieron los inconvenientes y durante varios días tuvieron que dejar de atender segmentos de clientes que les habilitaría la aplicación del servicio del RSC.

En conclusión, los temas diagnosticados y a los cuales se les dará solución son:

- Consolidar y organizar la información de los RSC.
- Dar acceso a todos los desarrolladores sobre esta información.
- Indicar como obtener acceso al código de estas soluciones.
- Dar a conocer a todo el Laboratorio como consultar esta información y en especial, como aportar a ella y al software, tanto con soluciones como con ideas.
- Entregar herramientas o métodos que permitan recopilar de forma productiva estos aportes.

Estos puntos son la base del diseño del gobierno, que por petición de la organización será autogestionado y por ende de modelo tipo "infraestructura" (ver Modelos de ISS), que será el resultado de este proyecto. Esta actividad se aborda en el siguiente apartado.

## 6.2 DISEÑO DE UN GOBIERNO AUTOGESTIONADO

En este apartado se diseñó un gobierno autogestionado para el Repositorio de Servicios Comunes (RSC) bajo las prácticas InnerSource (ISS) para incidir de forma positiva sobre los problemas diagnosticados con respecto a la gestión de este repositorio con el fin de reutilizar los componentes de software que contiene. Todos los elementos que se mencionan en este apartado se detallarán como anexos del presente documento, para no afectar el flujo de la lectura.

[3] *Mocks*: objetos de software que imitan el comportamiento de otros.
[4] *Swagger*: documentación autogenerada de las API de tipo REST. Ver glosario para mayor detalle.

### 6.2.1 Revisión de literatura en búsqueda de soluciones a los problemas detectados

Después de identificar los problemas en la fase anterior, en cuanto a la reutilización de los componentes de software almacenados en el RSC, y teniendo en cuenta que será una implementación ISS de modelo "infraestructura", se aseguró que el Gobierno ISS se diseñara "a la medida" de las necesidades del Laboratorio. La Tabla 6 relaciona las soluciones o patrones encontrados a los problemas identificados.

Tabla 6. Aplicación de patrones ISS a los problemas detectados

| Problema | Solución o patrón |
|---|---|
| Documentación | Portal InnerSource en *Confluence* (ver Tabla 3). A esta información se dará acceso de lecto/escritura a todos los roles para canalizar sus aportes. |
| Soporte por parte de la célula que desarrolló el componente del RSC | Garantía de 30 días (ver Tabla 3). |
| Inventario de ambientes y recursos | Portal InnerSource en *Confluence* (ver Tabla 3). |
| Documentación del proceso de pruebas de software o certificaciones de ambientes | Portal InnerSource en *Confluence* (ver Tabla 3). |
| Conocer de antemano las falencias o necesidades de implementación que requiera el componente | Tablero de Jira para gestionar los *Issues*. Aunque en la literatura consultada se recomienda hacerlo dentro del repositorio de GitHub (ver apartado "Herramientas básicas"), se decidió por Jira para no excluir a los roles no técnicos (57 % de la población). |
| Acceso al código | El Portal InnerSource tendrá documentadas las rutas de acceso al código, así como información de diseño de arquitecturas.<br>Todos los desarrolladores tendrán inicialmente acceso de solo lectura a los repositorios. Pero en el portal ISS tendrán el instructivo de como ejecutar los scripts de permisos. |

### 6.2.2 Desarrollo del gobierno ISS

El desarrollo del Gobierno ISS "a la medida" fue la parte más importante de este trabajo de grado ya que había muchas definiciones por levantar, acuerdos por hacer y textos por redactar, los cuales se incluyen por completo en el **ANEXO A: GOBIERNO**.

Entre las tareas realizadas para llevar acabo esta actividad estuvieron:

1. Se definieron las reglas de juego ISS, adaptadas al Laboratorio.

2. Se creó la plantilla base para nuevos proyectos ISS con todos los documentos del Gobierno ISS.
3. Se redactaron los manuales del Gobierno ISS.

Antes de definir las reglas de juego, se sentaron unas pautas que permitieran incidir de forma positiva sobre las necesidades del Laboratorio y prevenir las situaciones detectadas en los **Retos de las ISS**. Estas pautas fueron:

- Que la información estuviera organizada y optimizada para que fuera de fácil descubrimiento durante las búsquedas.
- Que la documentación ayudara a los desarrolladores a tomar decisiones rápidas sobre si aportar o no: lenguaje, calidad estática de código, diagrama de arquitectura, etc.
- Que la documentación contara con lo mínimo posible para no desestimar los aportes y fuera lo suficientemente completa para que cualquiera lo pudiera usar de forma autónoma (sin solicitar ayuda).

Previo al diseño del gobierno se estableció, como recomiendan las prácticas, un Comité de ISS que contó con la participación de un Administrativo del Laboratorio y uno de los Arquitectos entrevistados en la fase anterior. Acto seguido, se procedió a realizar un inventario de los servicios del RSC.

Para establecer la estructura básica o plantilla de proyectos, se analizó la documentación de 40 microservicios de la organización, incluidos los 12 del RSC, se establecieron las secciones que tendría la documentación del servicio, basado en las buenas prácticas detectadas en esta revisión y las recomendaciones de la literatura consultada en el **MARCO TEÓRICO DE REFERENCIA Y ANTECEDENTES**. La propuesta del diseño se presentó a evaluación por parte del Comité ISS. Tras realizar los ajustes propuestos por el Comité, se escogió un Proyecto Semilla donde se aplicó y se presentó en reuniones a las diferentes células cuyas retroalimentaciones se implementaron en la versión final de la plantilla, que se presenta a continuación:

1. Descripción básica del servicio: indicando el estado de despliegue del servicio, y el lenguaje de programación utilizado.
2. Arquitectura:
    1. Diagrama de arquitectura.
    2. Listado de recursos del servicio: (Url de repositorio, Url de análisis estático de código, Url de repositorio de Casos de Pruebas).
    3. Listado de *endpoints*: listado de los elementos relacionados con el servicio.
    4. Información de los servicios externos, aplican para los microservicios que hacen las veces de pasarela (proxy).
3. Documentación de la API bajo estándar Open API / Swagger: tiene información sobre

cada campo de consulta (*request*) o respuesta (*response*) del servicio.

4. Observabilidad: información de los *logs* que permiten analizar el funcionamiento del servicio.
5. Certificación del paso a producción.
6. Listado de células o equipos que utilizan dicho servicio: este permite gestionar el riesgo en caso de que se quiera contribuir con código.
7. Además, cada servicio tendría una sección de bitácora de despliegue donde se resumen los cambios y sirve como registro de cambios (Changelog).

Como reglas de juego o gobierno para el uso del repositorio de forma autogestionada, se redactó dicho Gobierno a manera de preguntas y respuestas, una versión detallada de este se puede encontrar en el **ANEXO A: GOBIERNO**, que se pueden agrupar o resumir como:

- Instrucciones para realizar aportes: quienes pueden hacerlo, de que tipo.
- Garantía de 30 días.
- Como atraer aportes a un servicio.
- Manuales anexos sobre:
    - Como leer y gestionar la documentación de las API mediante Swagger.
    - Gestión de seguridad o *tokens* que requieren estos servicios
    - Gestión de la calidad de procesos de software que incluye pruebas de rendimiento y automatización de pruebas funcionales o de regresión.
    - Recomendaciones básicas de GitHub: Código de Conducta y *Commits* Atómicos.
    - Uso conceptual del tablero Jira, donde se tratan temas de: como incluir tareas, como aprobarlas, como reportar que se está trabajando en una, y cómo “descartar” una idea de forma correcta. Esta última es muy importante, pues llamando el valor de transparencia debe quedar claro los motivos y además sirve para que no se presente la misma idea desde diferentes personas.
- Aunque en la mayoría de la literatura consultada se alienta a calificar la documentación encontrada, en este gobierno se invita a que las personas tomen la acción de actualizarla.

Una vez validado el diseño del gobierno, se procedió a implementarlo en los otros 11 servicios del RSC, actividad que se detalla en la siguiente fase.

## 6.3 IMPLEMENTACIÓN DEL GOBIERNO EN EL REPOSITORIO DE SERVICIOS COMUNES

En la implementación se llevaron a cabo las acciones necesarias para poner en funcionamiento el gobierno autogestionado en el Repositorio de Servicios Comunes (RSC) bajo las prácticas ISS como estrategia de Reutilización de Software al aplicar las soluciones propuestas en los **TRABAJOS RELACIONADOS,** a los problemas diagnosticados en la primera fase de esta sección de desarrollo (ver **DIAGNÓSTICO DE LA SITUACIÓN ACTUAL**). Los siguientes ítems muestran en detalle las acciones realizadas para llevar a cabo esta actividad:

1. Preparación de la implementación del gobierno autogestionado.
2. Implementación del gobierno a todo el RSC.
3. Presentación y capacitación a todo el personal.

Esta fase también se convirtió en ruta crítica del proyecto pues se desconocía que hubiera tantos servicios, 12 en total. Lo cual, aunque ya las definiciones estaban hechas en la fase anterior, requirió mayor esfuerzo para acomodar la información existente y averiguar la que faltaba con las consecuentes situaciones como: disponibilidad de los expertos, rotación de personal o incluso extravío de la data; en algunos casos se requirió acceder al código fuente para rastrear dependencias e interpretar funcionamientos.

Para determinar si finalmente se contaba con el inventario completo, se realizaron las siguientes acciones:

1. Se comprobó el listado de servicios obtenidos contra los servicios registrados en el API Gateway del RSC. Esta actividad se realizó al exportar el API Swagger del API Gateway como se explica en la sección del Gobierno "**Cómo documentar una API con Swagger.**".
2. Se buscó en el GitHub Empresarial los repositorios que tuvieran la nomenclatura relacionada a este tipo de servicios, para revisar la existencia de proyectos comunes no desplegados en el RSC.
3. Se hizo una búsqueda amplia en Confluence con palabras claves relacionadas con proyectos que podrían ser de uso común.
4. Se centralizó en el portal de documentación del RSC la información de los servicios comunes, tras eliminar de los espacios privados de las células las versiones alternas encontradas.
5. Se realizó limpieza al remover duplicados y falsos positivos o experimentos fallidos tanto del API Gateway, del GitHub y de la documentación interna de las células.

Tras extraer la lista de servicios existentes se comprobó que estuvieran bien clasificados

como nivel 2 de acuerdo con lo establecido organizacionalmente, que en resumen se pueden ver como se relaciona en la Tabla 7.

Tabla 7. Resumen de niveles de reutilización de servicios

| Nivel de reutilización de servicios | Gobierno |
|---|---|
| **Nivel 1**: servicios que solo sirven a una célula/proyecto | Cada célula. |
| **Nivel 2**: servicios que pueden servir a varias células de la entidad, en el caso específico de este trabajo de grado: Banco de Occidente. | Se espera que cada entidad encuentre una forma de gobierno. |
| **Nivel 3**: servicios transversales a otras entidades del Grupo Aval. | Células transversales específicas. |

Fuente: Propia, basada en documentación de la organización

Y es precisamente en estos servicios de nivel 2, los transversales a nivel Banco de Occidente, donde hacía falta establecer las bases de un gobierno que fue lo que se hizo con este trabajo de grado. Se aclara que al averiguar sobre como lo gestionan otras entidades del grupo, se encontró que estas crearon células específicas como se hace en el nivel 3, y que en el nivel 3 se inició un grupo de estudio sobre InnerSource, pero nunca hicieron la primera reunión.

Tras implementar el gobierno en el proyecto piloto, se procedió a aplicar en el resto del RSC. Se encontró que un Arquitecto había propuesto una clasificación de servicios, la cual se revisó y potenció en este trabajo, cuyo resultado se relaciona en la Tabla 8.

Tabla 8. Clasificación de servicios de RSC

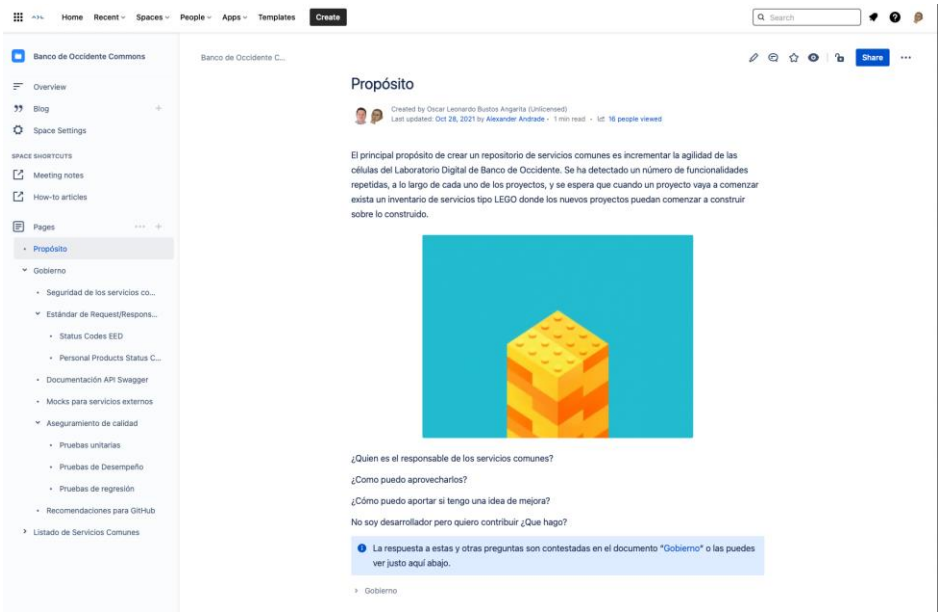
| Tipo de servicio del RSC | Descripción |
|---|---|
| Utilitarios | Servicios útiles para varias células pero que no tienen conexión con un tercero. |
| Proxy | Servicios útiles para varias células pero que tienen conexión con un tercero y aún cumple con el Nivel 2 de Reutilización (ver Tabla 7). |
| Librerías | Las librerías son proyectos con métodos o funciones puntuales en el cual puedes anexar a otros proyectos y complementarlo usando sus métodos específicos para una determinada solución. No son servicios, no exponen una API. |

Se encontró una propuesta de definición de características no funcionales para los servicios Utilitarios y Proxy, este trabajo de grado la retomó, completó y se documentó de forma tal que sirva como guía para el desarrollo y toma de decisiones sobre los servicios del RSC, cuyo resultado se relaciona en la Tabla 10.

La implementación se organizó en un espacio de *Confluence*, al cual pueden acceder y editar todos los colaboradores del Laboratorio y otras entidades del grupo empresarial. Durante las entrevistas informales se presentaron reticencias relacionados a este concepto de apertura, pero se demostró que la herramienta Confluence ofrece un buen historial de ediciones y opciones hasta de devolverse a una versión específica del texto. En la Figura 2

se puede observar un pantallazo del resultado final, donde a la izquierda se muestra la organización que tiene el mismo, pero se ha ocultado la información del listado de servicios por ser un tema sensible.

Figura 2. Pantallazo del espacio *Confluence*




Una vez implementado el gobierno en todos los componentes del RSC, se procedió a hacer presentación y capacitación a todo el Laboratorio, con el ánimo que tras salir de las presentaciones todos pudieran entrar al portal, explorarlo y hasta iniciar a aportar. Tras esto, se continuó con la actividad de validar la efectividad de la implementación, la cual se detalla en la siguiente fase.

## 6.4 VALIDACIÓN DE LA IMPLEMENTACIÓN

En la fase de validación, se evaluó que la implementación del Gobierno ISS resolviera los problemas detectados, que limitaban la reutilización de los componentes de software almacenados en el Repositorio de Servicios Comunes (RSC). Tras haber implementado el gobierno diseñado con las pautas recogidas de la revisión de literatura, ajustado a las necesidades del Laboratorio y sometido a la opinión de toda la plantilla de desarrolladores, solo quedaba realizar una encuesta que permitiera medir el grado de asertividad que tuvo este proyecto dentro de los problemas detectados; para lograr esto, se realizaron las siguientes actividades, que se detallan mas adelante:

- Diseño de una encuesta para medir la efectividad del Gobierno ISS.
- Ejecución de encuestado y tabulación de resultados.

### 6.4.1 Diseño de una encuesta para medir la efectividad del Gobierno ISS

Esta encuesta fue la piedra angular de la validación. Por tal motivo se tomó como base la encuesta realizada en la sección **Recopilación de información** para poder contrastar la percepción antes y después de la implementación del Gobierno ISS. Esto permitió mejorar el Gobierno establecido e incluso sentó las bases para proponer futuros proyectos para estudiar otros ángulos de la aplicación de las ISS en esta y otras organizaciones.

### 6.4.2 Ejecución de encuestado y tabulación de resultados

Para llevar a cabo esta actividad se realizaron las siguientes tareas:

1. Realizar encuesta.
2. Tabular los resultados de la encuesta.
3. Comparar los resultados (actuales) con los datos previos a la implementación del Gobierno ISS, para determinar los cambios en la situación.
4. Redactar mejoras y conclusiones.

La encuesta mencionada (ver apartado **Encuesta de evaluación y análisis de resultados**) se realizó a toda la población del Laboratorio Digital en el mes de enero de 2022. Para esto se diseñó un formulario que permitió obtener información de todos los roles sin importar su grado de experiencia o conocimiento y poder capturar el "estado del arte" de la situación e incluso contrastar las respuestas entre los roles administrativos del laboratorio, de negocio de los proyectos y técnicos (los desarrolladores), siendo estos últimos el foco principal. La Tabla 9 relaciona la ficha técnica de la encuesta, para la cual el Laboratorio proveyó una base de colaboradores con 144 personas, de las cuales se eliminaron las vacantes, las personas en vacaciones y cargos como los de áreas de soporte como Talento Humano.

Tabla 9. Ficha técnica de la encuesta de evaluación

| **Población total** | |
|---|---|
| Población | 144 |
| Nivel de confianza | 95 % |
| Margen de error | 5 % |
| Tamaño de la muestra | 96 (76 %) |
| **Desarrolladores (foco)** | |

| | |
|---|---|
| Población | 59 |
| Tamaño de la muestra | 45 (76 %) |

Para simplificar el análisis se agruparon los roles como muestra la Tabla 5.

La encuesta se cerró con un total de 90 respuestas (94 %), de las cuales, 45 correspondieron a los desarrolladores, obteniendo un 100 % de alcance en este grupo. Aunque la encuesta fue anónima, se solicitó un rango de periodos de tiempo de contratación de la persona, para que cuando se realizara la encuesta de evaluación al final de este proyecto se pudiera gestionar las situaciones relacionadas a la rotación del personal.

En la sección **Encuesta de evaluación y análisis de resultados** se puede ver en detalle cada una de las preguntas y las respuestas obtenidas. Como resultados generales de la encuesta, y algunas entrevistas realizadas, se pudo encontrar que:

- Se debe trabajar en reducir el hermetismo de las células. En la encuesta de diagnóstico (ver **DIAGNÓSTICO DE LA SITUACIÓN ACTUAL**) se puede observar que los Administradores del Laboratorio creían que los Servicios Comunes eran muy conocidos entre los desarrolladores, pero solo el 6 de 43 desarrolladores encuestados tenían conocimiento de estos, básicamente su número correspondía a los Arquitectos.
- Un ejercicio para romper el hermetismo es el poder trasladar buenas prácticas de documentación tanto al Gobierno directamente como a la información individual de cada célula.
- En las recomendaciones de esta encuesta y de las entrevistas informales, se sugiere designar a un responsable que acompañe a todos mientras adquieren la cultura de la responsabilidad compartida.
- Agregar mas secciones (específicas) a la documentación. Tras hablar con los Administrativos del Laboratorio, se decidió postergar esta opción hasta lograr mayor penetración de esta versión “light” del Gobierno, considerando que un número mayor de información podría hacer mas compleja su adopción y desincentivar los aportes.
- Aunque la anonimidad de las dos encuestas impidió hacer cruces entre estas, fue el garante de obtener mejor información como el oculto hermetismo de las células.

Una vez validada la efectividad de la implementación de gobierno en todos los componentes del RSC, se da por terminada la ejecución del proyecto de trabajo de grado.

# 7 CONCLUSIONES Y TRABAJOS FUTUROS

## 7.1 CONCLUSIONES

En este trabajo de grado se implementó un gobierno autogestionado para el repositorio de Servicios Comunes (RSC) del Laboratorio Digital del Banco de Occidente bajo las prácticas InnerSource (ISS) como estrategia de Reutilización de Software, para lo cual:

- Se diagnosticaron los inconvenientes de la gestión del RSC que limitan la reutilización de los componentes de software contenidos en este repositorio.
- Se diseñó un gobierno autogestionado y a la medida para el RSC bajo las prácticas ISS para incidir de forma positiva sobre los problemas diagnosticados.
- Se implementó el Gobierno ISS diseñado en todos los componentes de software del RSC.
- Se diseñó y aplicó una validación para verificar que el Gobierno ISS implementado resolvió los problemas diagnosticados.

Es importante señalar que desde el inicio de este trabajo se cuidaron todas las acciones y definiciones para que el resultado fuera totalmente útil para la organización y no se convirtiera en *deadware*, pudiendo recibir fácilmente aportes y mejoras, llevándolo al siguiente nivel de complejidad y cobertura de proyectos dentro de la empresa o sus aliados.

El trabajo realizado permite concluir que:

- Se han confirmado los hallazgos descritos en el apartado de **Conclusiones de la revisión de literatura**.
- La reutilización de componentes de software sigue siendo el santo grial de la Ingeniería de Software como dice Sommerville (2011). Pero, a pesar de la extensa documentación sobre los beneficios de la reutilización y cómo crear software reutilizable, hay poca información sobre cómo fomentar la reutilización de esos componentes; en otras palabras, solo se aborda la parte técnica de la reutilización.
- Las técnicas de reutilización no tienen en cuenta el tribalismo que puede surgir entre los desarrolladores de dichos componentes reutilizables. El potencial de reutilización de los componentes puede verse afectado por una documentación deficiente o, si existe, solo los más involucrados (la tribu) logran aprovecharla. El comportamiento tribal también puede provocar el rechazo de aportes externos debido al síndrome de “aquí no inventado” (Piezunka & Dahlander, 2014), limitando el crecimiento del componente y de la organización.

- Aunque herramientas como Jira, GitHub, Slack, *Confluence* y sus competidores, ayudan al trabajo en equipo, por si solas no hacen que un grupo de personas trabaje o se sienta como un equipo; se requieren decisiones a nivel organizacional, estrategias y reglas de juego para lograr este cometido.
- El desarrollador de software como personaje solitario está en vías de extinción. Hoy en día es poco común realizar un software en solitario, por lo que además de herramientas y lenguajes de programación, se requiere profundizar en las dinámicas sociales para producir software a nivel industrial y, sobre todo, de forma remota.
- El RSC con su gobierno ISS puede ser una válvula de escape para que los desarrolladores que se encuentran desmotivados por la quietud de sostener un mismo producto o la misma tecnología tengan la opción de experimentar nuevas situaciones, interactuando con personas de otros equipos, tecnologías y lenguajes de programación.
- Este RSC bajo ISS puede favorecer la creación de un ambiente de tutoría (*mentorship*) donde las ideas y los conocimientos se puedan compartir entre diferentes células, ayudando al crecimiento personal y profesional de las personas y, por tanto, de la compañía.
- La literatura más referenciada sobre ISS menciona poco sobre las actividades de preparación previas a la implementación y, en general, solo aborda la implementación en sí misma con pocos detalles de las actividades. En este trabajo de grado se abordan ambas fases y se pueden ver algunas definiciones a nivel de ingeniería como estándares como Open API, definición de requisitos no-funcionales, etc.
- En la literatura sobre ISS, aunque se menciona que los equipos son o planean ser remotos, los implementadores de ISS y los tomadores de decisiones se encuentran co-localizados en la misma sede de oficinas, lo que facilita la implementación. En este proyecto todo se hizo de forma 100 % remota, para lo cual se utilizó un tablero Trello no solo para monitorear el proyecto, sino también para controlar las esperas de decisiones o respuestas de otros, que en este caso se realizaban en comunicación asincrónica.
- Si bien la literatura sobre ISS recomienda que la documentación de los componentes reutilizables y el seguimiento de *Issues* se haga en GitHub, en este trabajo de grado se tomó la decisión de realizarlos en *Confluence* y en Jira respectivamente para no excluir a los roles no-técnicos, que representan el 57 % de la plantilla del Laboratorio. Por otro lado, si la documentación se hiciera en GitHub, sus modificaciones generarían redespliegues, lo que representa riesgos de indisponibilidad en los entornos.
- Aunque inicialmente se creyó que el Gobierno podía ser totalmente autogestionado, en la Encuesta de evaluación y análisis de resultados se pudo determinar que muchos consideran que es necesario contar con una persona que esté “vigilando” la estandarización, lo cual puede incidirse de forma positiva mediante el patrón ISS “Líder dedicado a la comunidad ISS” (ver Tabla 3).
- A lo largo de este Trabajo de Grado se menciona que InnerSource (ISS) es una cuestión de cultura más que de herramientas. Según (Heskett, 2021), un cambio cultural organizacional puede tomar un tiempo indeterminado para materializarse, el cual puede ser de años a

décadas, lo que supera las restricciones de tiempo del presente proyecto. En este orden de ideas, este trabajo sienta las bases para arraigar las prácticas ISS en la cultura de la organización y es necesario seguir evolucionándolo.

- Durante la realización de este trabajo de grado, su relevancia adquirió un mayor nivel de importancia debido a dos situaciones: 1) el Laboratorio tomó la decisión de volverse 100 % remoto y 2) en la actualidad se presenta un vertiginoso aumento en la movilidad laboral del sector de tecnología, por lo cual se ha hecho mas sensible tener definiciones claras para capturar en línea la documentación y el *know-how*.

Se espera que, con la documentación de las decisiones tomadas, así como de los resultados obtenidos en este trabajo de grado sirvan de guía para la implementación de las técnicas ISS en otras organizaciones sin importar su tamaño o sector económico.

