

**UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL**

**REDISEÑO DEL PROCESO DE DESARROLLO DE
PRODUCTOS DIGITALES A TRAVÉS DE UN ENTORNO ÁGIL
Y UN MODELO DEVOPS**

PROYECTO DE GRADO PARA OPTAR AL GRADO DE MAGÍSTER EN
INGENIERÍA DE NEGOCIOS CON TECNOLOGÍAS DE INFORMACIÓN

JULIO ENRIQUE FRANJOLA ARMIJO

**PROFESOR GUÍA:
EZEQUIEL MUÑOZ KRSULOVIC**

**MIEMBROS DE LA COMISION:
EDUARDO CONTRERAS VILLABLANCA
MÓNICA CORTES HIDALGO**

SANTIAGO DE CHILE
2021

RESUMEN EJECUTIVO

Banco SCL es una institución bancaria líder en la industria financiera chilena, con el objetivo de satisfacer las necesidades de los clientes. Destaca en sus valores la orientación al cliente, entendiendo que, como toda empresa de servicios, el cliente es la razón de ser.

Banco SCL tiene como objetivo ser el mejor banco para los clientes y para ello se ha propuesto rediseñar su propuesta de valor para entregar mejores servicios basados en la anticipación de las necesidades de los clientes y entregando un servicio digital que facilite la relación de los clientes con los productos y servicios que otorga el banco. Para ello se ha implementado el programa de transformación digital que propone rediseñar, entre otras cosas, el proceso de construcción de software que permita construir y entregar los productos digitales tan pronto como sea posible a los clientes.

Para realizar este proyecto, en primera instancia fue necesario realizar un análisis de la problemática organizacional con respecto a las dificultades de los distintos equipos que forman parte del proceso de construcción y entrega de software. Luego, con ayuda de la metodología de ingeniería de negocios, se realizó el rediseño de los procesos de la etapa de construcción y entrega de software. Como parte del trabajo de investigación se definió utilizar SAFe como el marco de referencia metodológico en conjunto con la implementación de un modelo de trabajo basado en DevOps, además de la incorporación de herramientas compatibles con la tecnología del banco.

El trabajo realizado en este proyecto considero aplicar prácticas y principios de DevOps que permiten automatizar en gran parte los procesos manuales, lo que permite disminuir tiempos y aumentar la calidad del producto final. Dentro de las actividades a destacar, se considera el proceso de gestión del cambio para acompañar la transformación de la mejor manera posible, además de un profundo cambio cultural en la organización, que, junto a la metodología ágil ya existente, proporcionan a Banco SCL la capacidad de lanzar sus productos y soluciones digitales en un menor tiempo, de esta manera, es posible anticiparse a otros banco e instituciones financieras del mercado chileno, esto permite el cumplimiento de la estrategia planteada de reducir los tiempos de desarrollo y de entrega de las soluciones digitales para que los productos digitales del banco estén disposición de los clientes en el menor tiempo posible.

DEDICATORIA

A Julia y German, mis padres, quienes han sido mis guías.

A German, mi hermano, con quien me encantaría compartir este momento.

A Marcela, mi mujer, mi inspiración y compañera de viaje.

A Pedro, Simón y Borja, mis hijos, son el motor que me impulsa para seguir siempre adelante.

AGRADECIMIENTOS

A mis padres, gracias por su amor incondicional y por todas sus enseñanzas de vida.

A mi hermana, Alejandra, gracias por enseñarme de pequeño a desarrollar mi curiosidad.

A Marcela, gracias por todo el apoyo de siempre, por darme la fuerza y hacerme creer que sí era posible.

A Ezequiel Muñoz, mi profesor guía en este trabajo, por su dedicación y constante motivación.

A Laura y Ana María por su excelente disposición durante todos estos años de MBE.

A la Universidad de Chile y al MBE por permitirme ser parte de este gran Magister.

TABLA DE CONTENIDO

CAPÍTULO 1: INTRODUCCIÓN.....	1
1.1 Problema del Proyecto	2
1.2 Objetivos del Proyecto.....	3
1.2.1 Objetivo General.....	3
1.2.2 Objetivos Específicos	3
1.2.3 Resultados esperados.....	4
1.3 Justificación y Relevancia del Proyecto.....	4
1.3.1 Justificación.....	4
1.3.2 Relevancia.....	5
1.4 Solución Propuesta	5
CAPÍTULO 2: MARCO TEÓRICO.....	6
2.1 Metodología de Ingeniería de Negocios	6
2.2 Arquitectura de Procesos de Negocios.....	7
2.3 Revisión de Literatura Relevante	10
2.3.1 Metodología ágil	10
2.3.2 Modelo Tradicional Vs Ágil	11
2.3.3 Scrum	13
2.3.4 Kanban	15
2.3.5 SAFe	16
2.3.6 DevOps	18
2.3.7 Prácticas de DevOps.....	21
2.3.8 DevOps y agilidad	23
CAPÍTULO 3: PROYECTO	25
3.1 Antecedentes de la empresa.....	25
3.2 Levantamiento de la Situación Actual.....	26
3.2.1 Servicio compartido Desarrollo de Software.....	28
3.2.2 Cadena de Valor del servicio compartido Desarrollo de Software.....	28
3.2.3 Administración de Relación con el Usuario	30
3.2.4 Gestión de Producción y Entrega de Software	31
3.2.5 Planificación y Control Producción	32
3.2.6 Producción y Entrega de Software	33
3.2.7 Producción de Software	34
3.2.8 Entrega de Software.....	35
3.3 Conclusión del levantamiento	36
3.4 Modelo conceptual de los procesos impactados	37
3.5 Diagnóstico de la Situación Actual.....	38

3.5.1	Problemas Identificados	38
3.6	Generación de Alternativas	42
3.7	Evaluación de Alternativas	46
3.8	Propuesta de Solución	49
3.8.1	Rediseño del proceso Administración de relación con el Usuario	49
3.8.2	Rediseños Gestión de producción y entrega de Servicios TI	51
3.8.3	Rediseño del proceso Producción y entrega de Software	53
3.8.4	Conclusión del rediseño	57
3.9	Plan de Implementación y Acción	57
3.9.1	Gestión del cambio	58
CAPÍTULO 4: IMPLEMENTACIÓN APLICADA		63
4.1	Adopción de GitLab	64
4.2	Gitflow	65
4.2.1	Rama Main y Develop	65
4.2.2	Rama Feature	65
4.2.3	Rama Release	66
4.2.4	Rama HotFix	66
4.3	