## 7.2 TRABAJO FUTURO

Las restricciones temporales del proyecto de grado llevaron a acotar su alcance a algo realizable y que tuviera impacto en la organización. Este proyecto tiene un gran potencial y a continuación se propone una serie de actividades para seguir evolucionándolo:

- Crear un grupo de estudio junto al Comité ISS (ver **Desarrollo del gobierno ISS**) sobre los Patrones de Solución de situaciones comunes en la implementación de ISS (ver Tabla 3) para resolver los diferentes **Retos de las ISS**.
- Diseñar y aplicar modelos para medir el nivel de penetración o madurez de ISS en la organización. Entre estos, se puede tener en cuenta mediciones como: Apertura de los grupos/canales de Slack (que se haga público el conocimiento contenido en estos) y apertura de tableros de Jira (para combatir el tribalismo).
- Explorar junto a RRHH: 1) el patrón ISS de asignación de tiempo a voluntariado (ver Tabla 3), y desarrollar un esquema de medición del nivel de satisfacción laboral de los desarrolladores bajo un ambiente donde se pueda realizar voluntariado: ISS. 2) Implementar formas de reconocimiento del trabajo de voluntariado ISS en las mediciones de desempeño.
- Proponer modelos para medir la calidad de las tutorías (*mentorship*) de los aportes ISS. Se puede integrar con RRHH para brindar cursos focalizados de comunicación asertiva, empatía y liderazgo.
- Desarrollar software, scripts o proponer modelos que permitan aplicar mediciones sobre ISS como lo hizo la empresa Bosch (Cooper et al., 2018, págs. 47-62):
    - Adopción: aumento de proyectos, aumento de contribuidores, aumento de uso de proyectos por parte de otras células.
    - Diversificación de los grupos: averiguar cuántas personas diferentes los de su célula ha

logrado alguna persona conoce gracias al voluntariado en el RSC.
    - Mejora de la colaboración: rompimiento de silos (diversificación): nuevos proyectos creados entre diferentes células e intramovilidad[5].
    - Crecimiento personal. Esto se puede medir al preguntar 1) ¿Crees que la iniciativa te ha ayudado a crecer como profesional? 2) ¿Crees que la iniciativa te ha ayudado a aumentar tu satisfacción laboral?
    - Aumento de productividad. Se propone medir con preguntas como ¿crees que la iniciativa te ha ayudado a tener mayor visibilidad o a crear más impacto en la organización, que el que tendrías sin ella?
    - Alineación con el negocio: Una pregunta que pude realizarse sería: ¿consideras que de con esta iniciativa aportas más a las metas de la organización?
- Aplicar mediciones automatizadas sobre los aportes en los repositorios de código (por ejemplo GitHub) como lo propone Caparro (2020).

### 7.2.1 Posibles proyectos académicos

Las siguientes actividades, aunque son Trabajo Futuro del actual proyecto, representan una mayor complejidad, sobre todo a nivel de investigación académica, y podrían ser la base para un trabajo de grado de pre o posgrado.

- Investigar sobre como generar un manual de implementación de ISS para el área de Tecnología (tradicional) de la organización (sistemas legados, lenguajes deprecados, tecnologías arcanas o propietarias) que se adapte a su cultura, herramientas y realidad.
- Investigar como crear un manual de implementación de ISS para un grupo empresarial. Esto implicaría estudiar la cultura de las empresas hermanas y encontrar los puntos comunes.
- Proponer un marco para medir qué solución común es candidata para convertirse en un código abierto (*Open Source*) y también las cuestiones culturales tendientes a que haya este tipo de apertura.
- Administración del Talento Humano y el como hacer cambios en equipos de alto desempeño o con habilidades altamente calificadas, como es el caso de los desarrolladores de software. Otros temas desarrollar son: el voluntariado como factor de motivación o retención laboral de desarrolladores y también el posible cálculo de un esquema de medición y reconocimiento monetario y no monetario del voluntariado.
- Elegir un modelo de gestión del cambio (como el modelo de 8 pasos de Kotter) para analizar el estado del cambio en el que se encuentra la empresa con respecto al ISS y proponer los pasos concretos para lograr un cambio sostenido en la organización.

[5] Personas que han cambiado de célula.

# 8 BIBLIOGRAFÍA


Barros-Justo, J. L., Benitti, F. B. V., & Matalonga, S. (2019). Trends in software reuse research: A tertiary study. *Computer Standards and Interfaces*, *66*, 103352. https://doi.org/10.1016/j.csi.2019.04.011

Bonewald, S. (2017). *Understanding the InnerSource Checklist: How to Launch Collaboration Within Your Enterprise*. O'Reilly Media.

Capilla, R., Gallina, B., Cetina, C., & Favaro, J. (2019). Opportunities for software reuse in an uncertain world: From past to emerging trends. *Journal of Software: Evolution and Process*, *31*(8), e2217. https://doi.org/10.1002/smr.2217

Capraro, M. (2020). *Measuring Inner Source Collaboration*. Friedrich-Alexander-Universität Erlangen-Nürnberg.

Cooper, D., Stol, K. J., & Safari, an O. M. C. (2018). *Adopting InnerSource: Principles and Case Studies*. O'Reilly Media.

Edison, H., Carroll, N., Morgan, L., & Conboy, K. (2018). An investigation into inner source software development: Preliminary findings from a systematic literature review. *Proceedings of the 14th International Symposium on Open Collaboration, OpenSym 2018*, 1–10. https://doi.org/10.1145/3233391.3233529

Heskett, J. L. (2021). How long does it take to improve an organization's culture? *Harvard Business School Working Knowledge*. https://hbswk.hbs.edu/item/how-long-does-it-take-to-improve-an-organizations-culture

IEEE. (2010). IEEE Standard for Information Technology--System and Software Life Cycle Processes--Reuse Processes. *IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999)*, 1–51. https://doi.org/10.1109/IEEESTD.2010.5551093

ISC. (2021). *InnerSource Patterns*. InnerSource Commons. https://github.com/InnerSourceCommons/InnerSourcePatterns

Mittman, D. (2018). InnerSource at JPL: Collaboration around software in a science, technology, engineering and research enterprise. *Nasa - Jpl*.

Montilva, J., Arapé, N., & Colmenares, J. (2003). Desarrollo de software basado en componentes. *Actas Del IV. Congreso de Automatización y Control. Mérida, Venezuela*.

O'Reilly, T. (2000). *Keynote speaker: Tim O'Reilly---Open Source: The Model for Collaboration in the Age of the Internet*. Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions. https://web.archive.org/web/20150215185341/http://archive.oreilly.com/pub/a/oreilly/ask_tim/2000/opengl_1200.html

Oram, A. (2021). *Getting Started with InnerSource - Keys to collaboration and productivity inside your company* (2nd ed.). O'Reilly Media.

Piezunka, H., & Dahlander, L. (2014). Distant Search, Narrow Attention: How Crowding

Alters Organizations' Filtering of Suggestions in Crowdsourcing. *Academy of Management Journal*, *58*(3), 856–880. https://doi.org/10.5465/amj.2012.0458

Reddy, M. (2011). API design for c++. In *API Design for C++*. Elsevier Science. https://doi.org/10.1016/C2010-0-65832-9

Sommerville, I. (2011). *Ingeniería del software* (9th ed.). Pearson educación.

Stol, K. J., Babar, M. A., Avgeriou, P., & Fitzgerald, B. (2011). A comparative study of challenges in integrating Open Source Software and Inner Source Software. *Information and Software Technology*, *53*(12), 1319–1336. https://doi.org/10.1016/j.infsof.2011.06.007

Stol, K. J., & Fitzgerald, B. (2015). Inner Source - Adopting Open Source Development Practices in Organizations: A Tutorial. *IEEE Software*, *32*(4), 60–67. https://doi.org/10.1109/MS.2014.77

Vargas-Fandiño, J. C., Sandoval-Ramirez, J. J., & Vera-Rivera, F. H. (2020). Implementación de un repositorio para el catálogo, búsqueda y uso de componentes software reutilizables en el desarrollo de aplicaciones web. *Revista UIS Ingenierías*, *19*(2), 11–20. https://doi.org/10.18273/revuin.v19n2-2020002

# 9 ANEXOS

A continuación, se relacionan documentos y otros elementos que complementan el presente trabajo de grado.

## 9.1 ANEXO A: GOBIERNO

Esta sección presenta una copia de los documentos “vivos” (modificables) creados para el gobierno establecido y que es de libre acceso para todos los funcionarios que tienen una cuenta de Confluence (en teoría, todos los miembros del Laboratorio Digital). Algunos de los enlaces incluidos en el texto no funcionarán porque se dirigen a sitios privados o remiten a secciones dentro del mismo texto, de gobierno, en un formato que Confluence puede interpretar.

Esta sección del Trabajo de Grado está redactada en un lenguaje cercano a los desarrolladores en forma de tuteo, y que no sigue las normas de los documentos académicos como el lenguaje impersonal o el manejo de cursivas en palabras en idioma diferente al español; tampoco se hacen introducciones o explicaciones jerga técnica.

### Propósito

| |
|---|
| Contribuidores al momento de realizar el trabajo: Oscar Leonardo Bustos y Alexander Andrade |

El principal propósito de crear un repositorio de servicios comunes es incrementar la agilidad de las células del Laboratorio Digital de Banco de Occidente. Se ha detectado un número de funcionalidades repetidas, a lo largo de cada uno de los proyectos, y se espera que cuando un proyecto vaya a comenzar exista un inventario de servicios donde los nuevos proyectos puedan comenzar a construir sobre lo construido, como si se tratara de piezas de Lego como lo ejemplifica la Figura 3.

Figura 3. Ejemplificación de construcción con bloques de Lego®

Fuente: Tenor.

¿Quien es el responsable de los servicios comunes?
¿Como puedo aprovecharlos?
¿Cómo puedo aportar si tengo una idea de mejora?
No soy desarrollador, pero quiero contribuir ¿Que hago?

La respuesta a estas y otras preguntas son contestadas en el documento “Gobierno” o las puedes ver justo aquí abajo.

**Gobierno**

| Contribuidores al momento de realizar el trabajo: Alexander Andrade |
|---|

Para poder trabajar de forma efectiva y auto organizarnos es necesario que sentemos unos acuerdos comunes y sencillos. Los siguientes acuerdos están basados en unas técnicas llamadas InnerSource que se implementaron inicialmente en PayPal y hoy ayuda a muchas empresas como Microsoft, GitHub y Netflix, a crear la cultura colaborativa que permiten recibir ideas y aportes de todos los roles tanto en código de software, mejora de documentación y hasta entendimiento de los procesos de negocio.
Con este “Gobierno” se pretende facilitar no solo la reutilización de estos servicios dentro de nuestros proyectos, sino también el crecimiento de las personas y la creación de entornos de mentoría donde todos podamos aprender de todos.

### 9.1.1 Preguntas frecuentes.

#### 9.1.1.1 ¿Quienes pueden aportar a estos Servicios Comunes?

Cualquier rol del Laboratorio Digital.

#### 9.1.1.2 ¿Por que contribuir? ¿Que gano?

Experiencia y reconocimiento. Veamos unos ejemplos:

- Si eres un **desarrollador**: imagina que hiciste un curso del lenguaje de programación Go, podrías encontrar un proyecto al cual contribuir de forma real y aprender de las interacciones que tengas con los desarrolladores y arquitecto que hicieron el proyecto. También podrías tomar uno de los proyectos pendientes por realizar y empezar a hacerlo, desde el diseño de la arquitectura y hasta mas.
- Si eres de un **rol de negocio** u otro: imagina que hace falta documentar un servicio y esto te

lleve a dialogar con personas de otra célula para lograr comprenderlo, utilizar técnicas de levantamiento de información de una capacitación que viste en YouTube o incluso aprender mas detalle técnico de los servicios o hasta hacer diagramas de arquitectura de software.

- Cualquiera que haga las veces de revisor del trabajo de otros puede ganar **experiencia y reconocimiento como mentor**. Muchas veces quisiéramos estar en una posición de liderazgo para poder impactar la vida de los demás, este puede ser un gran entrenamiento en tu crecimiento personal y laboral.
- Por último, sin importar tu rol, lejos de ser altruistas (ayudar a los demás) el apoyar a un proyecto puede facilitar que se mejoren los estándares y hacer que en el futuro tu célula pueda beneficiarse del proyecto al que aportaste o incluso, tu célula se conecte a un nuevo servicio que tomó como referencias los aportes que tú hiciste.

#### 9.1.1.3 ¿Cómo aportar?

Puedes aportar a nivel de:

- Documentación:
  - Mejora de la documentación existente.
  - Investigación para completar la documentación.
  - Validación de la documentación (para ver si está completa y es entendible).
  - Levantamiento de nuevos requerimientos.
- Gestión de Calidad de los procesos de software:
  - Documentación de casos de prueba.
  - Automatización de pruebas.
  - Mejora a pruebas unitarias: aleatorización, cobertura de líneas de código, cobertura de casos de pruebas.
- Codificación:
  - Mejorar la calidad del código (SonarQube): Calidad de Código ADL .
  - Corrección de bugs.
  - Adición de nuevas funcionalidades.
  - Haciendo revisiones al código existente o verificando la calidad de los PRs. Para esto deberá seguir el instructivo seguir el instructivo Revisión de Código (Code Review) en ADL.
- Otros:
  - Solicitud de nuevas funcionalidades.
  - Solicitud de mejoras en los servicios.
  - Detección y redacción de hallazgos (Bugs).
  - Validar la pertinencia de las solicitudes de otros.
  - Retroalimentar a los demás de como hacer mejores aportes.

- Apadrinar la documentación de un servicio.
- Ideas y cualquier aporte, sin importar del tipo que sea, que ayude a mejorar los servicios comunes.

En cuanto a la codificación se espera que envíes un PR (*Pull Request*) siguiendo las buenas prácticas, dentro de las cuales destacamos:

- Aplicar la estrategia de *Commits* pequeños y atómicos: que contengan solo lo que dicen contener, que sirvan como un historial del desarrollo y permitan revertir fácilmente. Vea mas detalle en Recomendaciones para GitHub | Commits-atómicos.
- Antes de hacer un commit debería aplicar *linters* o autoformateadores de código, y hacer las correcciones que sugiera SonarQube, para mantener el historial lo mas sencillo de entender.
- Debería intentar enviar un único PR con la solución completa. Lo más recomendable es aplicarse a sí mismo una *code-review* y verificar que se cumpla con el Recomendaciones para GitHub | Código-de-conducta-del-contribuidor-de-código y el manual de Revisión de Código (Code Review) en ADL y adjunte un pantallazo del SonarQube local donde demuestre que se ha cumplido a cabalidad con todo el *Quailty Gate*: ver Calidad de Código ADL y cualquier otra definición de este texto, esto facilitará la labor de Revisión del PR y acelerará la aceptación de sus cambios.

Es recomendable que si tus modificaciones afectarán la lógica existente o los contratos (*requests* y *responses*) de un proyecto, contactes antes con la célula que más recientemente ha implementado el servicio para que puedan entregar información sobre posibles riesgos. En la documentación del servicio verá un listado con las células que implementan ese proyecto en particular.

#### 9.1.1.4 ¿Quien es el responsable de los servicios comunes?

Los responsables de los servicios comunes somos todos, pero por 30 días calendarios el soporte reposará en la célula o desarrollador que haya hecho el último *merge* en el ambiente productivo. Después de los 30 días calendarios cualquier funcionario del Laboratorio podrá contribuir al servicio mediante la solución del *bug* o hallazgo detectados o mejoras.
**Nota**: el soporte a los ambientes bajos cesará a partir de los 30 días calendarios posteriores al paso a producción, por lo cual la célula o desarrollador deberá trasladar rápidamente los cambios entre los ambientes bajos para no dar soporte ilimitado a los cambios que no haya promocionado a producción.

#### 9.1.1.5 ¿Cómo encuentro información de un proyecto cuyos realizadores ya no pueden contactarse?

La rotación de personal es algo natural en toda industria y no es extraño encontrar algún proyecto que iniciaron personas que ya no se encuentren laborando en la organización, por lo cual puedes utilizar algunas de las siguientes estrategias para suplir esta situación:

1. En la documentación hay una lista de células que utilizan ese servicio, intenta consultar con aquella que lo haya implementado mas recientemente.
2. Revisa en el código los nombres de los desarrolladores individuales que han hecho aportes. Si no los puedes contactar, investiga a que células pertenecen o pertenecían para encontrar apoyos.
3. Revisa las modificaciones de la documentación de *Confluence* para intentar contactar con estos contribuidores.
4. Ponte en contacto con los administradores del Laboratorio.

#### 9.1.1.6 ¿Cómo logro acceso al código de un proyecto al que me interesa aportar?

Para esto, normalmente debes tener rol de desarrollador y "Hefestarte" o pedirle al Arquitecto de tu célula que lo haga. Las personas de roles no técnicos podrían acceder a una copia de este mediante un desarrollador; no obstante, todas las personas que trabajamos en el Laboratorio Digital debemos conservar la privacidad de estos desarrollos dentro del Laboratorio, en caso contrario al compartir estos se configura una acción de desvelo de secreto industrial o incumplimiento del contrato laboral.

#### 9.1.1.7 ¿Cómo puedo reutilizar el código?

En adición al punto anterior, puedes:

1. Utilizar o consumir el servicio en tu célula.
2. Antes que desarrollar una copia para tu célula (*fork*), lo mejor es ver como puedes aportar al servicio actual de forma tal que todo el Laboratorio se beneficie.
3. Puedes utilizar parte de la lógica o como se resolvieron algunos temas en un servicio dentro del código del proyecto que tienes a cargo. No obstante, se recomienda estudiar las formas de llevarlo a una librería y hacer que tanto tu proyecto como el Servicio Común consuman este y así centralizar la lógica para que todos nos beneficiemos.

Nota: debes tener cuidado del uso fuera del Laboratorio del código o partes del mismo, como se menciona en la pregunta ¿Cómo logro acceso al código de un proyecto al que me interesa aportar?

#### 9.1.1.8 ¿Que hago si debo conectarme a un servicio común y la documentación es insuficiente?

La idea es hacer de este manejo algo comunitario, entonces la invitación es a que tomes cartas en el asunto y ayudes a completarla. Completar la documentación será más rápido y efectivo que hacer el desarrollo de nuevo.

#### 9.1.1.9 ¿Cómo atraer aportes a un proyecto?

Algunas estrategias que puedes utilizar son:

- Documentación completa.
- Que el código cuente con una gran *Quality-Gate*, en especial una cobertura cercana al 100 % Calidad de Código ADL.
- Información clara sobre "trabajos pendientes" en el proyecto en el tablero de Jira.

**Nota**: utilizar lenguajes de avanzada podría ser una buena estrategia para atraer aportes de desarrollo, pero también podría limitar el número de desarrolladores que conocen del mismo. Debes ser estratégico al escoger el lenguaje a utilizar.

#### 9.1.1.10 ¿Cómo garantizo que los aportes que hacen al proyecto tienen calidad?

Es muy sencillo. Debes cuestionarle al contribuidor si aplicó todas las recomendaciones sugeridas en este documento. Cuando el contribuidor las haya aplicado, debes hacerle *review* a PR y si el aporte resulta no estar a la altura de los esperado, debes ponerte en contacto, preferiblemente a través de una llamada o en persona y brindarle la menoría necesaria para que aprenda de la experiencia y ganes un nuevo aliado en tu proyecto. **Estas mentorías se presentan como una oportunidad** para que el revisor practique este tipo de habilidad y también para recibir nuevas ideas, pues un mal entendimiento por parte del contribuidor puede dar pistas sobre mejora en la documentación o incluso que un proceso puede ser mejor modularizado.
En caso de que definitivamente debas rechazar el aporte, recuerda que estamos a la espera de recibir trabajo voluntario de otros profesionales y como tal debes ser muy humano para lograr explicar la situación y que ambas partes salgan fortalecidas del proceso: el revisor como mentor y el contribuidor como un mejor profesional. Al final, deben asegurarse entre ambos que la mentoría quede plasmada en los comentarios del PBI (*Product Backlog Item* o Historia) para que en el futuro el mismo contribuidor o uno nuevo no se vean en la tentación de insistir sobre el mismo caso.

#### 9.1.1.11 ¿Que hago si un aporte que realicé se rechazó?

Primero, debes solicitar la mentoría a quien rechazó el aporte para que se logres comprender en que debes mejorar, y entre los dos documentar la situación para que en el futuro otros puedan aprovechar esta experiencia para crecer. Esta "documentación" debe quedar como un comentario dentro del PBI (Product Backlog Ítem o Historia) de Jira para ser revisado en el siguiente aporte que realices. Estas mentorías escritas serán de gran utilidad para todos.

#### 9.1.1.12 ¿Cómo inicio un nuevo proyecto?

1. Identifica el tipo de proyecto que realizará: proxy (que va a un tercero, por ejemplo el Banco) o utilitario (que resuelve una situación sin conexiones fuera del Laboratorio).
2. Verifica en la lista de proyectos pendientes de acuerdo con el tipo.
3. Notifica el cambio de estado en la lista de proyectos pendientes o agregarlo si es totalmente nuevo.
4. Inicia la documentación en este espacio de *Confluence*, basándose en uno de los proyectos existentes. Debe evitar eliminar las secciones establecidas. Si considera que es importante agregar una nueva sección de documentación, debe hablarlo con la Administración del Laboratorio, para ver si se realiza la misma mejora en el resto de los proyectos.
5. Haz uso de la metodología *Design-First* (o API-*First*) por sobre *Code-First* (empezar a teclear código sin un plan claro). Esto quiere decir no solo diagramar antes la Arquitectura sino también diseñar el contrato (los *request* y *responses*) del servicio mediante el estándar Open API (Swagger) y teniendo en cuenta la definición de requests y responses del Laboratorio, así como el buen uso de los Verbos Http (*GET, POST, PUT, PATCH* y *DELETE*) y *Http Statuses* correctos. Todo lo anterior, permite, entre muchas otras ventajas, que las células que van a usar el servicio puedan desarrollar en paralelo mediante *mocks* y además el Swagger puede autogenerar el código fuente del servidor y de los clientes, lo cual aumenta más la velocidad.

Dependiendo del tipo de servicio común, deberás tener en cuenta en su diseño los requerimientos no-funcionales relacionados en la Tabla 10.

Tabla 10. Requerimientos no-funcionales de los servicios del RSC

| Requerimiento no funcional | Servicios utilitarios | Servicios proxy |
|---|---|---|
| ***Stateless***: Para permitir la escalabilidad, los servicios deben ser sin estado en lo posible. | X | X |

| Requerimiento no funcional | Servicios utilitarios | Servicios proxy |
|---|---|---|
| **Asíncronos**: Los utilitarios deberían permitir invocarse de modo asíncrono en lo posible para permitir la escalabilidad. | X | |
| ***Serverless***: Para minimizar el impacto en el gasto y en la gestión del servicio se recomienda en lo posible que los utilitarios no usen servidor. | X | X |
| **Genéricos**: Deben ser agnósticos al consumidor en lo posible. | X | |
| ***Observabilidad***: Deben ser llamados con información que permita la trazabilidad del mensaje y diagnosticar errores fácilmente. También debe llevar registros de que célula consume el servicio. | X | X |
| **Gestión de Llaves**: Dado que estos servicios se comunican de forma segura con servicios externos a través de certificados, estos deben poder actualizarse fácilmente desde un repositorio unificado. Verificar el método común de gestión de certificados. | | X |
| ***Circuit Breaker***: Estos servicios deben permitir proteger al servicio destino cuando el externo responda *timeout*. | | X |
| ***Throttle Pattern***: Estos servicios deben permitir proteger al servicio destino cuando los TPS máximos para el externo se hayan superado. | | X |
| **Responsabilidad contenida**: se excluyen servicios nivel 3 (de utilidad para todas las entidades del grupo, por ejemplo: consulta de seguridad social) que van directamente a terceros. | | X |
| **Autonomía**: los servicios proxy también deben entregar un *mock* del servicio del tercero, para usar o probar contra este en caso de que la conexión o servicio externo no estén funcionando en el momento y reducir la afectación de tiempos de la célula que desea implementarlo. | | X |
| **Estandarización de contratos**: garantizar que los contratos de servicio dentro de un inventario de servicios (empresa o dominio) se adhieran al mismo conjunto de estándares de diseño. | X | X |

#### 9.1.1.13 ¿Cómo documento los cambios en un proyecto (*Changelog*)?

A manera de *Changelog* o control de cambios, debes integrar los *Release Plans* (los planes de despliegue o salida a producción) como una hoja "hija" de la documentación del proyecto. Si el desarrollo de uno de estos proyectos se hizo dentro de los trabajos de tu célula, debes copiar del *Release Plan* solo los aspectos relacionados con el servicio común.
En este enlace puedes obtener la plantilla (*template*) de *release plan* mas reciente Release Plan Template (Actual)
En cuanto al versionamiento del servicio, más adelante encontrarás información detallada

de como gestionarlo.

#### 9.1.1.14 ¿Cómo versiono los cambios en un servicio?

El versionamiento se hace bajo el estándar "Versionamiento semántico", también conocido como "Versionamiento X.Y.Z", que a grandes rasgos es:

- X → Es el número de la mayor versión, es incrementada cuando se agrega una nueva funcionalidad del producto, por lo general cuando hay un incremento de valor en el producto se incrementa este número y al mismo tiempo cambia el valor de la mejora u optimización y el *Fix version*.
- Y → Es el número de la mejora u optimización (también conocida como menor versión), cuando se cuenta con una funcionalidad existente en producción y esta sufre un cambio por mejora u optimización, esta variable se incrementa, reiniciando el valor de *fix version* a cero.
- Z → Es el número del *Fix version*, representa el número de *bug* que se esta resolviendo luego de la mejora o salida de nueva funcionalidad, cada vez que se va a salir con corrección de bugs a producción este valor se incrementa manteniendo intacto los valores de mayor versión y mejora u optimización.

En la Figura 4 se muestra visualmente como se realiza el versionamiento semántico:

Figura 4. Ejemplo de versionamiento semántico


Puede ver más información en Versionado Semántico 2.0.0

### 9.1.2 Seguridad de los servicios comunes – *Token*.

| Contribuidores al momento de realizar el trabajo: Alexander Andrade y Juan Fernando Castro |
|---|

Cada producto del Banco de Occidente deberá crear sus credenciales únicas de acceso al *pool* de conexiones en Commons mediante un *App Client*. Así, al obtener el

bocc_product_client_id y el bocc_product_client_secret se podrá realizar el consumo y recibir el access_token.

Como guía, debes solicitar unas credenciales de Cognito con los siguientes *scopes*, a no ser que en la documentación del servicio se indique lo contrario:

| Se oculta la imagen por poseer información sensible. |
|---|

A continuación, se relacionan las URLs del *token* para cada ambiente:

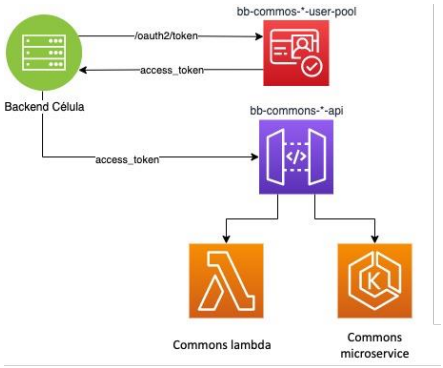
- DEV: https://xxx.auth.zone.amazoncognito.com/oauth2/token?grant_type=client_credentials
- STG: https://xxx.auth.zone.amazoncognito.com/oauth2/token?grant_type=client_credentials
- PRO: https://xxx.auth.zone.amazoncognito.com/oauth2/token?grant_type=client_credentials

La siguiente consulta / *request* de CuRL te permitirá crear un *token* una vez tengas unas credenciales válidas.

```
curl --location --
request POST 'https://XXX.auth.ZONE.amazoncognito.com/oauth2/token?grant_type=client_credentials' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'client_id=bocc_product_client_id' \
--data-urlencode 'client_secret=bocc_product_client_secret'
```

En cuanto al consumo de los servicios comunes desde tu proyecto, la Figura 5 muestra un esquema de consumo, que es realizado por **API GATEWAY**, el cual valida el cuerpo de la petición con un esquema de tipo JSON y que la cabecera de *Authentication* tenga un *token* tipo Bearer obtenido desde el servicio de autenticación de usuarios **COGNITO**.

Figura 5. Esquema de consumo seguro de servicios por API-Gateway




### 9.1.3 Estándar de Request / Response / Errores.

| Contribuidores al momento de realizar el trabajo: Raúl Rojas |
|---|

Las respuestas generadas por los servicios de la nube común se enmarcan en el uso de **códigos http** como primera capa para identificar el tipo de respuesta, siendo el código **200** una **respuesta exitosa**, el código **406** un **error de negocio** y el código **500** un **error operacional**.

Adicionalmente en los códigos 406 y 500 se recibe como respuesta un objeto con los valores *code* y *message*, los cuales me permiten obtener un mayor detalle que puede identificarse en la siguiente tabla, de acuerdo con el servicio. No debería existir dos códigos iguales de negocio, independientemente de que la invocación se haga desde servicios diferentes.

| Nota: esta información es altamente sensible, por lo cual no se muestra su detalle. |
|---|

### 9.1.4 Documentación API Swagger.

| Contribuidores al momento de realizar el trabajo: Alexander Andrade |
|---|

Para documentar los servicios deberás utilizar el estándar Open API (antes conocido como API Swagger). En resumen, te permite detallar los *request*, *responses* y la información de cada campo como descripción, tipo de datos y si es obligatorio o no. También te permite adjuntar ejemplos de data y ejemplos de respuestas obtenidas.

Open API (OAS) es un *framework* para documentar APIs Rest desde muy diferentes fuentes: Archivos de configuración, XML, C#, Javascript, Ruby, PHP, Java, Scala… además AWS puede

exportar rápidamente esta documentación desde el API Gateway.
La especificación Open API (OAS: Open API *Specification*) se define a sí misma como una interfaz estándar independiente del lenguaje para las API de tipo RESTful que permite que tanto los humanos como las computadoras descubran y comprendan las capacidades del servicio sin acceso al código fuente, documentación o mediante la inspección del tráfico de la red. Cuando se define correctamente, un consumidor puede comprender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación.
La especificación Open API te permitirá, entre otras:

- Indicar los *endpoints* para cada ambiente
- Documentar los request y responses indicando el tipo de cada campo, si es requerido u opcional y poner una descripción para cada uno.
- Poner ejemplos de consultas y ejemplos de cada respuesta, no solo la documentación técnica mencionada en el punto anterior.
- Realizar pruebas sobre el servicio, como si fuera un Postman, desde el navegador e incluso desde la misma documentación en Confluence.
- Autogenerar el código de cliente (quien consume) o servidor (el servicio) para diferentes lenguajes de programación. Esta plantilla generada tendrá los request y responses con sus validaciones de tipos y solo haría falta agregar la lógica.
- Importar el archivo YAML a Postman, para consumirlo desde allí; o a CastleMock para crear rápidamente un mock del servicio y adelantar desarrollo en paralelo.

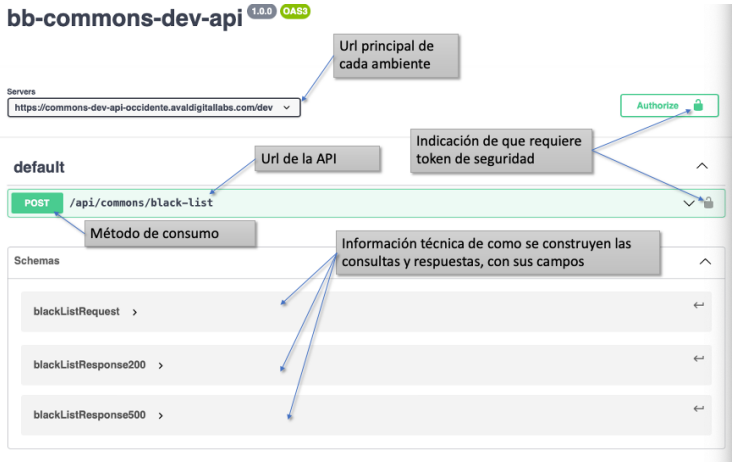
#### 9.1.4.1 Como leer un Swagger.

En el renglón de abajo encontrarás un Swagger incrustado en esta documentación. Puedes manipular sus opciones a conveniencia y sin riesgos. Mas abajo se explicará, mediante pantallazos, como leerlos de forma efectiva.

#### 9.1.4.2 Instructivo de lectura.

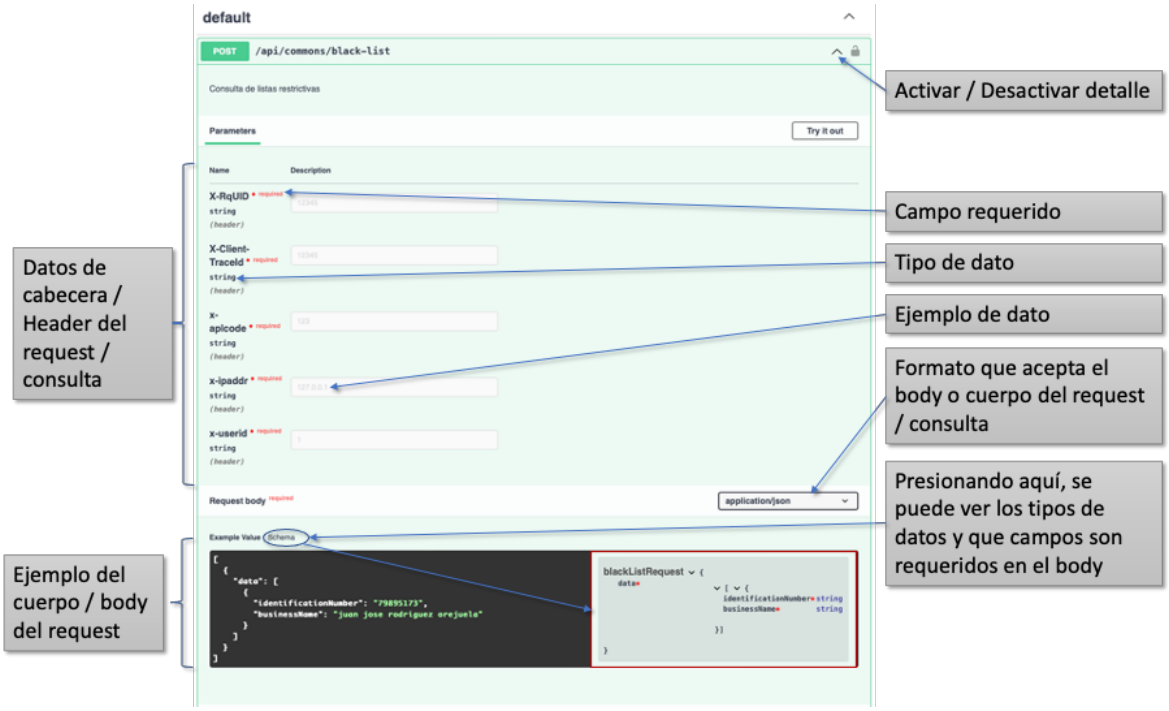
Para guiarte con un Swagger debes tener claro la información que contiene este, como lo registra la Figura 6:

Figura 6. Información básica de Open API / Swagger




Si entras a evaluar por ejemplo el método POST en su detalle podrás ver el detalle de como se realiza una consulta / *request*, como lo muestra la Figura 7.

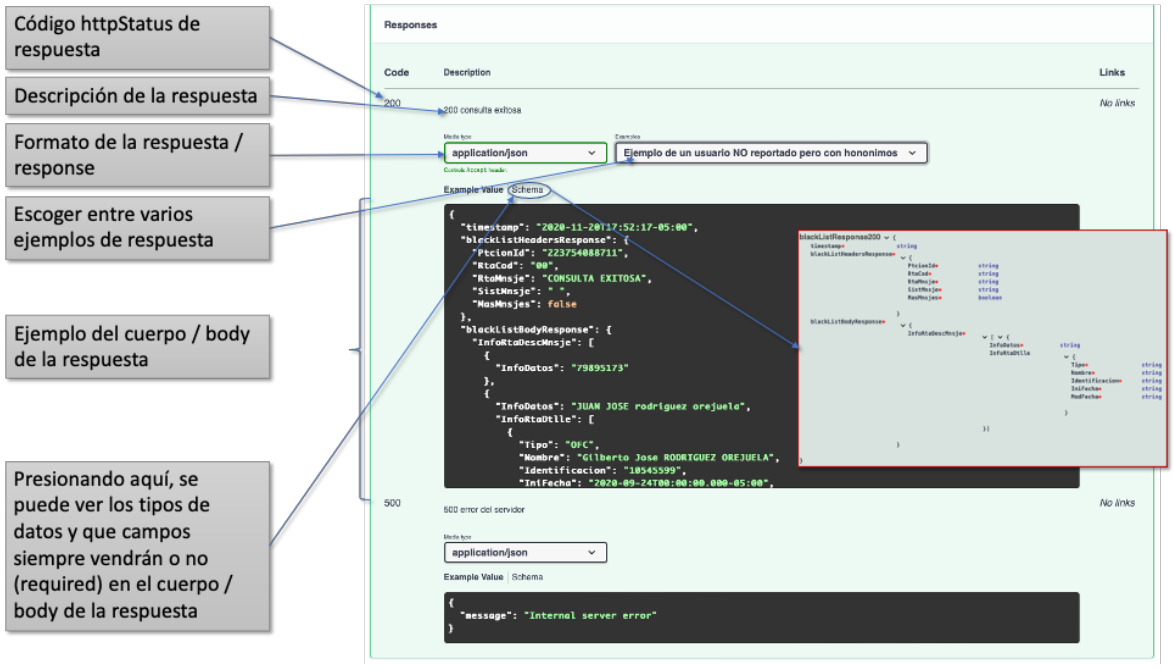
Figura 7. Ejemplo del *request* de una API en Swagger / Open API




Al igual que el *request*, puedes encontrar información detallada sobre como contestará (*response*) el servicio, sus campos, sus tipos de datos, cuales siempre vendrán en la respuesta (*required*) o son opcionales y hasta ejemplos de diferentes respuestas como lo detalla la Figura 8. Para los http-*statuses*, se debe tener en cuenta el instructivo Estándar

de Request / Responses / Errores.

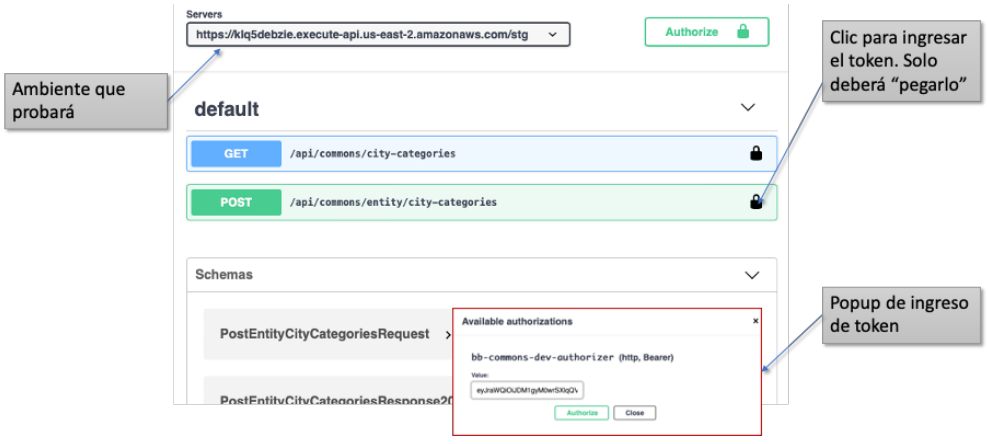
Figura 8. Ejemplo del *response* de una API en Swagger / Open API




### 9.1.4.3 Cómo probar el servicio desde un Swagger.

Para esto, deberás haber generado un token de servicios comunes como primer paso. Una vez tengas el *token*, podrás ingresarlo en el Swagger como se muestra en la Figura 9:

Figura 9. Cómo consumir una API desde un Swagger / Open API




Una vez ingresado el *token*, puedes utilizar los valores de ejemplo o modificarlos a tu conveniencia como lo muestra la Figura 10:

Figura 10. Valores del request para consumir una API desde Swagger / Open API


Tras la ejecución aparecerá el resultado. Por temas de seguridad, los servicios de ADL no se pueden consultar desde un Swagger, pero podrías utilizar el CuRL generado para consultarlo por otros medios (la terminal de tu equipo, Postman y otro). En la Figura 11, en el lado derecho se muestra como se vería la respuesta de un servicio.

Figura 11. Ejemplo de respuesta/response de una API consumida desde un Swagger


### 9.1.4.4 Cómo documentar una API con Swagger.

**Documentar una API existente**

Este es un paso que poco se usará, debido a que se ha establecido que los Servicios Comunes se desarrollarán mediante metodología Design-First no obstante, aquí se te explica como conocimiento general.
El API Gateway de AWS permite exportar el Swagger de todos los servicios que tiene expuestos. Será necesario que hayas hecho *login* en AWS mediante línea de comandos y luego ejecutes el siguiente comando

```
$ aws apigateway get-export --rest-api-id XXXXXXXX --stage-name dev --export-type swagger swagger.json
```

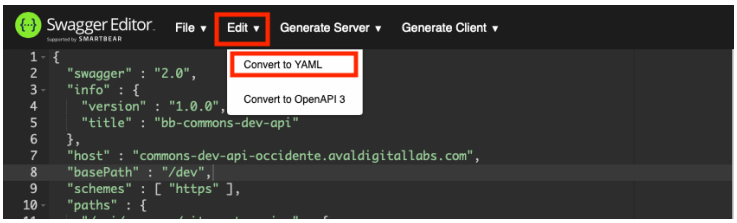
Este comando exporta el Swagger en formato JSON, por estándar debes convertirlo en YAML como se muestra en el título anterior. En el comando anterior, el código en la opción "rest-api-id" hace referencia al API Gateway por exportar.
El siguiente libro (de 11 páginas, en inglés) te entregará mayor detalle de como documentar una API existente.

**Como mejorar o completar un Swagger**

Para editar o mejorar el Swagger de la aplicación podrás utilizar el editor online Swagger Editor, que de paso va validando la información; este editor es gratuito y no requiere

registro. Es necesario que la información esté en formato YAML y no en JSON para que pueda ser utilizada por otras herramientas como CastleMock. Para convertir un Swagger en formato JSON a YAML, en el **Swagger Editor**, usa el menú "*Edit*" > "*Convert*" to YAML como se muestra en la Figura 12:

Figura 12. Cómo exportar a YAML desde Swagger Editor



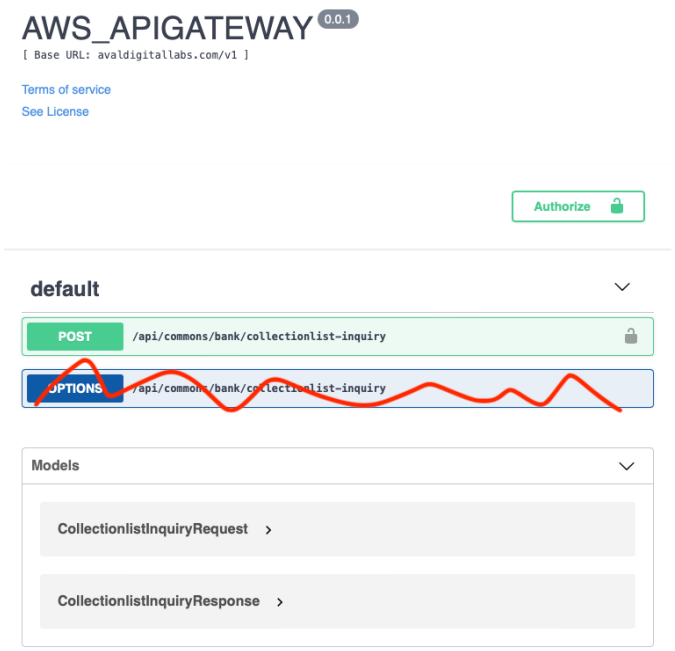

Si cierra el Swagger Editor, al volver al editor este recobrará el trabajo donde ibas, no se perderá. Es importante que al finalizar la edición guardes el archivo YAML en tu computador para poder importarlo luego a Confluence. Para exportar el Swagger puedes copiarlo y pegarlo en un archivo o puedes utilizar el menú "*File*" > "*Save as*", del Swagger Editor.
Puedes tomar como ejemplo el siguiente adjunto para guiarte al momento de crear un Swagger complejo.

| En este lugar iría el archivo adjunto a descargar |
|---|

**Recomendaciones al construir un Swagger**

De la documentación del Swagger se recomienda que elimines la información de los métodos *OPTIONS* para hacerla mas sencilla de entender como lo muestra la Figura 13.

Figura 13. Simplificar la documentación de API eliminando los OPTIONS




En esta documentación de Swagger debes, además:

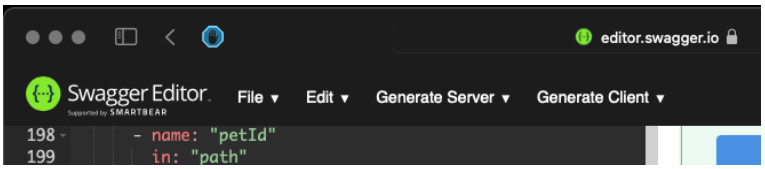
- En la consulta (request) especifica que campos son requeridos tanto en la cabecera (*headers*) como cuerpo (*body*).
- En la respuesta (*response*) especifica como "requeridos" los campos que siempre se devuelven. Se entenderá como opcionales (que a veces se responden) aquellos que no aparezcan como requeridos.
- Evita colocar en el *response* los *headers* que son naturales al servicio, como "Access-Control-Allow-Headers" y cualquier otro que no sea aprovechable por los consumidores del servicio. Esto mantiene la documentación mas limpia y fácil de entender.
- Describir cada campo y adjuntar como mínimo un ejemplo.
- Si hay varios tipos de respuesta para un mismo código httpStatus, debes colocar varios ejemplos. Por ejemplo, para la respuesta httpStatus=200 (respuesta exitosa):
    - Usuario sin novedad.
    - Usuario con novedad.
- Debes colocar los ejemplos en la sección de métodos (GET, POST, PUT, etc.) y no en la de *components*/*schema*. Esto produce una documentación más limpia y fácil de entender.
- Debes incluir en el Swagger todos los tipos de repuesta que podrá ofrecer el servicio, así como sus ejemplos.
- Debes tener en cuenta el instructivo Estándar de Request / Responses / Errores.
- Para el manejo del *bearer token* no debes crear un *header*, llamado "*Authorization*". Debes manejarlos mediante una estructura específica llamada "*security*", la cual podrás copiar del

Swagger de ejemplo que hay en la parte inicial de esta documentación.

#### 9.1.4.5 Autogenerar el código fuente del servicio (servidor) y cliente (consumidor).

Otra ventaja de hacer los proyectos mediante la metodología *Design-First*, además de que se puedan hacer los desarrollos de los clientes en paralelo, es que se puede autogenerar el código fuente del servicio y de los clientes. Este código resultante tendrá documentación incluida, posee validaciones para los tipos de datos y otras restricciones como cantidad de caracteres, y solo haría falta agregar la lógica propia del caso de uso.
Para autogenerar el código, debes utilizar en el Swagger Editor los menús de "*Generate Server*" o "*Generate Client*", como lo muestra la Figura 14.

Figura 14. Menús del Swagger Editor




**Lenguajes de servidor autogenerados**

Al momento de escribir esta guía, el Swagger Editor podía generar más de 32 lenguajes, entre los que destacan: Node, Java (varios tipos y *frameworks* como Spring), Go, Kotlin, Python y Rust, entre otros. En la Figura 15 se muestra una lista de estos lenguajes:

Figura 15. Lista de lenguajes autogenerados por Swagger Editor para servidores de API

| | | |
|---|---|---|
| ada-server | jaxrs-cxf-cdi | python-flask |
| aspnetcore | jaxrs-resteasy | rails5 |
| erlang-server | jaxrs-resteasy-eap | restbed |
| finch | jaxrs-spec | rust-server |
| go-server | kotlin-server | scala-lagom-server |
| haskell | lumen | scalatra |
| inflector | msf4j | sinatra |
| java-pkmst | nancyfx | slim |
| java-play-framework | nodejs-server | spring |
| java-vertx | php-silex | undertow |
| jaxrs | php-symfony | ze-ph |
| jaxrs-cxf | pistache-server | |

## Lenguajes de cliente autogenerados

Al momento de escribir esta guía, el Swagger Editor podía generar más de 57 lenguajes, entre los que destacan: Dart, Go, Html, Java, JavaScript, Kotlin, Python, Swift y TypeScript, entre otros. En la Figura 16 se muestra una lista de estos lenguajes:

Figura 16. Lista de lenguajes autogenerados por Swagger Editor para clientes de API

| | | | | |
|---|---|---|---|---|
| ada | dynamic-html | javascript | r | tizen |
| akka-scala | eiffel | javascript-closure-angular | ruby | typescript-angular |
| android | elixir | jaxrs-cxf-client | rust | typescript-angularjs |
| apex | elm | jmeter | scala | typescript-aurelia |
| bash | erlang-client | kotlin | scala-gatling | typescript-fetch |
| clojure | flash | lua | scalaz | typescript-inversify |
| cpprest | go | objc | swagger | typescript-jquery |
| csharp | groovy | perl | swagger-yaml | typescript-node |
| csharp-dotnet2 | haskell-http-client | php | swift | ue4cpp |
| cwiki | html | powershell | swift3 | |
| dart | html2 | python | swift4 | |
| dart-jaguar | java | qt5cpp | swift5 | |

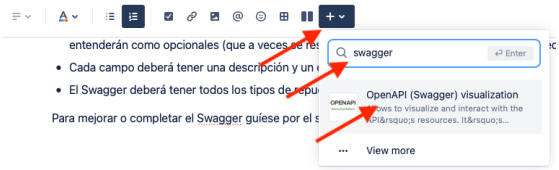
### 9.1.4.6 Cómo importar un archivo Swagger.

#### Cargar el Swagger a la documentación

Para cargar el Swagger a la documentación, o sea *Confluence*, debes hacerlo de dos formas: incrustado y adjunto. A continuación, se te indica como hacer cada una:

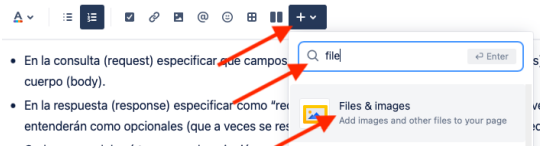
1. Agrega a la documentación un objeto de Swagger como se ve en la Figura 17:

Figura 17. Cómo incrustar un Swagger / Open API dentro de un documento de Confluence




2. Importa el archivo YML al *Confluence* como archivo adjunto. Esta segunda parte es importante poder descargar el Swagger e importarlo a CastleMock o Postman. Para esto, sigue los pasos que se muestran en la Figura 18:

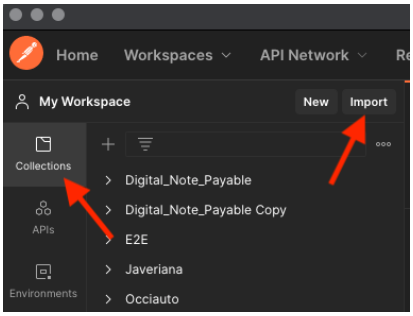
Figura 18. Cómo adjuntar un archivo o imagen a la documentación de Confluence




#### Importar el Swagger a Postman

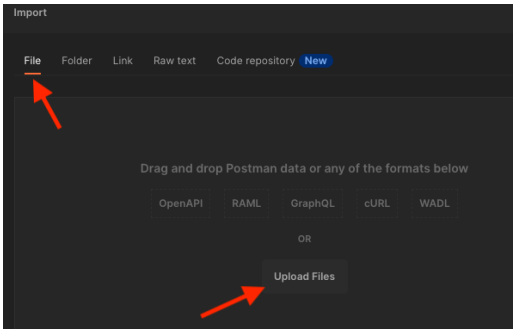
Para importar el Swagger en Postman, ve hasta la sección *Collections*/Colecciones y allí presiona el botón "*Import*/Importar" como se muestra en la Figura 19:

Figura 19. Cómo importar colecciones a Postman



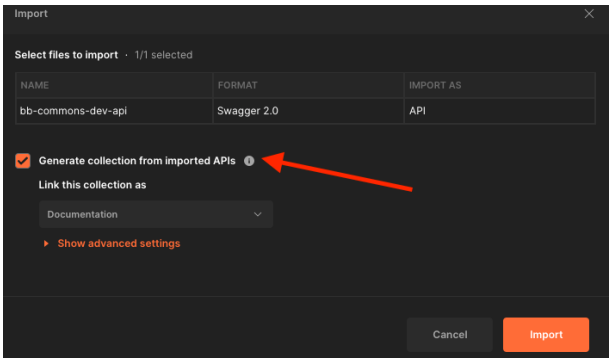

Aparecerá un diálogo, como se muestra en la Figura 20, en el cual verificarás que se encuentre en la pestaña "*File*/Archivo", luego presiona el botón "*Upload Files* / Cargar archivos".

Figura 20. Cómo cargar un archivo al proceso de importación de Postman




Al seleccionar el archivo YAML te aparecerá el cuadro de diálogo que se muestra en la Figura 21, fíjate que esté seleccionada la opción de "*Generate collection…* / Generar colección…" y presiona el botón "Import / Importar".

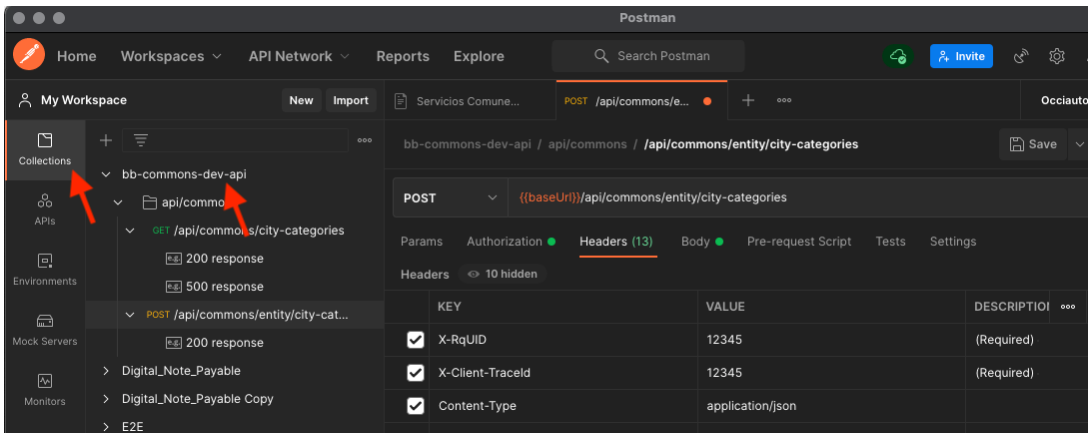
Figura 21. Cómo configurar los detalles de la importación a Postman




Encontrarás que Postman, como se muestra en la Figura 22, creó una nueva colección y esta

trae data de ejemplo para consumir la API, solo deberás tener presente incluir la información correcta de *token*.

Figura 22. Ejemplo de una colección recién importada a Postman

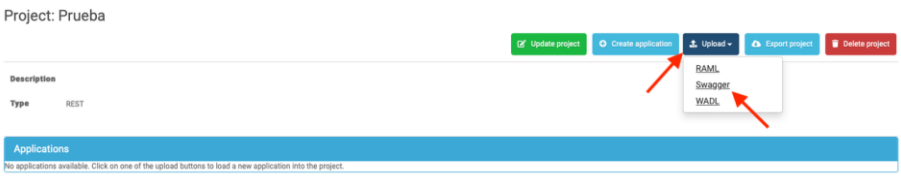



## Importar el Swagger a CastleMock

A veces querrás realizar *mocks* de las API, quizás porque no estén funcionando o aún no obtienes acceso a la data de prueba o los *tokens* de conexión. CastleMock te permite importar el archivo y crear al tiempo las respuestas con los ejemplos incluidos en el Swagger.

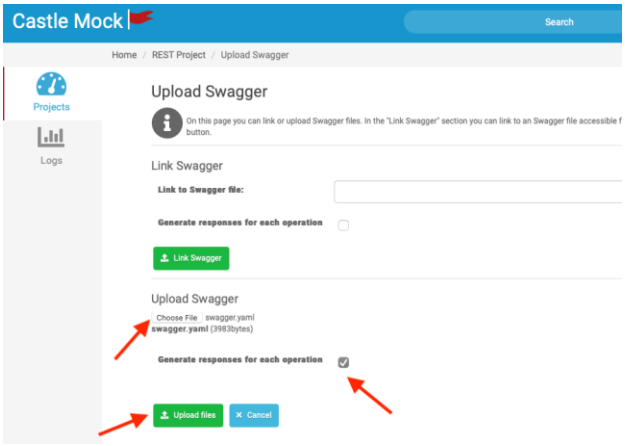
Para esto, en CastleMock crea un nuevo proyecto y luego utiliza el botón “Upload / Cargar”, escoge Swagger de las opciones que aparecen, como se ve en la Figura 23:

Figura 23. Cómo importar archivos a Castlemock




En la nueva pantalla que aparece, como lo detalla la Figura 24, asegúrate de cargar el archivo y seleccionar la opción que te permite generar todas las respuestas (quemadas) para los ejemplos detallados en el Swagger.

Figura 24. Cómo configurar los detalles de la importación a Castlemock




### 9.1.5 *Mocks* para servicios externos.

| Contribuidores al momento de realizar el trabajo: Alexander Andrade |
|---|

#### 9.1.5.1 ¿Por qué *mocks*?

En los servicios de tipo proxy, que van a otros servicios externos, los *mocks* representan una buena estrategia para acelerar la velocidad de desarrollo, pruebas y la implementación del servicio en otras células. Lo anterior es debido a que como no controlamos la infraestructura del servicio externo, un fallo o cambio en cualquier punto (los túneles VPN, los *gateways*, los datapowers, la aplicación, vencimiento de certificados, su base de datos o los terceros a que esta se conecten) podrían atrasar días o semanas la implementación del servicio. Al tener *mocks*, por un lado, sus respuestas son mas rápidas porque están en tu red y por el otro lado, puedes iniciar a desarrollar incluso a la par que se desarrolla el otro servicio o mientras este se recupera de un fallo.

#### 9.1.5.2 ¿Los *mocks* presentan desventajas?

Sí, pero…

Hacer un *mock* de un servicio externo te garantiza hasta un 99 % de probabilidad que al salir a producción no tengas problemas, no obstante, existe un 1 % de margen de error, pero este no es atribuible a la tecnología o concepto de los *mocks* sino al proceso de ingeniería de desarrollo como:

- Mala planificación: implementar cambios en los *request* o responses del servicio sin verificar

como impacta esto a los que lo consumen.

- Mala gestión del riesgo: relacionado al anterior, no anticipar los cambios que necesitan hacer las partes para ajustarse a la nueva forma de respuesta.
- No aplicar ingeniería de microservicios como el "*Design-First*": diseñar y socializar (con los usuarios: nosotros) antes de realizar los cambios.
- Mala documentación de los contratos de los microservicios: no confundir con contratos legales. Se trata de que se documente muy bien que elementos componen el *request* y response, indicar de que tipo de dato es cada campo, para que se usa y si es obligatorio o no. Nota: lo anterior es una simplificación, pero incluso hay empresas que trabajan con metodologías *Contract Testing* en el cual nunca hacen esfuerzos por conectarse a los ambientes bajos de un servicio, confían en el contrato.
- Si utilizas un Castlemock comunitario o compartido con otro proyecto, podrían modificar o borrar tu *mock*. Lo mejor es exportar el proyecto, una vez has terminado de desarrollar. Se recomienda que si tienes muchos *mocks* en tu proyecto crees para ellos un repositorio (tradicional, no Hefestado) en el GitHub corporativo.

#### 9.1.5.3 ¿Cómo construyo rápidamente un *mock* de un servicio externo?

La primera estrategia es utilizar el contrato (WSDL para los SOAP o Swagger para la mayoría de servicios) e importarlos en CastleMock como se muestra aquí Documentación API Swagger | Importar-el-Swagger-a-CastleMock, estas instrucciones sirven también para los WSDL.
Si tienes un contrato del servicio, situación que solo debe presentarse si el servicio no ha superado la etapa de diseño, la célula o tercero deben proveerte algún documento donde se han diseñado las diferentes respuestas del servicio y deberás crearlas una a una en CastleMock. Las principales recomendaciones al momento de construir *mocks* son:

1. Que cada servicio esté en un proyecto independiente.
2. Cuidar que el *mock* devuelva los Http Statuses correctos.

### 9.1.6 Aseguramiento de calidad.

#### 9.1.6.1 Pruebas unitarias.

| Contribuidores al momento de realizar el trabajo: Alexander Andrade |
|---|

En esta sección se recopilarán algunas buenas prácticas relacionadas con las pruebas unitarias y la cobertura de líneas de código. Es importante precisar que estos principios también aplican a otros tipos de pruebas automatizadas como las regresiones.

**Aleatorización de pruebas unitarias**

Las pruebas unitarias (*Unit tests*) de software representan la red de seguridad que nos permite manipular con confianza y velocidad el software de forma tal que podamos cumplir con las necesidades del negocio y las corrientes de Agilismo y DevOps. No obstante, estas también presentan una serie de anti-patrones y uno de ellos está asociado al descuido de ejecutarlas en orden determinístico (preestablecido o implícito), creando dependencias ocultas entre ellas.

¿Alguna vez has visto pasar una prueba unitaria de forma aislada, pero que falla cuando se ejecuta en una suite? O viceversa.

Aleatorizar el orden de las pruebas nos ayudará a eliminar los errores en el diseño de nuestras pruebas.

Comencemos por examinar un síntoma … ¿Alguna vez ha visto nombres como este en un conjunto de pruebas?

*Func test01RunThisTestFirst() { … }*

*func testCheckSomethingElse() { … }*

Este es un truco sucio para hacer que la prueba superior se ejecute primero, antes que cualquier otra prueba en la suite, se basa en el hecho de que actualmente, las **pruebas dentro de una suite se ejecutan en orden lexicográfico,** así 01 viene al principio de ASCII, por lo que test01RunThisTestFirst se ejecutará primero.

**¿Por qué recurrimos a estos trucos?**

¿Qué nos impulsa a hacer esto en primer lugar? Esto sucede porque esa primera prueba produce algún tipo de efecto secundario que queremos que se aplique a las otras pruebas. Las pruebas unitarias no se ejecutan en un vacío total. Puede haber un estado mutable compartido oculto en:

- Una preocupación transversal, como la analítica
- El sistema de archivos
- Valores predeterminados de usuario
- Persistencia de datos: una base de datos local o remota

Cuando se ejecuta una prueba, puede dejar efectos secundarios; hay una fuerte tentación

a utilizar estos efectos secundarios como punto de partida para la próxima prueba.

**¿Cómo nos afecta el orden de la prueba?**

No te equivoques: agregar caracteres a los nombres de prueba para forzarlos a un orden particular es un truco. Una prueba no debería influir en otra prueba. Las pruebas unitarias deben seguir el principio *FIRST*:

- *Fast*: rápidas.
- *Isolated*: aisladas / independientes.
- *Repeatable*: repetibles.
- *Self-validating*: autovalidadas.
- *Timely*: escritas "a tiempo", junto al código, justo antes o justo después, dependiendo de la estrategia, por ejemplo, TDD.

Para el caso del desarrollo de nuestras pruebas unitarias, se hace énfasis en la R de Repetible. "No deben depender de ningún estado inicial asumido, no deben dejar ningún residuo que impida que se vuelvan a ejecutar".
Esto es importante porque las pruebas no son un sistema fijo. Agregamos nuevas pruebas. Eliminamos pruebas que ya no aportan valor. Las ejecutamos individualmente. Es importante que los casos de prueba internos de una suite sigan siendo independientes para evitar problemas a medida que nuestras suites crecen y cambian.
Si las pruebas siempre se ejecutan en el mismo orden, 82el archivo "testA" viene antes que "testB", este orden implícito de los casos de prueba significa que creamos dependencias entre las pruebas y ni siquiera notarlo. Por lo general, cuando nos encontramos con el problema del orden de ejecución implícito, generalmente no ha sido porque alguien nombró deliberadamente las pruebas para controlar su orden. En cambio, ha sido completamente por accidente. Y no encuentra el problema hasta que ejecuta una prueba aislada que siempre se ha ejecutado como parte de una suite. (O bien, la prueba funciona de forma aislada, por lo que la registra es necesario para otra prueba que no estamos ejecutando en ese momento y que podría fallar).

**¿Cómo hago que mis pruebas se ejecuten aleatoriamente?**

Al momento de aleatorizar las pruebas es primordial, en caso de que fallen, poder reproducir el orden de ejecución exacto para poder depurar la prueba. Para esto, el *framework* de pruebas debe poder entregarte la semilla (*seed*) de aleatorización ejecutada y poder ingresarla de forma tal que sea repetible. A continuación, se muestra como hacerlo en varios *frameworks* populares.

**NodeJS**

Para NodeJS con Mocha, como *framework* de pruebas, debemos instalar el módulo "rocha" que aleatoriza las pruebas:

*npm install rocha –83save-dev*

Para ejecutarlo, podemos crear un script en nuestro archivo package.json así:

*"test": "nyc –reporter=lcov rocha test/*.test.js",*

Para lo cual solo tendremos que ejecutar

*npm run test*

Si la cantidad de pruebas ejecutadas (efectivas o no) es inferior a la del comando original, significa que hay algún archivo mal nombrado, por ejemplo **prueba.js** en vez de **prueba.test.js**. Este error también altera los indicadores del análisis estático de código mediante herramientas como SonarQube.
En caso de que fallen las pruebas, solo tendrás que ejecutar de nuevo las pruebas, pues estas se ejecutarán en el orden exacto que fallo hasta que se depuren y se logre una ejecución positiva. Esto es gracias a que Rocha crea un archivo ".rocha.json" que guarda la semilla (*seed*) de la ejecución; este archivo se eliminará cuando se ejecute una prueba efectiva.

**AngularJS**

Debes modificar el archivo **src/karma.conf.js** así:

```
1.  const karmaJasmineSeedReporter = require('./karma-jasmine-seed-reporter'); // 1. Importing
    custom reporter
2.
3.  module.exports = function(config) {
4.    config.set({
5.      ...
6.      plugins: [
7.        ...
8.        karmaJasmineSeedReporter // 2. Reporter register as karma plugin
9.       ],
10. ...
11. client: {
12.       ...
13.       83version: {
14.         random: true,  // 3. Randomize
15.         stopOnFailure: true,
16.         failFast: true,
17.         // seed: 94349 // 4. Specific seed if needed
18.       },
19.       ...
20. },
```

```
21. …
22. reporters: [“progress”, “kjhtml”, “jasmine-seed”],
23. …
24. )};
```

Luego deberá crear el archivo **src/karma-jasmine-seed-reporter.js** con el siguiente contenido:

```
1.  const karmaJasmineSeedReporter = function(baseReporterDecorator) {
2.    baseReporterDecorator(this);
3.
4.    this.onBrowserComplete = function(browser, result) {
5.      if (result.order && result.order.random && result.order.seed) {
6.        this.write('\n%s: Randomized with seed %s\n', browser.name, result.order.seed);
7.      }
8.    };
9.
10.   this.onRunComplete = function() {
11.      // Empty function
12.   }
13. };
14.
15. module.exports = {
16.   'reporter:jasmine-seed': ['type', karmaJasmineSeedReporter] // 1. 'jasmine-seed' is a name that
    can be referenced in karma.conf.js
17. };
```

Para ejecutar las pruebas solo es necesario que corras el siguiente comando:

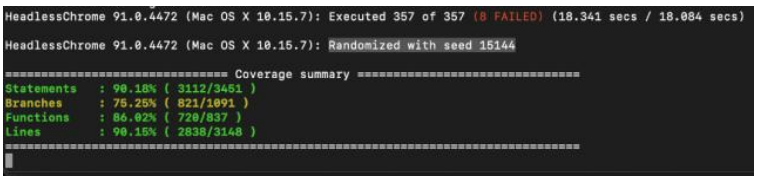
```
ng test
```

O crear en el archivo package.json un *script* así:

```
“test”: “ng test”,
```

Al finalizar la prueba, mostrará la semilla (*seed*), como lo detalla la Figura 25, necesaria para reproducir el orden exacto de ejecución de la prueba. Este valor puedes incluirlo en el archivo src/karma.conf.js

Figura 25. Respuesta de ejecución de pruebas en AngularJS con semilla de aleatorización



Para ver instrucciones mas detalladas, dirígete a Flaky test in Angular — lesson learned

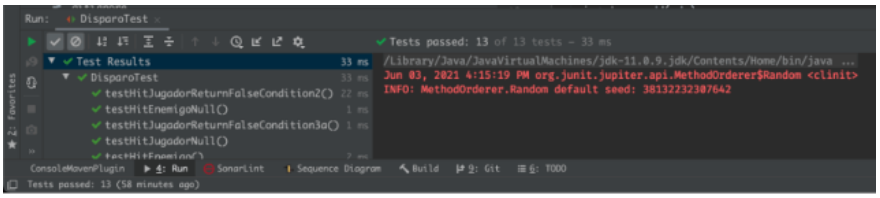
### *Framework* Junit

Para Junit versión mayor o igual a 5.6 (jupiter) es necesario agregar al comienzo de la clase la siguiente anotación:

*@TestMethodOrder(MethodOrderer.Random.**class**)*

Cuando ejecutes las pruebas mediante el IDE, podrás ver que estas aparecen en orden aleatorio. En la consola de ejecución aparecerá la *seed* o semilla de aletarorización, como muestra la Figura 26.

Figura 26. Ejemplo respuesta ejecución de Pruebas Unitarias en IntelliJ con Seed




Cuando ejecutes las pruebas con Gradle o Maven, podrá ver en la consola la *seed* o semilla como lo muestra la Figura 27:

Figura 27. Ejemplo resultado pruebas Junit mediante Gradle con Seed


### Proyectos administrados con Maven

Si no usas Junit o la versión de Junit que tienes es antigua, Maven te provee una forma para ejecutar las pruebas aleatoriamente. Debes modificar el pom.xml del proyecto de la siguiente forma:

```
1.  <plugin>
2.    <groupId>org.apache.maven.plugins</groupId>
3.    <artifactId>maven-surefire-plugin</artifactId>
4.    <85version>3.0.0-M3</85version>
5.    …
```

```
6.      <configuration>
7.        …
8.        <runOrder>random</runOrder>
9.        …
10.     </configuration>
11.     …
12.   </plugin>
```

Para ejecutar la prueba solo será necesario que corras el comando:

*mvn test*

Si quieres comprobar que las *Unit Tests* se están ejecutando aleatoriamente, corre el siguiente comando:

*mvn test -X | grep runOrder*

**ADVERTENCIA**: al momento de escribir este instructivo, Maven y su *plugin* Surefire aún no entregan información de la semilla (*seed*) de ejecución. Esta *feature* se encuentra en desarrollo, en el siguiente link puedes seguir la implementación Adding support for externally passed random seed and printing used seed on console by cardil · Pull Request #112 · apache/maven-surefire

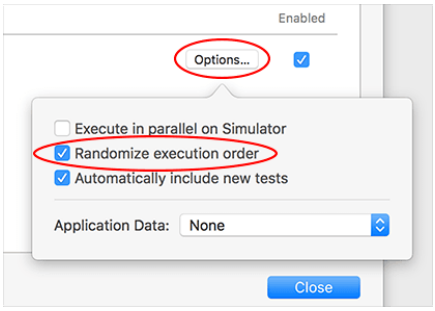
**Xcode**

Xcode 10 agregó una nueva opción importante a XCTest: orden de prueba aleatorio. Puedes habilitarlo editando el esquema del proyecto. Atajo de teclado: en lugar de pasar el mouse al selector de esquema, hacer clic y seleccionar «Editar esquema …», simplemente presiona ⌘ <
En el editor de esquemas, como lo muestra la Figura 28, selecciona “Prueba” en la columna de la izquierda, luego selecciona la pestaña Información.

Figura 28. Configuración de pruebas en Xcode


Para cada paquete de prueba, como muestra la Figura 29, haz clic en el botón “Opciones”. A continuación, selecciona la opción «Orden de ejecución aleatoria».

Figura 29. Configuración de las opciones de alteatorización de pruebas en Xcode




Esto aleatorizará dos niveles de ordenamientos al ejecutar pruebas:

- Suites de prueba
- Casos de prueba dentro de cada suite

¡Hagamos esto para cada paquete de prueba dentro de cada esquema!

Pero Xcode se queda corto. Supón que nos hemos basado en el orden de prueba implícito (determinístico) sin saberlo. Existe la posibilidad de que al ejecutar las pruebas dentro de una suite en orden aleatorio se manifieste un problema oculto. Esté atento a las pruebas que fallan inesperadamente. ¿Pero entonces, qué? Con un diagnóstico cuidadoso, podríamos intentar encontrar una solución. Pero la próxima vez que hagamos las pruebas, **estarán en un orden diferente.** Esto podría enmascarar la falla. Entonces, ¿cómo sabemos si corregimos el problema? ¡Nosotros no! *No, a menos que podamos volver a ejecutar las pruebas en el mismo orden.* RSpec nos da un ejemplo de cómo hacer esto correctamente. Cuando le dice a RSpec que ejecute pruebas en orden aleatorio, imprime la semilla que utilizó para su generador de números aleatorios. Luego, puedes especificar esa misma semilla para obtener pruebas en el mismo orden.
Hasta que tengamos alguna forma de bloquear el orden aleatorio, ¡no sabremos si hemos eliminado los problemas que revela esta función!
La imposibilidad actual de volver a ejecutar las pruebas en el mismo orden evita que esta nueva característica sea tan útil como podría ser.

**Conclusión**

La redacción y comprensión de las pruebas debería ser bastante fácil para los desarrolladores para que sean eficaces. La ejecución de pruebas aleatorias ayuda a los desarrolladores a crear pruebas más confiables y consistentes.

**Referencias**

Este artículo se encontró originalmente en la web ¿Por qué la ejecución aleatoria es una

oportunidad para mejorar las Pruebas Unitarias? - Alexander Andrade

#### 9.1.6.2 Análisis estático de código.

El análisis estático de código es una estrategia que te permite determinar de manera automatizada si el código escrito presenta bugs, vulnerabilidades o incluso complejidad innecesaria que haría mas difícil que otro desarrollador entendiera que se hace en la funcionalidad.
Existen diferentes herramientas de este tipo, pero en ADL se ha escogido SonarQube. Hay dos formas de utilizarlo y lo mejor es hacerlo en conjunto:

1. A través de un Linter que va revisando tu código mientras escribes y te entrega sugerencias de mejora y,
2. A través de una instancia externa (en tu máquina local) de una versión completa de SonarQube, cuya diferencia es que te entrega información de cobertura y en una sola pantalla el resultado del Quality-Gate.

En pocas palabras, debes ejecutar esta segunda opción antes de subir tu PR y aumentar la probabilidad de que este sea integrado (mergeado).

En las siguientes secciones se explicará como configurar estas dos herramientas.

##### 9.1.6.2.1 SonarLint.

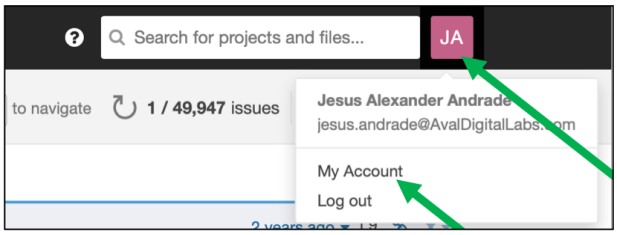
Es una extensión para que el editor de código (IDE) te vaya informando en línea sobre posibles errores de codificación que reducen calidad al código o incluso aumentan la probabilidad de vulnerabilidades o hasta acciones que parecen inofensivas, pero pueden generar comportamiento errático. Tu productividad aumentará, al ir corrigiendo las sugerencias a medida que desarrollo y no tendrás reprocesos de pedir permiso para integrar el código, integrarlo, esperar a que corra el pipeline, verificar el *Quality-Gate* y repetir hasta solucionar.

**Paso previo: generar Token de conexión de Sonar en la nube**

Para poder usar la extensión, tendrás que generar un *token* en http://sonar.avaldigitallabs.com o http://sonarcloud.io , se recomienda configurar solo uno, preferiblemente el que está en dominio de ADL. Los pasos son similares en ambas herramientas.
Ingresa a la herramienta seleccionada (preferiblemente la que está en el dominio de ADL) y

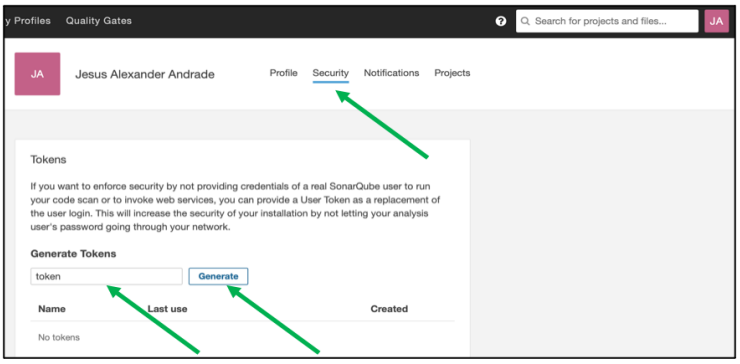
luego a la configuración de tu cuenta, como se muestra en la Figura 30.

Figura 30. Sonar: Configuración de cuenta



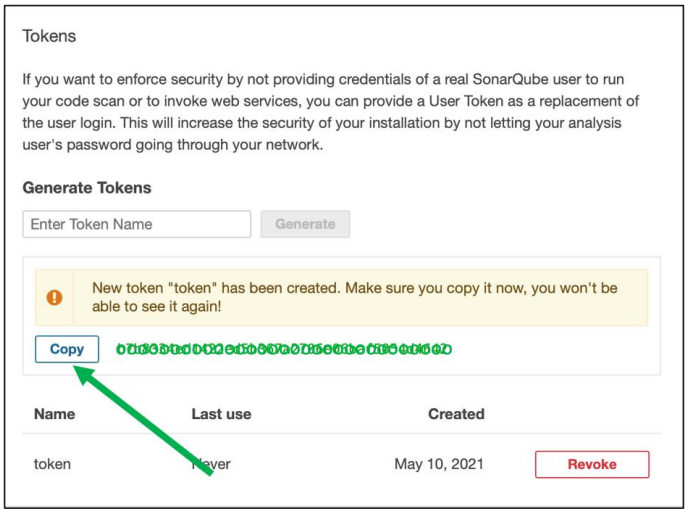

Luego, como muestra la Figura 31, ve al apartado de "*Security*" / "Seguridad" y puedes generar un *token* bajo cualquier nombre, por ejemplo "token" y hace clic sobre el botón "*Generate*" / "Generar".

Figura 31. Sonar: Generar token




Guarda el *token* generado, como muestra la Figura 32, pues tendrás que usarlo mas tarde.

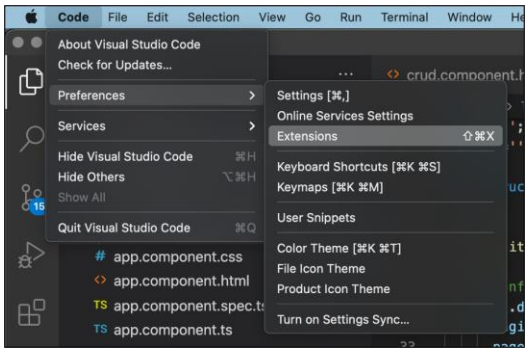
Figura 32. Sonar: Copiar token generado




## Configuración en Visual Studio Code (VS Code)

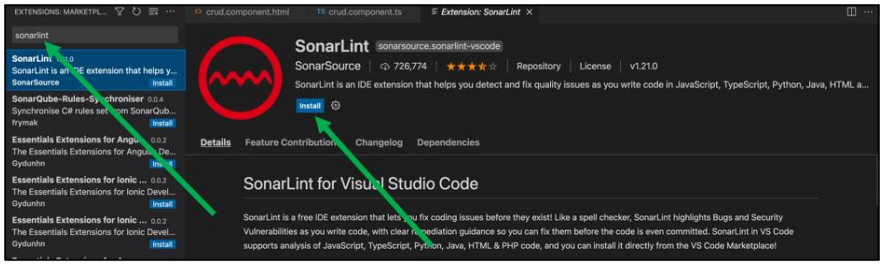
Para configurar tu VSC, debes ir a las preferencias de la aplicación mediante el menú "*Code*" > "*Preferences*" > "*Extensions*", como se detalla la Figura 33.

Figura 33. VSC: Menú de extensiones



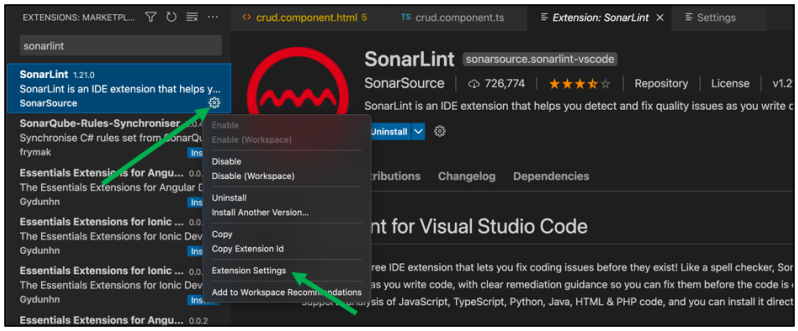

Busca la extensión de SonarLint entre las opciones que aparecen, como muestra la Figura 34.

Figura 34. VSC: Instalación de extensión de SonarLint



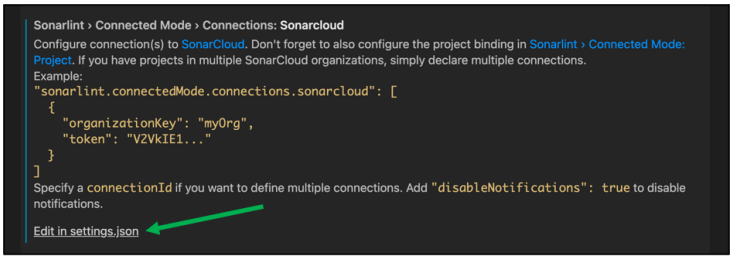

Tras instalar, ve a la configuración de la extensión que se encuentra justo en la rueda dentada (engranaje) junto al nombre de la extensión, como se ve en la Figura 35.

Figura 35. VSC: Opciones de configuración de extensión de SonarLint




En la configuración, como se muestra en la Figura 36, busca la opción llamada “SonarCloud”, que es generalmente la segunda. Abajo nos aparece un enlace que dice “*Edit in settings. json*”.

Figura 36. VSC: Entrar a archivo de configuración general "settings.json"




En el archivo que se abre, debes agregar uno de los siguientes objetos, dependiendo del Sonar donde se sacó el *token* (SonarCloud o SonarQube, respectivamente).

```
1. "sonarlint.connectedMode.connections.sonarcloud": [
2.
3.     {
```

```
4.
5.          "organizationKey": "ExampleOrganization",
6.
7.          "token": "ooooooooooooooooooooooooooooooooooooo"
8.
9.      }
10.
11.       ]
```

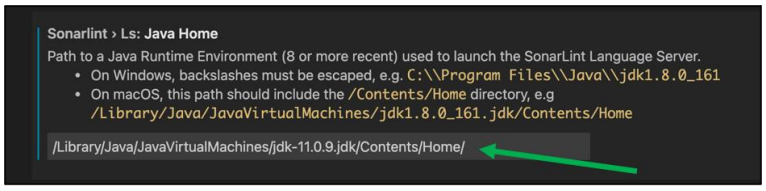
O (para SonarQube, opción menos recomendada)

```
1. "sonarlint.connectedMode.connections.sonarqube": [
2.
3.      {
4.
5.        "serverUrl": "https://sonar.example.com",
6.
7.        "token": "ooooooooooooooooooooooooooooooooooooo"
8.
9.      }
10.
11.       ]
```

Reemplaza los valores de los *tokens* y URL (u "organizationKey" si usas SonarCloud, opción menos recomendada) y luego guarda el archivo y vuelve a las configuraciones de VSC.

Dentro de las configuraciones, que se muestran en la Figura 37, debes buscar las opciones de "Java Home" y "Node Executable". En la siguiente imagen muestra el "Java Home" en el cual debes introducir la ruta del JDK >= 11.

Figura 37. VSC: Configurar ruta JDK

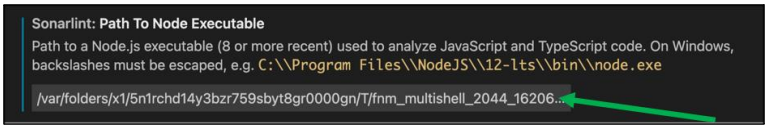



Nota:

- Ruta en Windows de los JDK: C:\Program Files\Java\
- Ruta en Mac de los JDK: /Library/Java/JavaVirtualMachines/[version]/ Contents/Home/

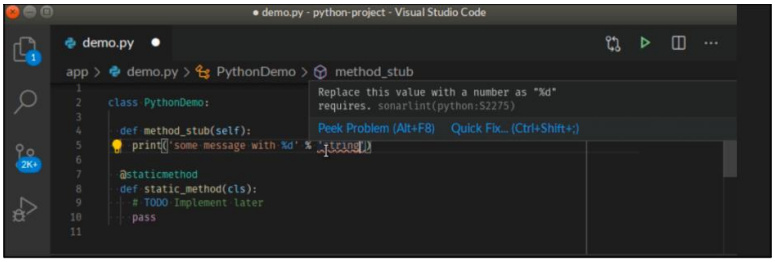
El Node executable se configura en la siguiente opción, como se ve en la Figura 38. Si estás en un Unix (Linux o Mac) puedes ejecutar el comando "type node" para saber la ruta exacta de node.

Figura 38. VSC: Configurar ruta del ejecutable de NodeJS




Al finalizar, debes reiniciar VSC y este descargará las configuraciones de sonar de la organización (empresa). La Figura 39 muestra un ejemplo de como se ven las sugerencias de SonarLint:

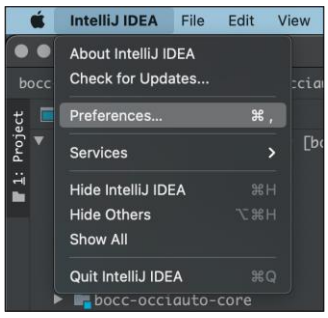
Figura 39. VSC: Ejemplo sugerencia de SonarLInt




## Configuración en IntelliJ

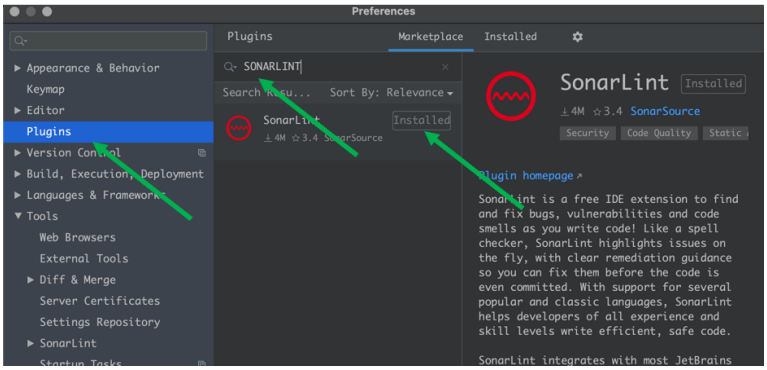
Para configurar SonarLint en IntelliJ, debes ir a las preferencias de la aplicación como lo muestra la Figura 40.

Figura 40. IntelliJ: Menú de preferencias



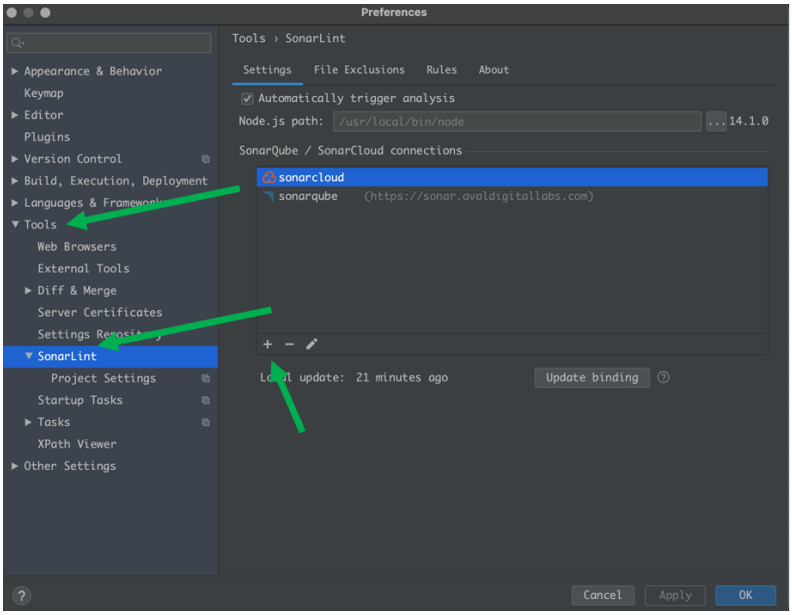

En la ventana que aparece, como se ve en la Figura 41, debes escoger "*Plugins*" del panel derecho y en el buscador introduces la palabra "SONARLINT". En la extensión que aparece tras la búsqueda, realizas un clic en el botón "*Install*" que luego se convertirá en un mensaje que dice "*Installed*".

Figura 41. IntelliJ: Instalación de plugins



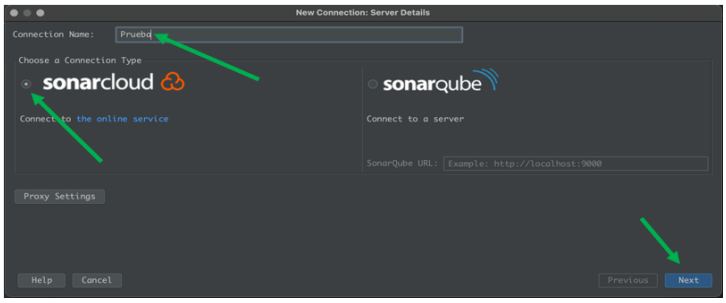

Tras instalar, el sistema te pedirá reiniciar IntelliJ. Al volver, debes ir nuevamente a las preferencias de la aplicación (ver primer punto) y ahora escogerás en el panel derecho la opción "*Tools*" > "SonarLint", como se ve en la Figura 42.

Figura 42. IntelliJ: Configuración de SonarLint




Estando allí, como lo muestra la Figura 43, debes escoger el signo más (+) para agregar una nueva conexión de SonarQube / SonarCloud. En la ventana que aparece, debes poner un nombre para nuestra conexión (ej: Prueba), luego escoges entre "sonarcloud" o "sonarqube", siendo el mas recomendable SonarQube por estar en el dominio de ADL. Posteriormente haces clic sobre el botón "*Next*".

Figura 43. IntelliJ: Configuración de token en SonarLint




En la siguiente ventana se te pedirá ingresar el *token*, el cual generaste al inicio de este instructivo. Podemos repetir el proceso por cuantas conexiones de SonarQube o SonarCloud tengamos, pero lo mas recomendable es dejar solo una.

**Extra**

IntelliJ al igual que VSC va revisando el código mientras se digita o se abren los archivos, pero en el caso de IntelliJ también ofrece opciones para revisar el código de todo el proyecto a discreción. Para esto, debes ir al menú "*Analyze*" de donde podremos escoger dos opciones:

1. *Analyze* VSC: analizar solo el código que hemos cambiado con respecto al repositorio.
2. *Analyze all files*: analizar el código de todo el proyecto.

#### 9.1.6.2.2 Instalar SonarQube en una instancia local

A diferencia del SonarLint, este no irá revisando el código mientras se escribe, pero es mucho mas completo. No deberías subir un PR sin antes haber ejecutado esta acción en tu máquina local, con lo que te ahorras mucho tiempo y a tu equipo.
Este tiene dos formas de ejecutarse, la primera (y recomendada, pues tiene un impacto menor en los recursos de tu computador) es descargar el JAR y ejecutarlo, y la segunda es a través de un docker con el correspondiente sacrificio de recursos del sistema. Para ambos casos utilizaremos la edición "*Community Edition*" en su versión LTS (*Long Term Support*: Suporte de largo término).

**Instalar SonarQube**

**La versión JAR de SonarQube**

En esta web puedes descargar la versión "*Community Edition*" del JAR Download | SonarQube, prefiriendo siempre la versión LTS. Una vez descargado, descomprime el archivo ZIP y ubica su contenido donde puedas tenerlo a mano, por ejemplo, en Documentos.
Para ejecutarlo, solo tendrás que acceder con una Terminal (si estás en Mac o Unix) o el "Símbolo de Sistema" (MSDos, si estás en Windows) hasta la ruta donde tienes guardado el ejecutable JAR del sonar y ejecutar el siguiente comando:

- En Windows: *start sonarqube/bin/macosx-universal-64/sonar.bat start*
- En Mac / Unix: *sh sonarqube/bin/macosx-universal-64/sonar.sh start*

**La versión Docker de SonarQube**

Si prefieres ejecutar el SonarQube mediante un docker (se recomienda la versión JAR por su bajo consumo de recursos con respecto a esta solución), podrás ejecutar desde tu Terminal o "Símbolo de Sistema" el siguiente comando
*docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube:lts*

**Configurar el proyecto para analizarse con SonarQube**

**Generar los tokens del SonarQube local**

Para esto, debes ingresar a la web local de Sonar http://localhost:9000, el usuario y contraseña por defecto es **admin**. Si el sistema te pide cambiar la contraseña, es recomendable dejarla como "admin123".

Para generar el *token*, sigue las instrucciones que se encuentran en este mismo instructivo: Análisis estático de código | Paso-previo:-generar-Token-de-conexión-de-Sonar-en-la-nube.

Como acto final, deberás crear una variable de entorno llamada "SONAR_TOKEN" con el contenido del *token*, por ejemplo:

*SONAR_TOKEN=ooooooooooooooooooooooooooooooo*

**Nota**: para que los programas lo puedan usar, lo mejor es reiniciar dicho programa o Terminal/Símbolo del Sistema.

**Analizar localmente tus proyectos**

**Configurar proyectos NodeJS**

Deberás instalar las librerías npm con el siguiente comando:

*npm install sonar-scanner sonarqube-scanner --save-dev*

En la misma ruta del package.json (la principal del proyecto) deberás crear los siguientes archivos con estos contenidos:

**/sonar-project.properties**

```
1. sonar.sourceEncoding=UTF-8
2. sonar.sources=src
3. sonar.exclusions=**/node_modules/**,**/*.spec.js
4. sonar.tests=test
5. sonar.test.inclusions=**/*.test.js
6. sonar.javascript.lcov.reportPaths=coverage/lcov.info
7. sonar.projectKey=empresa-celula-nombrelargodelproyecto:nombrecorto
8. sonar.projectName=empresa-celula-nombrelargodelproyecto:nombrecorto
9. sonar.scm.provider=git
```

Tener cuidado con las siguientes líneas, que deben reflejar la realidad de tu proyecto:

- Línea 4: debe reflejar el nombre de la carpeta de las pruebas unitarias.
- Línea 5: debe reflejar el patrón del nombre de los archivos de pruebas en su carpeta de pruebas unitarias.
- Línea 6: debe reflejar el archivo donde queda la información de cobertura.
- Líneas 7 y 8: reflejan la *key* y nombre del proyecto con el que quedará en el reporte de Sonar del *pipeline* (en la nube).

**/sonar-project.js**

```
1. const sonarqubeScanner = require("sonarqube-scanner");
2. sonarqubeScanner(
3.   {
4.     serverUrl: "http://localhost:9000",
5.     options: {
6.       "sonar.sources": "src",
7.       "sonar.inclusions": "src/**/*.js"
8.     }
9.   },
10.         () => {}
11.       );
```

El archivo **/package.json** deberá tener líneas similares a las siguientes para que todo funcione:

```
1. "test": "nyc --reporter=lcov rocha test/*.test.js
   test/**/*.test.js",
2. "sonar-local": "npm test && node sonar-project.js",
3. "cover": "npm run test && nyc report --reporter=html && open
   coverage/index.html",
4. "sonar": "sonar-scanner -Dsonar.organization=$SONAR_ORGANIZATION -
   Dsonar.host.url=$SONAR_URL -Dsonar.login=$SONAR_TOKEN"
```

Finalmente, podrá ejecutar el sonar local con los siguientes comandos en tu Terminal o Símbolo del Sistema. Primero, debes tener corriendo una instancia de SonarQube en tu máquina local:
*npm run cover*

*npm run sonar-local*

**Configurar proyectos Gradle**

En el archivo build.gradle deberás agregar las siguientes configuraciones en los puntos que correspondan. No debes copiar los tres puntos horizontales (…) del ejemplo de abajo, se pusieron para indicar que donde están estos hay otras partes del formato de dicho archivo:

```
1. plugins {
2.     id "java-library"
3.     id "org.sonarqube" version "3.0"
4. }
5. ...
6. apply plugin: 'org.sonarqube'
7. ...
8. sonarqube {
9.     properties {
10.             property "sonar.projectKey", "empresa-celula-
   nombrelargodelproyecto:nombrecorto"
11.             property "sonar.projectName", "empresa-celula-
   nombrelargodelproyecto:nombrecorto"
12.             property "sonar.sources", "src"
13.             property "sonar.sourceEncoding", "UTF-8"
14.             property "sonar.java.binaries", "build/classes"
15.             property "sonar.coverage.exclusions", "**/*.java"
16.             property "sonar.exclusions", "**/*Generated.java"
17.             property "sonar.tests", ""
18.             property "sonar.scm.provider", "git"
19.         }
20.     }
21.     ...
```

Para realizar el análisis deberás tener corriendo una instancia local de SonarQube y ejecutar

el siguiente comando:

*./gradlew sonarqube*

**Configurar proyectos Maven**

En el archivo pom.xml deberás tener las siguientes configuraciones en los puntos que correspondan. No debes copiar los tres puntos horizontales (…) del ejemplo de abajo, se pusieron para indicar que donde están estos hay otras partes del formato de dicho archivo:

```
<properties>
    ...
       <sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
       <sonar.coverage.exclusions>
        **/xx/xxx/xxxxx/common/audit/enums/EventType.java
       </sonar.coverage.exclusions>
       <sonar.cpd.exclusions>
          **/xx/xxx/xxxx/auth/model/**
       </sonar.cpd.exclusions>
       ...
      </properties>
      ...
      <build>
        <pluginManagement>
          <plugins>
          ...
          <!-- Sonarqube & Test -->
            <plugin>
               <groupId>org.jacoco</groupId>
               <artifactId>jacoco-maven-plugin</artifactId>
               <version>0.8.6</version>
               <configuration>
                  <destfile>${basedir}/target/jacoco.exec</destfile
  >
               </configuration>
               <executions>
                   <execution>
                       <id>jacoco-initialize</id>
                       <phase>initialize</phase>
                       <goals>
                          <goal>prepare-agent</goal>
                       </goals>
                   </execution>
                   <execution>
                       <id>jacoco-report</id>
                       <phase>prepare-package</phase>
                       <goals>
                         <goal>report</goal>
                       </goals>
                   </execution>
               </executions>
```

```
            </plugin>
            <!-- Sonarqube -->
            <plugin>
                <groupId>org.sonarsource.scanner.maven</groupId>
                <artifactId>sonar-maven-plugin</artifactId>
                <version>3.8.0.2131</version>
            </plugin>
            ...
          </plugins>
       </pluginManagement>
      ...
     </build>
     ...
```

Para realizar el análisis deberás tener corriendo una instancia local de SonarQube y ejecutar el siguiente comando:

*mvn sonar:sonar*

#### 9.1.6.3 Pruebas de desempeño.

| Contribuidores al momento de realizar el trabajo: Oscar Leonardo Bustos, Alexander Andrade |
|---|

Para verificar el desempeño de los componentes comunes se recomienda utilizar la herramienta Artillery https://artillery.io/.

Prerrequisitos: Tener instalado node y npm

Instalación:

npm install -g artillery

**Configuración**:

Se requiere configurar un YML con la prueba. A continuación, se muestran las pruebas para la lambda de engine-rules. En esta prueba se apagó temporalmente el WAF y el Cognito. Se creó el siguiente archivo llamado **rule_lambda.yml**

```
config:
  target: "https://XXX.execute-api.ZONE.amazonaws.com"
  phases:
    - duration: 60
      arrivalRate: 20
scenarios:
  - flow:
```

```
    - log: "Nuevo usuario"
    - post:
        url: "/dev/api/commons/engine-rules"
        json:
          rules: "https://XXX.ZONE.amazonaws.com/rules-test/example-rules.json?"
            a: 1
            c: 4
```

Con el siguiente comando se ejecutaron las pruebas de carga:

*artillery run rule_lambda.yml --output result.json*

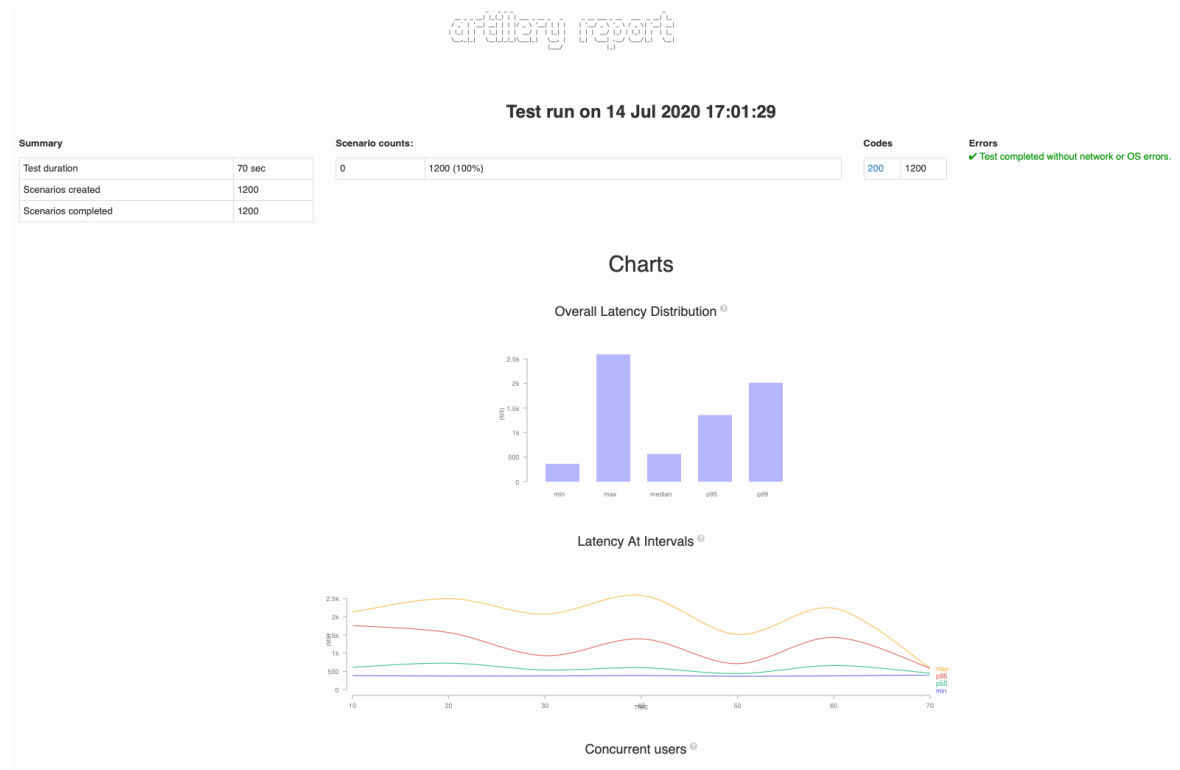
**Reporte**:

Con el siguiente comando puedes generar el reporte gráfico.

*artillery report result.json*

La Figura 44 muestra cómo se vería el resultado de ejecución:

Figura 44. Ejemplo archivo HTML de ejecución de pruebas con Artillery



### 9.1.6.4 Pruebas de regresión.

| Contribuidores al momento de realizar el trabajo: Andrés Felipe Burbano |
|---|

Actualmente podría hacer uso **solo** para la creación de la estructura del proyecto y el *pipeline*.

El *stack* de pruebas de regresión para los servicios de la Nube Azul está basado en NodeJS usando el motor CLI Newman de Postman. Las figuras Figura 45 y Figura 46 detallan, respectivamente, los flujos de los procesos de Calidad de Software (SQA) y de desarrollo para el esquema planteado.

Figura 45. Diagrama de flujo del Proceso de Calidad (QA)

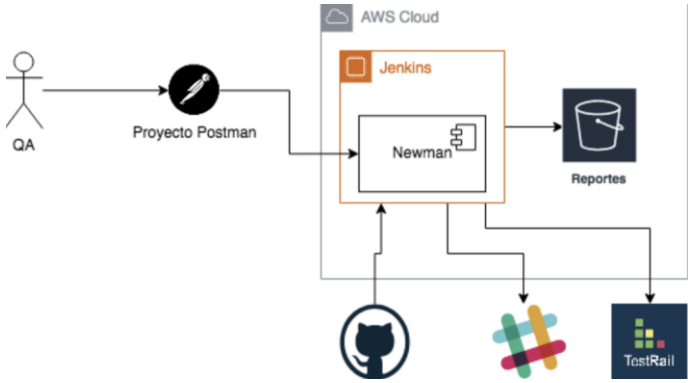
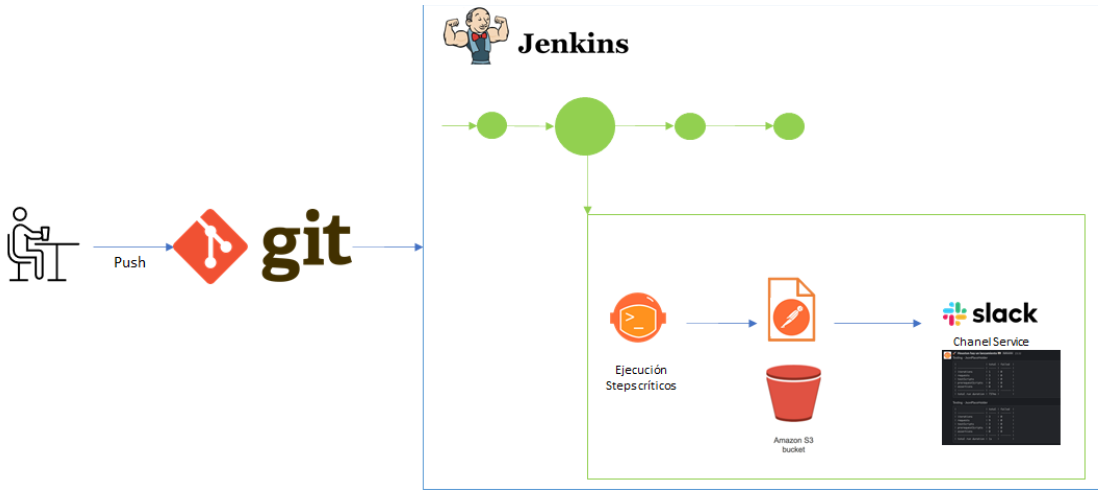



Figura 46. Diagrama de flujo del proceso del desarrollador




**Acuerdos**

- Se debe acordar el nombramiento de la colección y los ambientes del proyecto Postman por célula.
- Se propone tener un canal por componente común en donde se pueden incluir los desarrolladores - QAs de las células que estén consumiendo el servicio común.

**Datos de prueba**

Se debe validar y acordar los datos de prueba que van a usarse en la ejecución.

**Integraciones**

La solución integra diferentes componentes:

- Slack: Notifica el estado de la ejecución del pipeline
- AWS S3: Almacena los reportes de salida con el detalle de la ejecución
- Jenkins: Ejecución del pipeline
- GitHub: Proyecto creado en NodeJS
- TestRail: Integración para marcar la ejecución automáticamente los casos de pruebas (próxima integración)

Repositorio de GitHub: XXX
Pipeline: XXX

### 9.1.7 Recomendaciones para GitHub.

| |
|---|
| Contribuidores al momento de realizar el trabajo: Alexander Andrade. NOTA: Esta sección se redactó en tercera persona y no en forma de tuteo como el resto del gobierno, pues pretende ser muy formal. |

Estas recomendaciones, accesorias, sobre el uso de GitHub te ayudarán a tener un mejor código, que este sea revisado más rápidamente y que tus cambios se aprueben de forma muy fácil.

#### 9.1.7.1 Código de conducta del contribuidor de código

La siguiente información es una traducción literal del código de conducta de contribuidores

de ADL vigente para la fecha 09-nov-2021 y es una adaptación del Pacto de Colaboradores, versión 1.4, disponible en Contributor Covenant:. Este tendrá validez mientras ADL lo actualiza o se basa en una nueva versión del *Contributor Covenant* o el Laboratorio establece uno nuevo.

**Nuestro compromiso**

Con el interés de fomentar un entorno abierto y acogedor, nosotros, como contribuyentes y mantenedores, nos comprometemos a hacer de la participación en nuestro proyecto y nuestra comunidad una experiencia libre de acoso para todos, independientemente de su edad, tamaño corporal, discapacidad, etnia, identidad y expresión de género. nivel de experiencia, educación, estatus socioeconómico, nacionalidad, apariencia personal, raza, religión o identidad y orientación sexual.

**Nuestras Normas**

Ejemplos de comportamiento que contribuye a crear un ambiente positivo incluyen:

- Usar un lenguaje acogedor e inclusivo
- Ser respetuoso con los diferentes puntos de vista y experiencias.
- Aceptando con gracia las críticas constructivas
- Centrarse en lo que es mejor para la comunidad
- Mostrar empatía hacia otros miembros de la comunidad

Ejemplos de comportamiento inaceptable por parte de los participantes incluyen:

- El uso de lenguaje o imágenes sexualizadas y atención o insinuaciones sexuales no deseadas.
- Troleo, comentarios insultantes / despectivos y ataques personales o políticos
- Acoso público o privado
- Publicar información privada de otros, como una dirección física o electrónica, sin permiso explícito
- Otra conducta que razonablemente podría considerarse inapropiada en un entorno profesional.

**Nuestras responsabilidades**

Los encargados del mantenimiento del proyecto son responsables de aclarar los estándares de comportamiento aceptable y se espera que tomen las medidas correctivas adecuadas y justas en respuesta a cualquier caso de comportamiento inaceptable.
Los responsables del proyecto tienen el derecho y la responsabilidad de eliminar, editar o rechazar comentarios, confirmaciones, códigos, ediciones de wiki, problemas y otras

contribuciones que no estén alineadas con este Código de conducta de ADL, o prohibir temporal o permanentemente a cualquier colaborador por otros comportamientos. que consideren inapropiadas, amenazantes, ofensivas o dañinas.

**Alcance**

Este Código de Conducta se aplica tanto dentro de los espacios del proyecto como en los espacios públicos cuando una persona representa el proyecto o su comunidad. Ejemplos de representación de un proyecto o comunidad incluyen el uso de una dirección de correo electrónico oficial del proyecto, la publicación a través de una cuenta oficial de redes sociales o la actuación como representante designado en un evento en línea o fuera de línea. Los encargados del mantenimiento del proyecto pueden definir y aclarar más la representación de un proyecto.

**Aplicación**

Los casos de comportamiento abusivo, acosador o inaceptable de cualquier otro modo se pueden informar comunicándose con el equipo del proyecto en devops@avaldigitallabs.com. Todas las quejas se revisarán e investigarán, y resultarán en una respuesta que se considere necesaria y apropiada a las circunstancias. El equipo del proyecto está obligado a mantener la confidencialidad con respecto al informante de un incidente. Se pueden publicar más detalles de las políticas de aplicación específicas por separado.
Los mantenedores del proyecto que no sigan o hagan cumplir el Código de conducta de buena fe pueden enfrentar repercusiones temporales o permanentes según lo determinen otros miembros del liderazgo del proyecto.

#### 9.1.7.2 Commits atómicos

Para un revisor de código, es muy cómodo encontrase con *commits* atómicos, pues le permite entender la historia o evolución del PR que está revisando. Seguir esta guía te permitirá tener facilitarle el trabajo al revisor de código y acelerar la integración de este.

Este artículo se tomó de la web Haciendo commits atómicos y siempre primarán las normativas que se establezcan a nivel de la organización o de ADL.

**Commits atómicos**

La palabra átomo significa la partícula más pequeña posible de materia, indivisible.
Si al utilizar Git sigues la estrategia de crear una rama nueva por cada *feature* a crear o por cada bug a resolver, un commit atómico se correspondería con cada una de las subtareas en las que podrías desglosar la tarea principal.
Estas subtareas es posible que signifiquen cambiar o crear un fichero o varios, lo importante es darle al *commit* un valor semántico.

**¿Porque *commits* atómicos?**

Los *commits* atómicos tienen principalmente tres ventajas:

- *Code reviews* más fáciles, independientemente de que el *code review* se haga a través de un *pull request* o con el compañero sentado al lado, es mucho más fácil de entender el historial de *commits* si están creados desde un punto de vista semántico y atómico que si están creados desde un punto de vista subjetivo a solución aplicada técnicamente.
- Más fácil de revertir, imagina que después de solucionar un *bug*, no lo hemos solucionado correctamente o incluso hemos introducido otro, en este caso una posible solución es revertir la solución que habíamos aplicado. Si no existe un *commit* que indique claramente el punto donde se introdujo esa solución va a ser complicado de revertir y no quedará más remedio que bucear por el código.
- Moverte por el historial de *commits* es más fácil, hay diferentes motivos por lo que puede ser interesante moverte a un punto del historial de *commits* en concreto. En este caso no hay nada peor que hacer *check out* de un *commit* y que el proyecto no compile o le falte algo a cierta funcionalidad para estar completa y que esté en otros *commits* mezclado con otras funcionalidades.
- **Sentido común**

Es cierto que hay parte subjetiva en todo esto y es posible que haya diferentes criterios entre varias personas, lo importante es aplicar el sentido común y tener una estrategia base consensuada entre los componentes del equipo aceptando estas pequeñas diferencias de criterio.
Sin embargo, hay situaciones que son difíciles de poder defender, por ejemplo:

- Un *pull request* con 1 solo commit y muchos ficheros medicados.
- Un *pull request* con 1 commit por cada fichero modificado.

Aunque puedan parecer casos muy exagerados, son casos que me he encontrado revisando *pull request* más veces de las que me gustaría y no es fácil de lidiar.

**El hábito hace al monje**

Si durante tu día a día vas realizando *commits* para cada una de esas subtareas que vas completando es más fácil conseguir tener un historial de *commits* atómicos.
Todo lo contrario, sucede si caes en la práctica de hacer commit al final del día o según te apetece.

**Ejemplo**

Si un diseñador te envía una serie de cambios considerables a realizar en varias pantallas, una posible estrategia sería crear una rama por cada una de las pantallas y un *commit* por cada cambio que nos piden dentro de cada pantalla.

### 9.1.8 Uso del tablero JIRA de los servicios comunes

Contribuidores al momento de realizar el trabajo: Alexander Andrade

Todos los proyectos de los Servicios Comunes comparten un único tablero en el cual se podrán listar los trabajos pendientes por realizar, así como ver el avance de estos y filtrarlos por cada Servicio Común. A continuación, se explican unas sencillas reglas para poder tener el mejor seguimiento con el mismo.

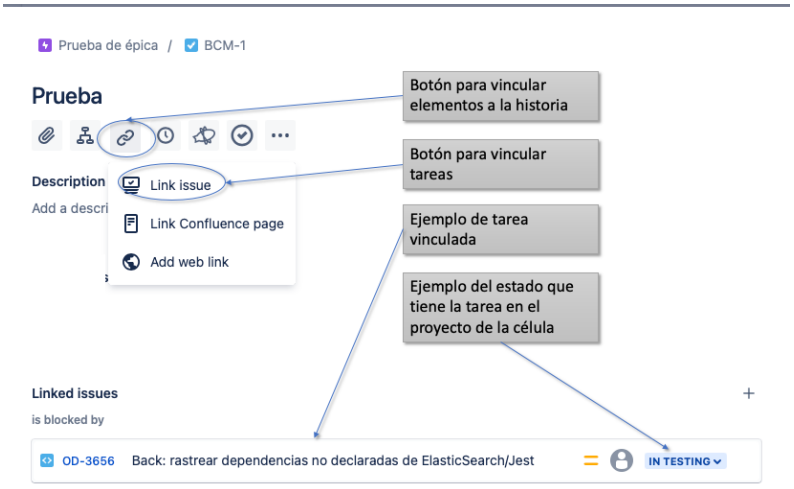
#### 9.1.8.1 Reportar el avance de tareas sin aumentar el esfuerzo

En este tablero solo se mapearán PBI (*product backlog items*) de tipo macro: grandes y sin detalles. A continuación, podrás ver como usar el tablero cuando se haces una contribución desde tu célula o cuando se trata de un contribuidor individual.

##### 9.1.8.1.1 Células contribuidoras

A continuación, verás como publicar tus avances sin que esto tenga impacto en tus labores. En pocas palabras, se trata de que en las macro-tareas contenidas en este tablero se relacionen las tareas del tablero de tu célula, así cuando actualices allá (en el tablero de la célula) los contribuidores o usuarios de los Servicios Comunes tendrán avance de como van las cosas.

Figura 47. Instrucciones para vincular tareas desde otros tableros

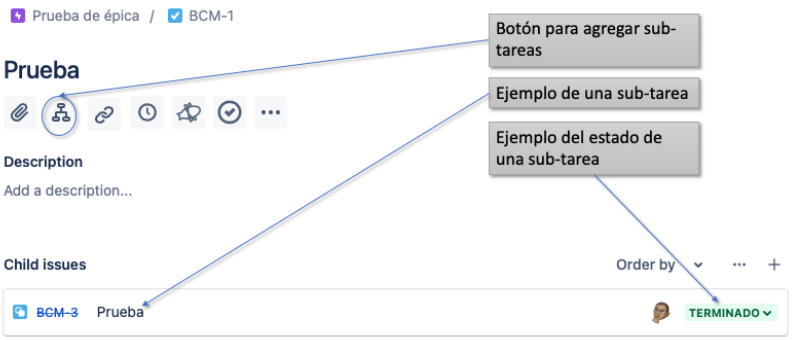



Un truco puede ser que en vez de vincular tareas individuales crees una épica para el trabajo que estás haciendo para el Servicio Común, y esta épica será la que se coloque como tarea vinculada.

#### 9.1.8.1.2 Contribuidores individuales

Si como contribuidor individual no puedes crear tareas en el tablero de tu célula, pues es un "trabajo independiente", puedes reportar el avance de las actividades mediante la creación de sub-tareas (o *Child-tasks o Child-issues*).

Figura 48. Botones para agregar sub-tareas




### 9.1.8.2 Estados del tablero

El tablero solo tiene cinco estados y debes recordar que no tendrás que poner el detalle de

las historias en este tablero como se indica en el punto anterior. Los estados son los siguientes:

- Por revisar: Aquí llegan todas las ideas a espera que alguien (un mentor, un voluntario o cualquier persona) lo valide considerando que aporta valor a toda la comunidad del Laboratorio del Banco de Occidente.
- Aceptado (*To Do*): Aquí estarán las historias que están aceptadas y esperan que un voluntario quiera tomarla para desarrollarla.
- En desarrollo: Aquellas historias que están en proceso de desarrollo. Se recomienda leer el título anterior sobre "Reporta el avance de tareas sin aumentar el esfuerzo".
- ✓Terminado: Las historias finalizadas. En caso de que la historia se haya tratado de
    - Investigación o Documentación: un muy breve resumen con conclusiones. Por ejemplo: *"La investigación se encuentra plasmada en la documentación bajo el título 'Cómo crear clientes'"*.
    - Desarrollo de software: se entenderá como finalizado cuando se encuentre en desplegado en Producción y traer el enlace al *Release Plan* con el cual se desplegó.
- ✗Inviable: Historias que se consideran no realizables y deberá tener una documentación del motivo por el cual se descarta de acuerdo a lo mencionado en Gobierno | ¿Que-hago-si-un-aporte-que-realicé-fue-rechazado?

#### 9.1.8.3 Detalle de uso de cada tipo de historia

1. Utilizar las **Épicas** (*Epic*) para agrupar los proyectos. Por ejemplo, todas las actividades relacionadas con el servicio de "Listas restrictivas" deben agruparse en una épica llamada así; de esta forma es fácil filtrar estos trabajos y aportar con varias soluciones o hacer un seguimiento a la evolución del servicio. Nota: recuerda que la definición en Jira para épica es "contenedor de historias, tareas y otros", lo cual difiere un poco del término de "épica" en *Scrum*.
2. Las **Historias de Usuario** (*Story*) se utilizarán para contener dentro de si el desarrollo inicial de un servicio, y luego las modificaciones en cuanto al incremento de valor de cara al usuario. Este valor puede ser tangible (la adición de un dato de respuesta) o intangible (una nueva fórmula para calcular). Sobre todo, son incrementos de valor completo que tienen el potencial de salir a producción, que no poseen dependencias de trabajos que están por realizarse, y recibir retroalimentación de los usuarios mediante su uso. Estas historias estarán escritas en formato Gherkin y se debe evitar hacer referencias a la tecnología (botones, pantallas, etc.) y acciones de interacción con esta como clics, envíos, etc.; debe estar escrito en términos de proceso de negocio y no como un manual de usuario de la herramienta.
3. Las **Historias Técnicas** (*Tech Story*) se usarán para mapear los trabajos de desarrollo de software que no son incrementos de valor en términos de proceso de negocio (ver

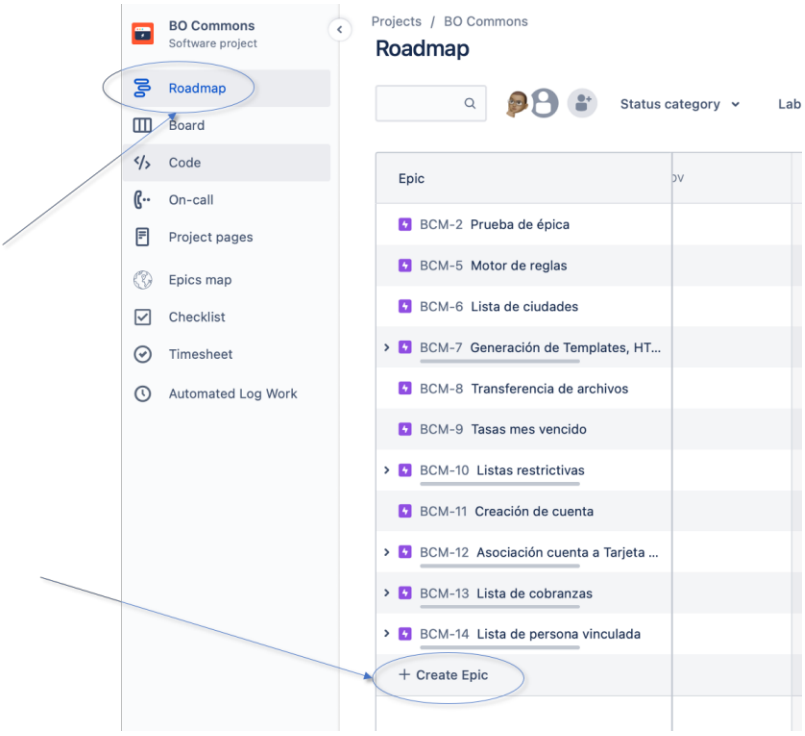
descripción anterior). Por ejemplo: cambiar un botón por un *radio-button*, agregar un elemento de seguridad. Otro caso habitual es el desarrollo parcial de una solución que aún requiere de otra parte (también conocido como un habilitador); en este último caso se puede hablar de cambios de infraestructura y otros.

4. Los **Bugs** serán utilizados para reportar los elementos que no funcionan cómo se han definido por hallazgos atribuibles al proceso de desarrollo. No serán bugs situaciones como el fallo de algo porque falta información en una base de datos, o porque faltan unos permisos en AWS; mientras si lo será la falta de despliegue de un parámetro o de un componente que solo llegó hasta un ambiente anterior. Para declararse *bug* también debe cumplir con la condición de que se puedan entregar los pasos exactos para reproducirse en ambientes bajos.
5. Los ***Story Bugs*** hacen referencia a hallazgos relacionados con una historia en curso, o sea una que no se ha finalizado.
6. Las **Deudas Técnicas** serán elementos similares a los bugs pero que no son atribuibles al proceso de desarrollo. En este caso son cambios de tecnología o elementos que se dejaron para futuras entregas como un acuerdo de equipo (técnico).
7. Los ***Spikes*** son tareas de averiguación, estudio o hasta de investigación.
8. Las **Tareas** (*Tasks*) son tareas de cualquier tipo que no encajan en las anteriores, o sea, pueden ser cosas rutinarias o conocidas, con poca incertidumbre como la anteriores. Aquí caben cosas como mejoras en la documentación entre otros.

#### 9.1.8.3.1 Crear épicas (proyecto de servicio común)

En los apartados anteriores se explicaba que los Servicios Comunes deben estar contenidos entre épicas con el nombre del proyecto. Para crear una nueva épica, debes ir en el menú de la izquierda a "*Roadmap*" y luego a "*Create Epic*".

Figura 49. Instrucciones para crear una Épica en el tablero




#### 9.1.8.3.2 ¿Cómo indicar que una tarea está bloqueada?

Es posible que una historia que ya se encuentra en un estado diferente a “Por revisar” deba ser bloqueada por alguna razón. En este caso se debes hacer clic-derecho a la historia y ponerle una bandera (*Flag*) indicando el motivo. Esto servirá para que otro voluntario entienda que no se puede gestionar hasta resolver la situación reportada.

## 9.2 ANEXO B: PLANTILLAS

### 9.2.1 Listado de servicio comunes.

Contribuidores al momento de realizar el trabajo: Oscar Leonardo Bustos, Juan Carlos Luna, Cesar Mejía, Christian Braatz, Alexander Andrade

Este listado se compone de una tabla (ver Tabla 11) que resume todos los servicios, tanto disponibles como en planes de creación. Para el Trabajo de Grado se evita colocar información real, dada la sensibilidad de la información.

Tabla 11. Plantilla inventario (resumen) general de servicios comunes

| Servicio | Estado | Tipo | Nivel de Servicio Objetivo | Nivel de Servicio Actual | Observación |
|---|---|---|---|---|---|
| Nombre del servicio y link a su documentación | Migrado, En proceso, Candidato | Utilitarios, Proxy o Librería. Ver Tabla 8. | Número que indica el nivel de reutilización objetivo o actual. Ver Tabla 7. | | Comentarios generales |

#### 9.2.1.1 Listado de servicios utilitarios

Contribuidores al momento de realizar el trabajo: Oscar Méndez, Oscar Leonardo Bustos, Christian Braatz, Alexander Andrade

Servicios útiles para varias células pero que no tienen conexión con un tercero. Estos deben cumplir con unos requerimientos no-funcionales comunes y unas recomendaciones generales. La Tabla 12 muestra la plantilla de inventario de este tipo de servicios.

Tabla 12. Plantilla inventario (resumen) de servicios comunes de tipo utilitario

| Servicio | Tecnología | Estado | Descripción |
|---|---|---|---|
| Nombre del servicio y link a su documentación | Lenguaje de programación y otros | Migrado, En proceso | Comentarios generales |

#### 9.2.1.2 Listado de servicios proxy

| |
|---|
| Contribuidores al momento de realizar el trabajo: Raúl Rojas, Oscar Leonardo Bustos, Christian Braatz, Alexander Andrade |

Servicios útiles para varias células pero que tienen conexión con un tercero y aún cumple con el Nivel 2 de Reutilización. Estos deben cumplir con unos requerimientos no-funcionales comunes y unas recomendaciones generales. La Tabla 13 muestra la plantilla de inventario de este tipo de servicios.

Tabla 13. Plantilla inventario (resumen) de servicios comunes de tipo proxy

| Servicio | Servicio destino | Proveedor | Tecnología | Estado | Descripción |
|---|---|---|---|---|---|
| Nombre del servicio y link a su documentación | Servicio destino | Nombre del proveedor | Lenguaje de programación y otros | Migrado, En proceso | Comentarios generales |

#### 9.2.1.3 Listado de librerías

| |
|---|
| Contribuidores al momento de realizar el trabajo: Raúl Rojas, Alexander Andrade |

Las librerías son proyectos con métodos o funciones puntuales en el cual puedes anexar a otros proyectos y complementarlo usando sus métodos específicos para una determinada solución. No son servicios, no exponen una API.

### 9.2.2 Plantilla de documentación

Para documentar los diferentes elementos, se tienen dos plantillas: 1) para servicios (utilitarios o proxy), que se relaciona en la Tabla 14, y otra para librerías, que se relaciona en la Tabla 15. Para llegar al diseño final, se tomaron ideas de documentaciones anteriores realizadas por Raúl Rojas, Oscar Bustos, Christian Braatz, Juan Fernando Castro, Oscar Méndez, Felipe Ordoñez y Alexander Andrade.

Tabla 14. Plantilla de documentación de servicios utilitarios y proxy

| Elemento | Descripción |
|---|---|
| Introducción | Breve descripción con estado del paso a producción y lenguaje de desarrollo. |
| Arquitectura | • Diagrama de contexto de arquitectura<br>• Recursos del proyecto: url de repositorio de código, Resultado de *Quality-Gate* y enlace al informe, y enlace a documentación de casos de prueba.<br>• *Endpoints*: del servicio en los tres ambientes. |
| Arquitectura extra: aplica solo para servicios proxy. | • Información del servicio de destino: contiene la entidad, unidad o área responsable de dar soporte e información sobre procesos de seguridad como *tokens* o certificados.<br>• *Mock* del servicio externo: información sobre como conectar el servicio al *mock* del servicio externo y copia del archivo del *mock* para cargar en la herramienta. |
| Documentación de la API / Swagger | Objeto incrustado de la especificación Open API / Swagger. |
| Log de auditoría | Este contiene información sobre como consultar el log de observabilidad del servicio. |
| Gestión de calidad | Certificación sobre el primer paso a producción del servicio |
| Células que utilizan el servicio | Listado sobre las células que utilizan el servicio y fecha de implementación. Estas células tienen un link que lleva a información de contacto de estas. |

Tabla 15. Plantilla de documentación de librerías

| Elemento | Descripción |
|---|---|
| Introducción | Breve descripción de la librería, con enlace a repositorio GitHub y enlace a reporte de análisis estático de código. |
| Miembros | Información sobre la familia de librerías a la que pertenece. |
| Constructor (opcional) | Constructor de clase |
| Métodos (por cada método) | • Descripción del método<br>• Parámetros que recibe<br>• Respuesta (response) |
| Diseño (opcional para librerías wrapper) | Diagrama de clases |
| Instalación | Instrucciones de instalación |
| Preguntas frecuentes | Información con preguntas frecuentes que se presentan sobre la librería |
| Uso | Como utilizarla y ejemplos |

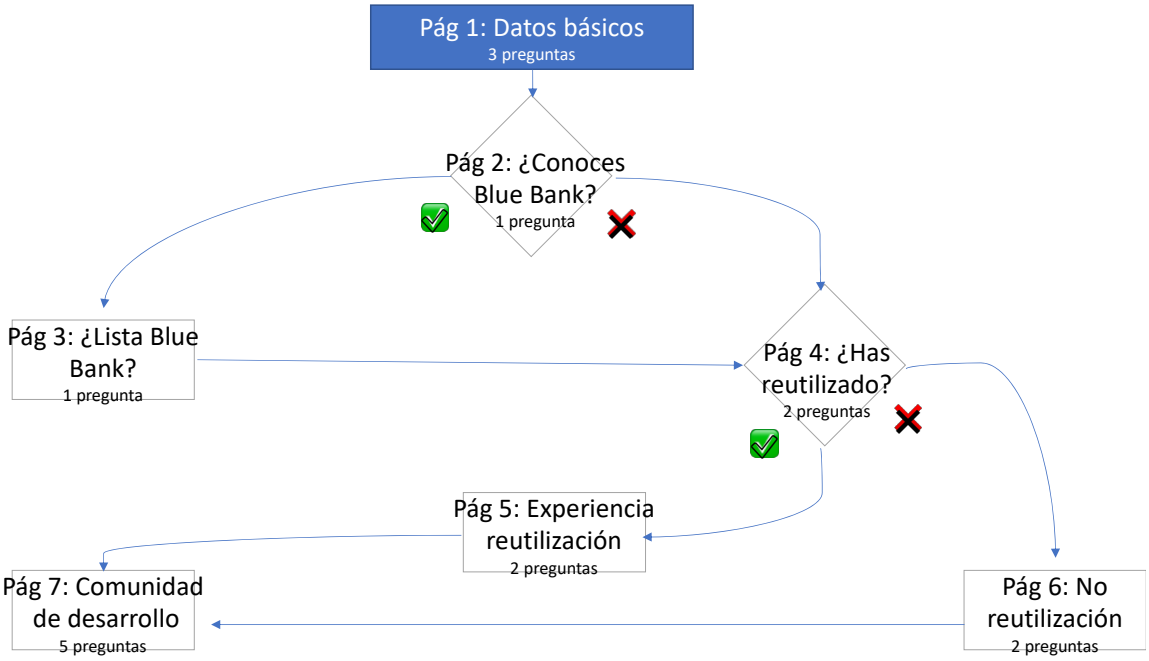
## 9.3 ANEXO C: ENCUESTAS

A continuación, se muestra tanto el diseño de las dos encuestas realizadas y el tabulado o graficado de sus resultados.

### 9.3.1 Encuesta de diagnóstico y análisis de resultados

En la sección DIAGNÓSTICO DE LA SITUACIÓN ACTUAL se hace una introducción a esta encuesta de diagnóstico. La encuesta realizada se diseñó a manera de un flujo, como se detalla en la Figura 50, para recopilar información de todos los funcionarios sin importar si hubiesen reutilizado o no el RSC o tuvieran experiencias de reutilización de activos de software.

Figura 50: Análisis encuesta de diagnóstico: flujograma




Esta encuesta se compartió a todo el Laboratorio mediante la herramienta Google Forms. El texto introductorio a la encuesta rezaba lo siguiente:

| Esta encuesta de 10 preguntas tiene como objetivo conocer el estado de reutilización de activos de |
|---|

software (documentación, código, servicios y más) y uso de los Servicios Comunes (Blue Bank o Nube Azul) del Laboratorio Digital para proponer mejoras que lleven a crear una comunidad de desarrollo de software, similar a las comunidades *open-source*, pero dentro del Laboratorio. Esta encuesta surge como iniciativa del proyecto de grados de la Maestría en Ingeniería de Software, proyecto que pretende ayudar a crear unas dinámicas sociales de colaboración entorno al desarrollo de software, sin importar tu rol. Nota: algunas preguntas se desactivan de acuerdo con tus respuestas anteriores.

 La privacidad de las respuestas

Tus respuestas serán anónimas para darte la tranquilidad de conteste con total libertad y sinceridad. En la parte de arriba puedes ver que no se comparte tu cuenta, lo que evita que se pueda realizar trazabilidad entre tus respuestas y tu cuenta de correo.

 Retomar la encuesta más tarde

Puedes cerrar la encuesta y retomar donde ibas en cualquier momento que tengas más tiempo o tranquilidad para contestarla. Para tal efecto, abres nuevamente el link que te llegó al correo electrónico.

 Uso de la información recolectada

La información recolectada se tratará, analizará y presentará en forma de gráficos agregados (resumidos) a los administrativos del Laboratorio y académicos de Javeriana; cuidando siempre que no se incluya información específica del negocio.

A continuación, se relacionan las preguntas planteadas.

#### 9.3.1.1 Preguntas básicas

Estas preguntas se plantearon a todas las personas.

Preguntas 1 y 2- La Figura 51 muestra el análisis de las preguntas básicas, que eran realizadas a todo el personal. La pregunta sobre la antigüedad en el Laboratorio se diseñó con el propósito de poder gestionar la rotación laboral.

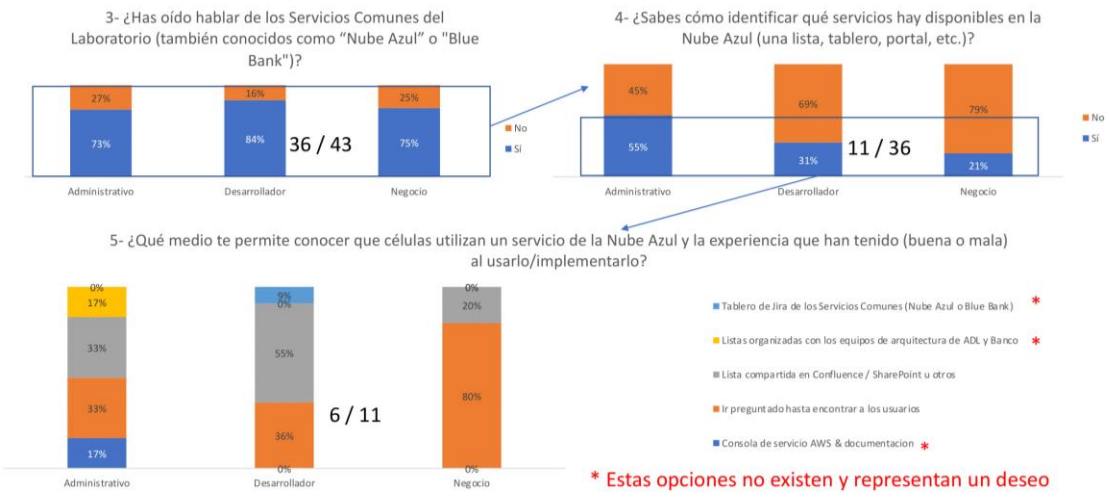
Figura 51. Análisis encuesta de diagnóstico: datos básicos

### 9.3.1.2 Preguntas sobre la Nube Azul

Estas preguntas se realizaron solo a las personas que afirmaron conocer la Nube Azul o Repositorio de Servicios Comunes (RSC). Vea la Figura 50 para entender el flujo de la encuesta.

Preguntas 3, 4 y 5. En la Figura 52 muestra el detalle de quienes contestaron “SÍ” a la pregunta 3, así mismo como aquellos que contestaron “SÍ” a la pregunta 4. Con respecto a la pregunta 5, en esta se introdujeron unos elementos que no existían (deseos) como manera de confirmar si realmente las personas conocían como identificar los servicios disponibles en la Nube Azul (pregunta 4). Los números que acompañan a esta imagen fueron de gran utilidad para mostrar a los Administrativos del Laboratorio que los roles de desarrolladores realmente desconocían, contrario a lo que ellos creían, el Repositorio de Servicios Comunes (RSC o Nube Azul); básicamente en la pregunta 5 se puede ver que solo 6 de 43 desarrolladores conocían la forma real de explorar el RSC, estas 6 personas corresponden a los Arquitectos, por lo cual se puede decir que hasta ese momento la información del RSC estaba clasificada para personas de alto rango.

Figura 52. Análisis encuesta de diagnóstico: conocimiento general del RSC o Nube Azul




### 9.3.1.3 Preguntas generales sobre reutilización

Estas preguntas se hicieron a todas las personas. Vea la Figura 50 para entender el flujo de la encuesta.

Pregunta 6- La Figura 53 permite ver que solo el 9 % de los desarrolladores estaría dispuesto

a hacer los cambios en la documentación de otra célula, otro 9 % tiene confianza que de mencionar las posibles mejoras la otra célula se tomaría en serio su corrección, mientras que el 51 % considera que mencionarlo no produciría efectos inmediatos ni tampoco haría seguimiento a que se realice. En la encuesta de evaluación (final) se podrá ver que este comportamiento se debe a un fuerte hermetismo de las células, tanto en abrir su documentación como en aceptar sugerencias. Este hermetismo luego es confirmado en la Encuesta de Evaluación que se encuentra en el apartado **Encuesta de evaluación y análisis de resultados**.

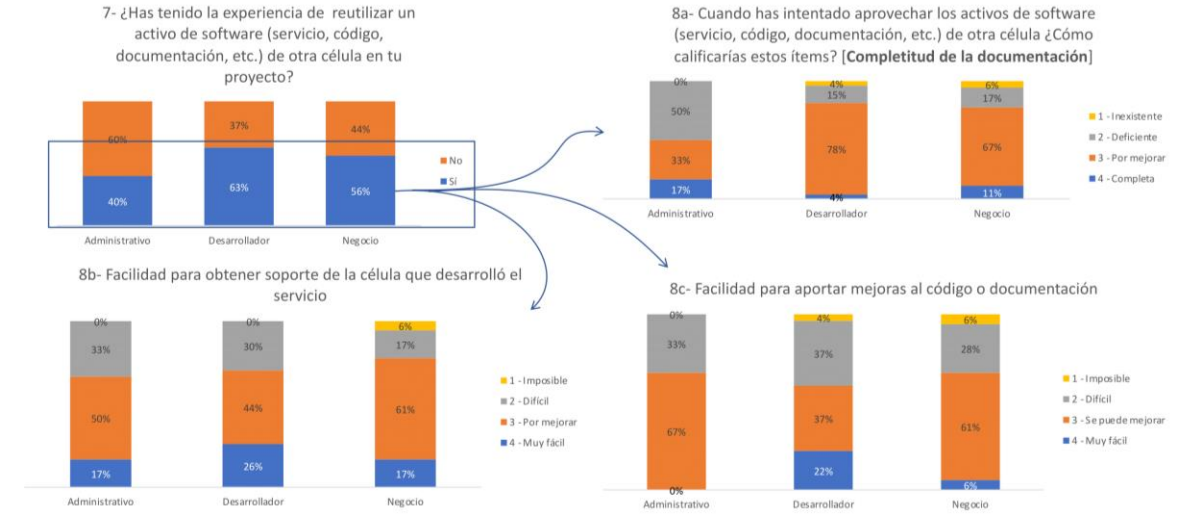
Figura 53. Análisis encuesta de diagnóstico: experiencia al reutilizar activos de software


#### 9.3.1.4 Preguntas sobre experiencia de reutilización

Estas preguntas se realizaron a las personas que contestaron “Sí” en la pregunta 7. Vea la Figura 50 para entender el flujo de la encuesta.

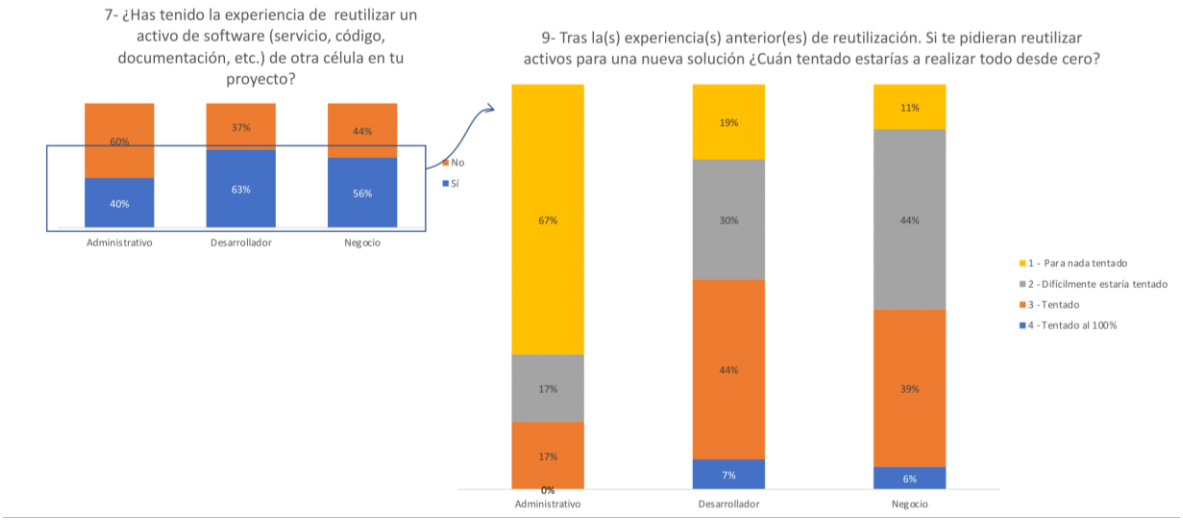
Pregunta 8- La Figura 54 muestra en detalle las respuestas de quienes contestaron que sí han tenido experiencia de reutilizar activos de software. En la pregunta 8 que es compuesta, se puede ver que toda la experiencia es ampliamente calificada como “por mejorar”.

Figura 54. Análisis encuesta de diagnóstico: experiencia al reutilizar activos de software #1




Pregunta 9- En la Figura 55 se puede observar que entre quienes han reutilizado activos de software, el 51 % ha tenido una experiencia negativa, por lo cual estarían dispuestos a hacer todo desde cero.

Figura 55. Análisis encuesta de diagnóstico: experiencia de reutilización de activos de software #2

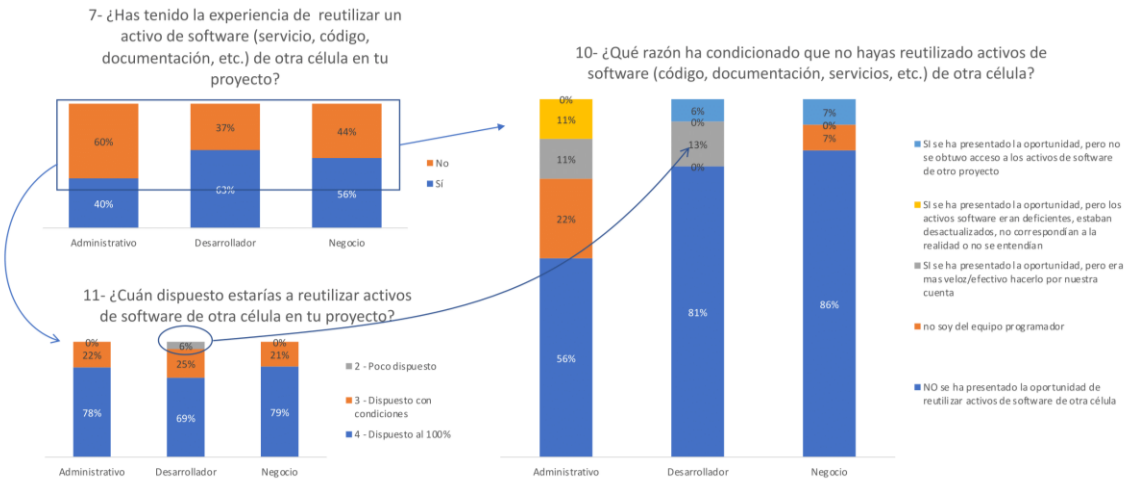



### 9.3.1.5 Preguntas sobre motivos de no contar con experiencia de reutilización

Estas preguntas se realizaron a las personas que contestaron “No” en la pregunta 7. Vea la Figura 50 para entender el flujo de la encuesta.

Preguntas 10 y 11- En la Figura 56 se grafican los resultados de consultar los motivos principales de la falta de experiencia de reutilización de activos de software, donde se puede ver que en la pregunta 10 al 81 % no se le ha presentado la oportunidad. Es importante resaltar que en la pregunta 11 el 6 % que estaría poco dispuesto a reutilizar es el mismo 13 % de la pregunta 10 que dice que, aunque ha tenido la oportunidad ha preferido hacer todo desde cero; en este caso podemos hablar de una renuencia a reutilizar, que es una posición natural de tribalismo que se espera se reduzca como efecto de la aplicación del Gobierno y un consecuente cambio de cultura.

Figura 56. Análisis encuesta de diagnóstico: motivos de no reutilización de activos



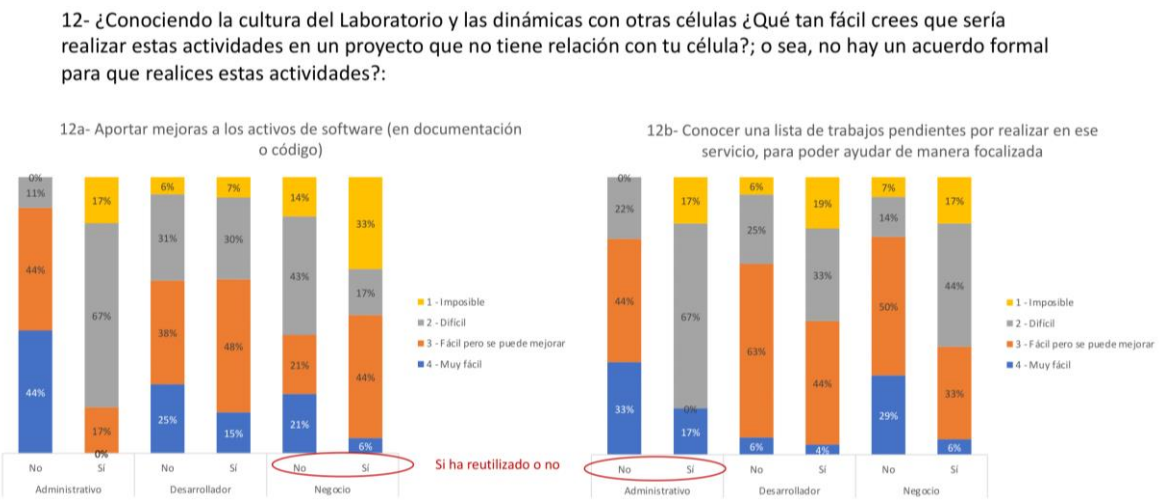

### 9.3.1.6 Preguntas sobre cultura de comunidad de desarrollo

Estas preguntas se hicieron a todas las personas. Vea la Figura 50 para entender el flujo de la encuesta.

Pregunta 12- En la Figura 57 se profundiza sobre preguntas que permitan determinar el nivel de apertura dentro del Laboratorio. En estas preguntas se contrasta la postura de quienes si han reutilizado software (ver Figura 54 y Figura 55) y los que no (ver Figura 56). En la pregunta 12a se puede observar que ambos grupos coinciden en una postura positiva del 63 %, pero entre quienes ya han tenido la experiencia de reutilizar activos consideran que la facilidad de aportes es un poco menor respecto al grupo que no. En la pregunta 12b entre quienes ya han reutilizado activos, solo el 48 % considera que las células harían apertura de sus tableros o listas de trabajo; nuevamente se evidencia el hermetismo de las

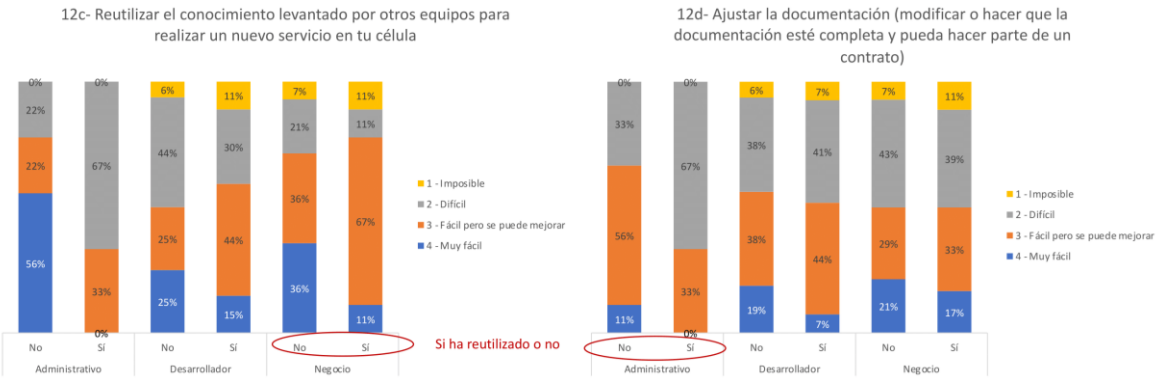
células que se confirma en la Encuesta de Evaluación que se encuentra en el apartado **Encuesta de evaluación y análisis de resultados**.

Figura 57. Análisis encuesta de diagnóstico: apertura de colaboración en el Laboratorio #1




Pregunta 12 (continuación)- En la Figura 58 se continúa con la pregunta 12 (ver Figura 57) y se puede apreciar que en cuanto a reutilizar el conocimiento levantado por otros equipos (pregunta 12c) los desarrolladores que ya han reutilizado activos son mas optimistas 59 % que quienes no 50 %. En cambio, en la pregunta 12d, al momento de preguntarles ya no de ser pasivo (consumir el conocimiento, pregunta 12c) sino pasar a la acción de ajustar, los desarrolladores que ya han tenido la experiencia son menos positivos (51 %) que quienes no la han tenido (57 %).

Figura 58. Análisis encuesta de diagnóstico: apertura de colaboración en el Laboratorio #2




Pregunta 13- En la Figura 59 se detalla la pregunta abierta sobre impedimentos a la apertura de colaboración en el Laboratorio. Se recibieron 79 respuestas, que para analizarlas se

realizó una nube de palabras para determinar aquellas que mas se repetían y posteriormente se filtraron las mas significativas de estas. Este análisis permitió entender las necesidades que debía cumplir el gobierno y las situaciones culturales para tener en cuenta durante su diseño.

Figura 59. Análisis encuesta de diagnóstico: apertura de colaboración en el Laboratorio (pregunta abierta)

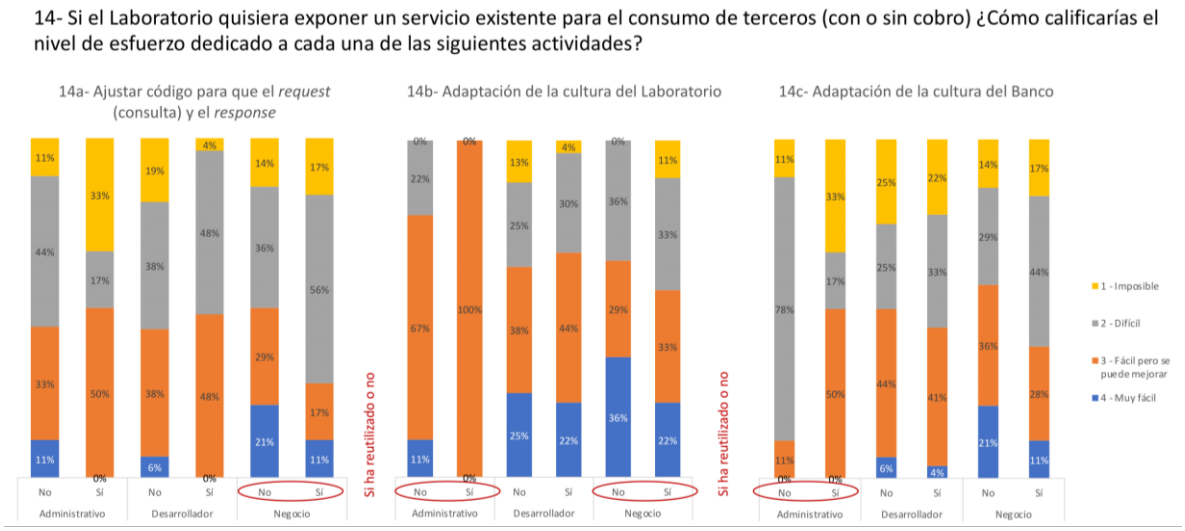
13- ¿Qué crees que hace falta para que todos los servicios existentes en el Laboratorio (estén o no en Nube Azul) puedan recibir aportes en código o documentación por parte de cualquier persona del Laboratorio sin importar su rol? Por favor, detalla tu respuesta.


79 respuestas completas

1. Tener un repositorio común de la información. Que todos tengan acceso al repositorio y que los arquitectos tengan actualizaciones constantes de la información/servicios que están implementados en el laboratorio
2. Conocer un poco más cuales son todos los servicios que se exponen en las otras células.
3. Creo que lo más importante es tener un panorama muy breve y concreto de la arquitectura de la Nube Azul y la relación (funcionamiento) de los servicios en ésta. Con el fin de realizar buenos aportes por parte de cualquier persona del Laboratorio.
4. Una documentación clara y sobretodo estandarizada que no solo explique como es el consumo del servicio; sino cómo está construido internamente; y claro, un readme claro de cada uno de los repositorios.
5. Divulgación inicial de como funciona Nube Azul, posterior a esto revisar como usar los servicios.
6. Mayor socialización y difusión, a todos los integrantes del laboratorio, de los servicios disponibles o mejoras que se le agregan a estos, mediante canales como Slack o Confluence
7. Asignar un dominio o gobierno a estos servicios
8. Primero habría que dar transparencia a todos los miembros del laboratorio acerca de los servicios disponibles, además de hacer saber a los mismos que estos pueden ser mejorados incluso por nosotros mismos. Porque personalmente siento que pareciera que quien lo hizo inicialmente, es quien tiene el "permiso" de actualizarlo.
9. Una vista global y centralizada que permita fácilmente conocer los servicios y conocer información relevante de los mismos.
10. Manejo de herramienta adecuada para la documentación y centralización de todos los servicios desarrollados y expuestos por las diferentes mesas de trabajo.
11. Un lineamiento claro para realizar cambios o mejoras de un servicio o todos los servicios en general, esto seria desde realizar el clone del repo o el fork hasta finalizar con el PR de la mejora (En el proceso regirse al lineamiento de que cosas no se deben tocar, buenas practicas etc..)
12. Muchas veces no tenemos acceso a los repos de dichos servicios, eso sería primordial, y la gestión de esos accesos muchas veces es muy difícil o nula.
13. Contexto sobre las necesidades del servicio y como impactaría al área de negocio o a otras células
14. Debemos migrar servicios de la nube común a la nube azul y de esta manera disponibilizarlos.

Pregunta 14- Esta pregunta estaba compuesta de una matriz de calificación para tres factores. En la Figura 60 se contrasta la pregunta 14 y la 7 (ver Figura 54) para poder ver si había cambios de percepción entre las personas que afirman tener experiencia de reutilización. En la sub-pregunta 14a se puede observar que el 54 % de los desarrolladores es pesimista con respecto a la dificultad de las adaptaciones planteadas, y que aquellos que han tenido experiencia de reutilización son mas pesimistas que quienes no (6 %). No obstante, en la sub-pregunta 14b el 65 % de los desarrolladores son optimistas en cuanto a las capacidades del Laboratorio para cambiar su cultura respecto a la situación hipotética planteada en la pregunta; además no hay diferencias notables entre los experimentados en reutilización. La sub-pregunta 14c contrasta la cultura del Laboratorio y el Banco, donde solo el 47 % de los desarrolladores se presentan como optimistas; además, el pesimismo aumenta entre aquellos que afirman tener experiencia de reutilización, lo cual se ve reafirmado en la pregunta 15 (ver Figura 61).

Figura 60. Análisis encuesta de diagnóstico: hipotética exposición de servicios

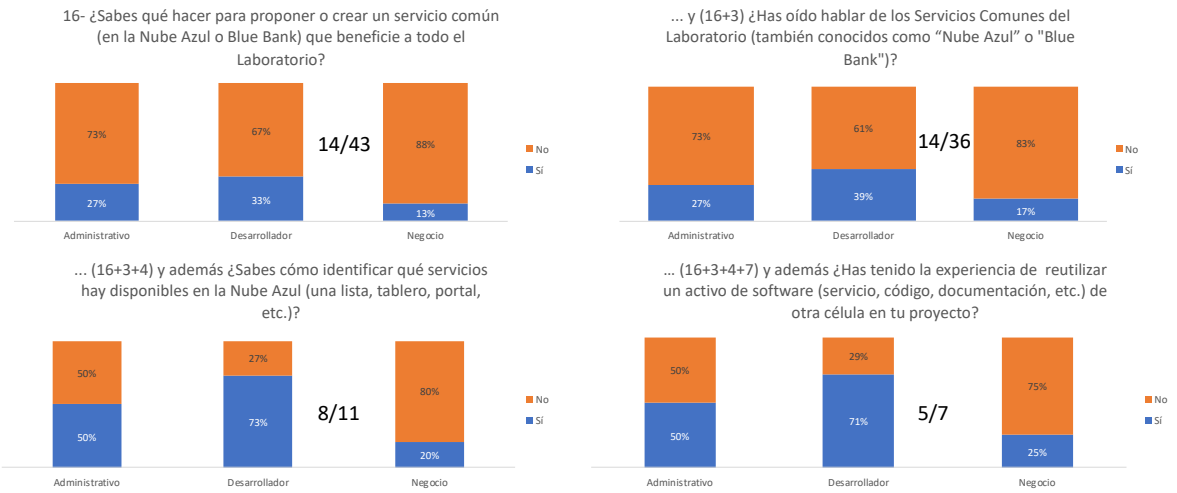



Pregunta 15- En la Figura 61 se detalla la pregunta abierta sobre impedimentos a la apertura de colaboración del Laboratorio y entidades externas al Banco como podrían ser *Fintechs*. Para analizar las respuestas se realizó una nube de palabras para determinar aquellas que mas se repetían y posteriormente se filtraron las mas significativas de estas. Este análisis permitió entender las necesidades que debía cumplir el gobierno y las situaciones culturales para tener en cuenta durante su diseño.

Figura 61. Análisis encuesta de diagnóstico: hipotética exposición de servicios (pregunta abierta)

Pregunta 16- Esta pregunta se diseñó como una pregunta de confirmación para entender si realmente las personas conocían el RSC (o Blue Bank) mas allá que las menciones de pasillo. En la Figura 62 se puede observar que solo el 33 % de los desarrolladores afirman saber como proponer o crear un servicio común. En los siguientes análisis se contrasta esta pregunta con otras preguntas de confirmación como forma de asegurarse de que se excluya a aquellas personas que contestan que conocen cosas que desconocen por miedo a ser juzgados, a pesar de que se informó de que la encuesta era anónima. Ante este efecto, se esperaría que, al combinar las otras preguntas de confirmación, la variación numérica fuera muy poca. Las siguientes partes de la Figura 62 muestran una serie de secuencias, donde cada una de las preguntas de confirmación fueron integrándose una a una, profundizando cada vez en estas cuestiones entre aquellos que contestaron "Sí". Finalmente, se puede ver que solo 5 desarrolladores que afirman saber como disponibilizar servicios comunes, además han escuchado del RSC (Nube Azul) (pregunta 3, ver Figura 52), saben realmente como identificar la documentación de los servicios (pregunta 4, ver Figura 52) y también han tenido experiencia reutilizando (pregunta 7, ver Figura 54). Este número (5) es muy similar al confirmado en la pregunta 5 (ver Figura 52), que correspondería a los Arquitectos, reconfirmando que el conocimiento sobre el RSC es una información que solo existe en un circulo cerrado del Laboratorio.

Figura 62. Análisis encuesta de diagnóstico: proponer servicios reutilizables

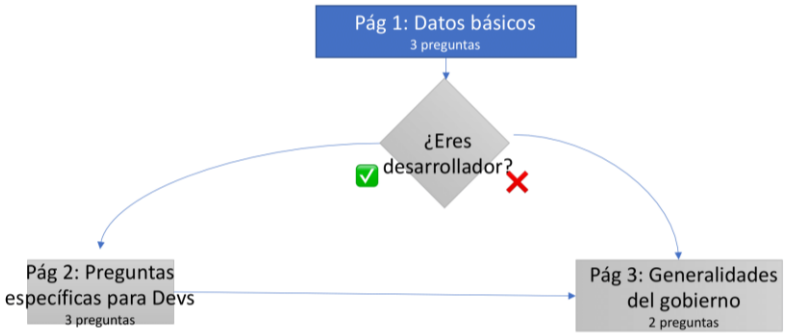



### 9.3.2 Encuesta de evaluación y análisis de resultados

En la sección **VALIDACIÓN DE LA IMPLEMENTACIÓN** se hace una introducción a esta encuesta de evaluación. La encuesta realizada se diseñó a manera de un flujo, como se

detalla en la Figura 63, para recopilar información de todos los funcionarios sobre su opinión respecto al Gobierno del Repositorio de Servicios Comunes (RSC) desarrollado en este proyecto de grado.

Figura 63: Análisis encuesta de evaluación: flujograma



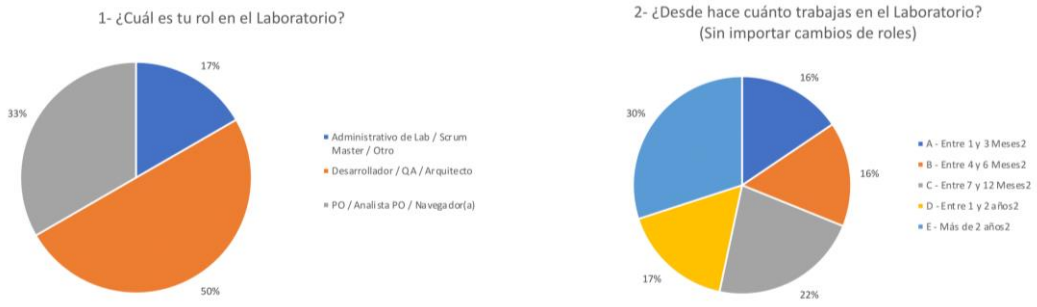

Esta encuesta se compartió a todo el Laboratorio mediante la herramienta Google Forms. El texto introductorio a la encuesta fue el mismo utilizado en la encuesta de diagnóstico y que podrá ver en la sección **Encuesta de diagnóstico y análisis de resultados**.

A continuación, se relacionan las preguntas planteadas.

#### 9.3.2.1 Preguntas básicas

Preguntas 1 y 2- La Figura 64 muestra el análisis de las preguntas básicas, que eran realizadas a todo el personal. La pregunta sobre la antigüedad en el Laboratorio se diseñó con el propósito de poder gestionar la rotación laboral.

Figura 64. Análisis encuesta de evaluación: datos básicos

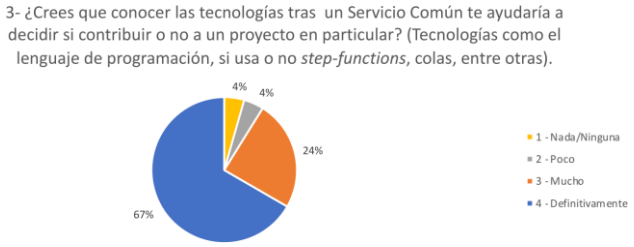


#### 9.3.2.2 Preguntas para desarrolladores

Estas preguntas solo se formularon a los roles de desarrollo: Arquitecto, QA y Desarrollador.

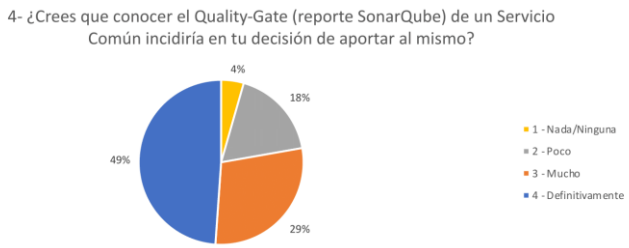
Pregunta 3- En la Figura 65, el 92 % de los desarrolladores considera importante el conocer las tecnologías tras los servicios para poder determinar en cuales podrían aportar.

Figura 65. Análisis encuesta de evaluación: tecnologías tras los servicios



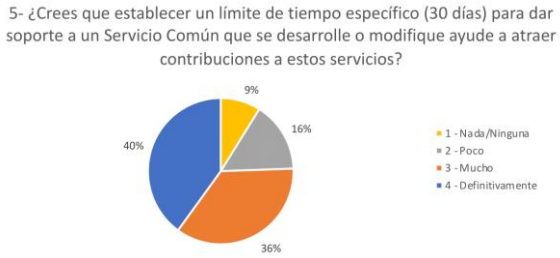

Pregunta 4- Ver Figura 66. Aunque el 78 % da una respuesta positiva, que el 22 % considere la Calidad de Desarrollo de Software como algo accesorio o poco fundamental habla de la necesidad de interiorizar estos conceptos en toda la plantilla, pues estamos hablando de casi 1 de cada 4 roles técnicos. Las entrevistas informales revelan que es difícil hacer que se realicen los ajustes de Calidad sobre los servicios una vez estos se encuentran en producción, bajo excusas como que ya es productivo y no representa riesgo, pero el riesgo se materializa durante los procesos de evolución de los servicios, pues no cuentan con la "red de seguridad" que suponen entre otras las pruebas unitarias al momento de modificar los servicios.

Figura 66. Análisis encuesta de evaluación: indicadores de calidad del proceso de desarrollo




Pregunta 5- Ver Figura 67. El 76 % de los desarrolladores considera positivo el establecer un tiempo específico para dar soporte a los desarrollos en la Nube Azul.

Figura 67. Análisis encuesta de evaluación: garantía de 30 días



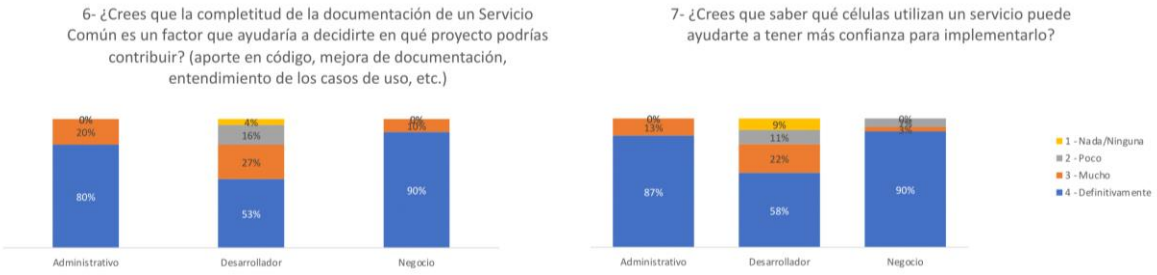

### 9.3.2.3 Preguntas para todos los roles

Estas interrogantes se plantearon a todos los roles del Laboratorio.

Pregunta 6- Ver Figura 68, el 80 % de los desarrolladores votan como importante la completitud de la documentación para decidirse en cual contribuir. Los roles no técnicos también lo consideran importante, incluso en proporciones mayores.

Pregunta 7- Ver Figura 68, las respuestas a esta pregunta son muy similares a la anterior pregunta. Para el 80 % de los desarrolladores conocer que células implementan el servicio les genera mayor confianza.

Figura 68. Análisis encuesta de evaluación: información sobre los servicios

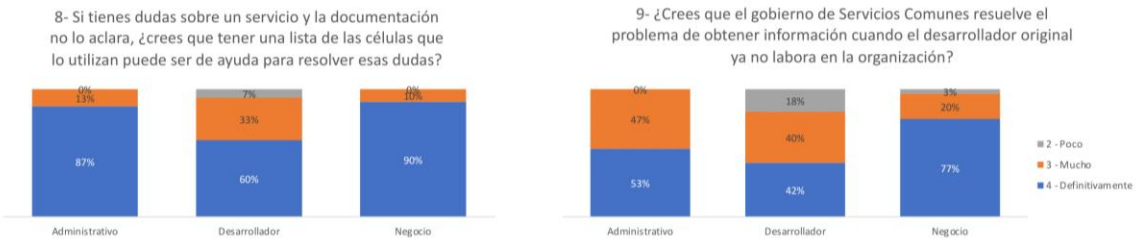



Pregunta 8- Ver Figura 69. Esta pregunta se diseñó como un control de la pregunta 7, como una forma de confirmar la información recabada en la pregunta anterior. Analizando los resultados presentados en la Figura 69, el 93 % de los desarrolladores, es importante o positivo saber quienes han implementado el servicio para poder obtener apoyo o ayuda de ellos.

Pregunta 9- De acuerdo con la Figura 69, para el 82 % de los desarrolladores considera que el Gobierno es importante para obtener información cuando el desarrollador ya no labora

en la organización. De este 82 %, el 42 % considera que el aporte es vital. La pregunta 9 se planteó a manera de confirmación o profundización de la 8.

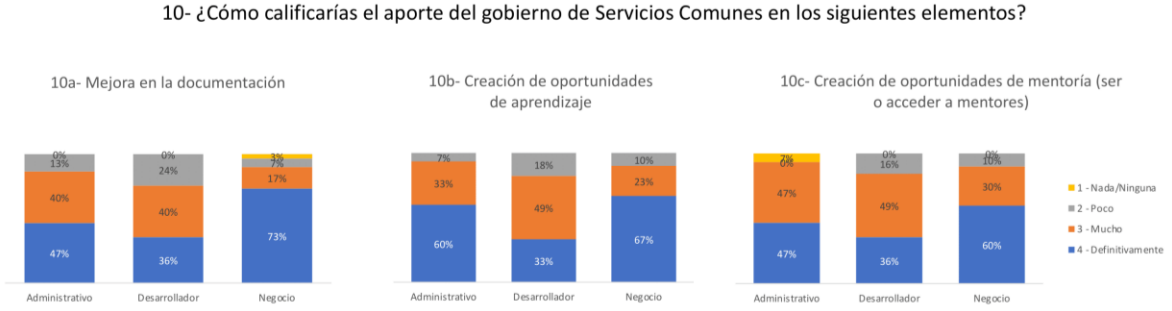
Figura 69. Análisis encuesta de evaluación: descubrimiento de información




Pregunta 10- Se le solicita al personal que califique varios elementos. Sus respuestas se grafican en la Figura 70 y la Figura 71. Detallando o analizando su información se puede encontrar que (ver Figura 70):

- Pregunta 10a- El 76 % de los desarrolladores, (3 de cada 4) considera positivo el aporte que realiza el Gobierno respecto a la mejora o completitud de la documentación.
- Pregunta 10b- El 82 % de los desarrolladores, considera positivo el aporte que realiza el Gobierno respecto a la creación de oportunidades de aprendizaje. Queda pendiente documentar ejemplos específicos de aprendizaje para roles no técnicos del Laboratorio; esto último como resultado de una entrevista informal con un *Scrum Master*.
- Pregunta 10c- El 85 % de los desarrolladores, considera positivo el aporte que realiza el Gobierno respecto a la creación de oportunidades de mentoría. Queda pendiente documentar ejemplos específicos de mentoría para roles no técnicos del Laboratorio; esto último como resultado de una entrevista informal con un Scrum Master.

Figura 70. Análisis encuesta de evaluación: calificación de elementos del Gobierno #1

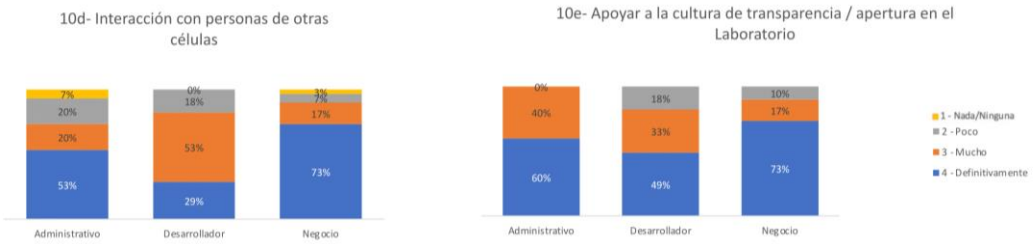



Por su parte, en la Figura 71 se da continuidad a los análisis de la pregunta 10, detallando

que:

- Pregunta 10d- El 82 % de los desarrolladores, considera positivo el aporte que realiza el Gobierno respecto a la interacción con personas de otras células. No obstante, de estos, el 53 % no lo consideran definitivo. En entrevistas informales a varios roles y también preguntas abiertas de esta encuesta, dan a entender que aún hay mucho hermetismo en el Laboratorio.
- Pregunta 10e- El 82 % de los desarrolladores, considera positivo el aporte que realiza el Gobierno respecto a la apertura dentro del Laboratorio, que, según otras preguntas abiertas de esta encuesta y unas entrevistas informales a varios roles, sigue siendo muy hermética. En contraste a la pregunta anterior, en este caso es mayor el número de personas que considera que este Gobierno es un paso definitivo hacia esta dirección.

Figura 71. Análisis encuesta de evaluación: calificación de elementos del Gobierno #2



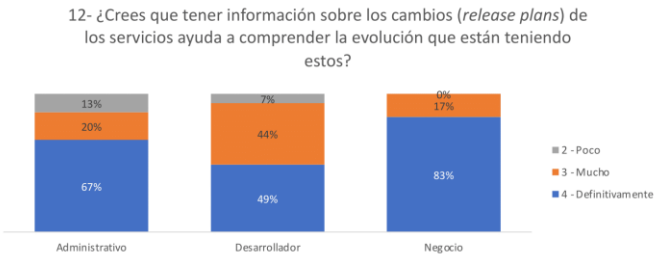

Pregunta 11- ¿Qué preguntas o situaciones crees que el gobierno no contesta (o no lo hace bien)? Esta pregunta era abierta y el 79 % de los encuestados consideran que está bien, completo u otra respuesta afirmativa. 19 personas dieron otras respuestas que se pueden clasificar como:

- Responsabilidad: consideran que el equipo no está maduro para autogestionarse o que se requiere un supervisor, que las personas de por si mismas no van a ser capaces de documentar de manera seria y responsable.
- Como aportar: en los ejemplos de como aportar se pusieron ejemplos para los roles de técnicos y de negocio, pero faltó ejemplos específicos para otros roles. Hacer una sección de “¿Que puedo aprender?”
- Notificaciones: explicar como notificar a las células que utilizan un servicio que se modificará.
- Resolución de impedimentos: se solicita documentar, pero significa que hay que hacer mayor énfasis de que se pretende que el Laboratorio sea una comunidad y entremos en contacto.
- Salida a producción: documentar el procedimiento completo.
- Otras: preguntas que ya están contestadas en el Gobierno, pero parece que se perdieron en la lectura rápida que hicieron los encuestados, pero que son indicios

de mejora o simplificación: datos de pruebas, Swagger, archivos Postman, etc. los tres están contenidos en el Swagger, así como los datos de contacto de los servicios que van a terceros. También preguntan sobre Arquitectura de los servicios y tecnologías en los que están desarrollados que son parte medulares y obligatorias de la documentación solicitada.

Pregunta 12- Ver Figura 72. El 93 % de los desarrolladores, considera positivo tener los *release plans* (controles de cambios) como forma de entender la evolución de los servicios.

Figura 72. Análisis encuesta de evaluación: información de evolución de los servicios




Pregunta 13- ¿Qué sección agregarías a la documentación solicitada por servicio (link de ejemplo de un servicio)? Esta pregunta era abierta y el 67 % de los encuestados consideran que está bien, completo u otra respuesta afirmativa. Se debe anotar que esta pregunta es una validación de la anterior pregunta abierta, como una forma de recabar mas detalle. 30 personas dieron otras respuestas que se pueden clasificar como:

- Responsabilidad: consideran que el equipo no está maduro para autogestionarse o que se requiere un supervisor, que las personas de por si mismas no van a ser capaces de documentar de manera seria y responsable. En la pregunta abierta anterior se habló de lo mismo.
- Documentación: uno de los encuestados considera necesario aumentar la documentación con información recabada por los roles de negocio o que incluso tenga pasos a pasos para consumir el servicio.
- Información de incidentes: reporte de incidentes ocurridos en producción.
- Negociaciones: información sobre temas como las áreas del Banco que entregan o regulan determinada información, así como acuerdos de soporte a fallos por parte de los servicios de terceros.
- Resolución de impedimentos: se solicita documentar, pero significa que hay que hacer mayor énfasis de que se pretende que el Laboratorio sea una comunidad y entremos en contacto. En la pregunta abierta anterior se habló de lo mismo, por personas diferentes a las que lo hicieron en esta oportunidad.
- Analítica: informar si el servicio reporta a analítica y como lo hace.
- Otras: preguntas que ya están contestadas en el Gobierno, pero parece que se perdieron en la lectura rápida que hicieron los encuestados, pero que son indicios de mejora o simplificación: datos de pruebas, Swagger, archivos Postman, etc. los

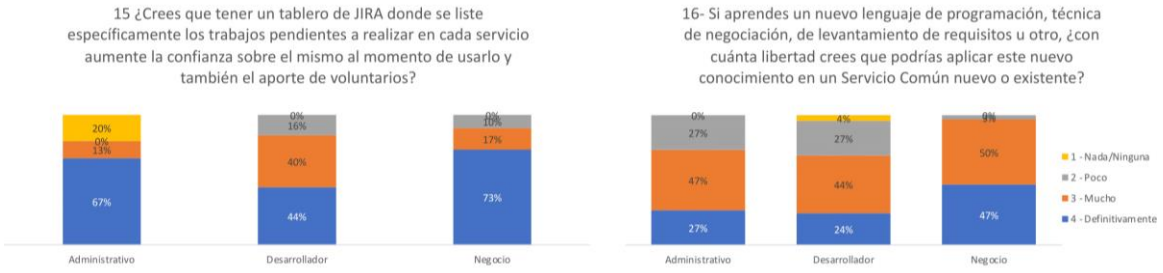
tres están contenidos en el Swagger. También preguntan sobre Arquitectura de los servicios y tecnologías en los que están desarrollados que son parte medulares y obligatorias de la documentación solicitada, así como los datos de contacto de los servicios que van a terceros. En la pregunta abierta anterior se habló de lo mismo.

Pregunta 14- ¿Qué sección eliminarías o cambiarías de la documentación solicitada por servicio? Esta pregunta era abierta y se diseñó para ser el "control" de la pregunta 17, como una forma de recabar mas información sobre el mismo tópico. En cuanto a las respuestas, el 99 % de los encuestados consideran que el Gobierno está bien, completo u otra respuesta afirmativa. Se debe anotar que 8 personas dieron respuestas que realmente se trataban de adiciones al Gobierno que harían a la documentación y que se encontraban analizadas en la pregunta anterior. Solo hubo una persona que sugirió designar un responsable que esté atento a eliminar información desactualizada o ya deprecada, lo cual también se analizó en las preguntas 15 y 17.

Pregunta 15- Ver Figura 73. El 84 % de los desarrolladores, considera positivo tener un tablero JIRA donde se recojan los trabajos pendientes a realizar en cada servicio. Al hacer una entrevista con un Administrativo, este consideraba que no era necesario un tablero JIRA pues se podría manejar dentro de cada proyecto. Se tuvo que justificar todas las incomodidades que generaba intentar manejarlo en los tableros de cada célula o con herramientas externas, logrando comprender la necesidad de esta herramienta.

Pregunta 16- Ver Figura 73. El 68 % de los desarrolladores, considera que el Gobierno brinda la oportunidad de probar nuevas cosas. El 31 % restante es un número muy similar a la cantidad de personas que afirmaban no tener experiencia de reutilización en la encuesta anterior y que en las entrevistas informales también informaban de poca apertura entre los funcionarios del Laboratorio.

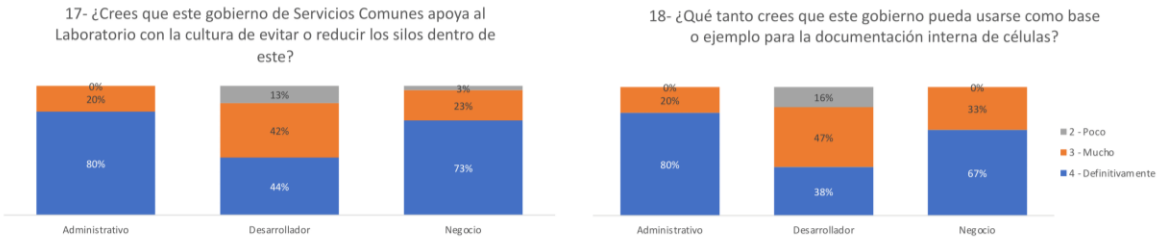
Figura 73. Análisis encuesta de evaluación: lista de trabajos y aprendizaje




Pregunta 17- Ver Figura 74. En contraste a la pregunta anterior, el 86 % de los desarrolladores consideran que el Gobierno es una oportunidad de mejora a la reducción de Silos dentro del Laboratorio. En las entrevistas informales hablaban de que no haría el cambio per-sé, pero si ayudaba a visibilizar este impedimento.

Pregunta 18- Ver Figura 74. El 86 % de los desarrolladores votan como positivo que el Gobierno sirva como base a la documentación interna de las células. Durante las entrevistas informales conminaban a verificar las buenas prácticas dentro de todas las células y hacerlas públicas, pues hasta ahora el hermetismo no ha permito explotar esta posibilidad.

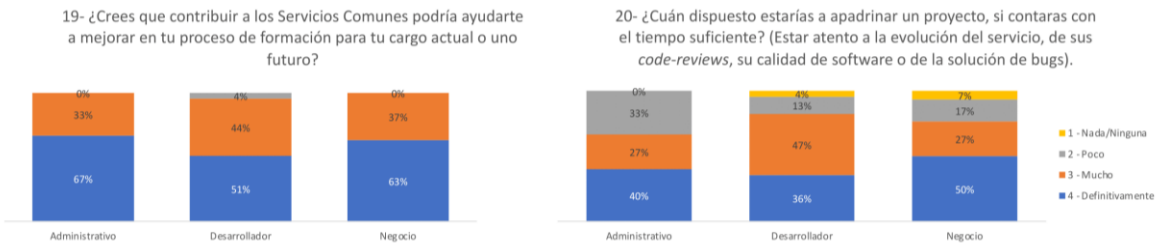
Figura 74. Análisis encuesta de evaluación: cultura y estandarización




Pregunta 19- Ver Figura 75. Con un rotundo sí, el 95 % de los desarrolladores considera que el Gobierno puede ayudar a contribuir a su proceso de formación. En las entrevistas informales, queda claro que se deben encontrar ejemplos para que los roles no técnicos puedan comprender como este Gobierno podría ser de utilidad para ellos.

Pregunta 20- Ver Figura 75. El 83 % de los desarrolladores considera probable el poder apadrinar a un proyecto compartido en caso de contar con el tiempo suficiente, lo mismo los roles de negocio. En los roles administrativos es un poco menor, pero unas entrevistas informales dan como resultado es que se debe trabajar por encontrar ejemplos de como ellos podrían aportar en esta situación de padrinazgo.

Figura 75. Análisis encuesta de evaluación: crecimiento y voluntariado




Pregunta 21- A nivel general, ¿qué crees que limitaría la aplicación, continuidad o evolución del gobierno de Servicios Comunes? Esta pregunta era abierta y el 57 % de los encuestados consideran que el gobierno no tendría inconvenientes en su evolución o continuidad. El 43 % consideran que se requiere de un responsable, alegando temas de cultura y que aún los integrantes del Laboratorio no logran el nivel de madurez que les permita aportar sin que haya alguien revisando la calidad de ese aporte. En este grupo se encontró una persona que afirma que la documentación es muy poca, pero en la pregunta sobre las secciones que agregarían informan que está bien, pero que se requiere un responsable.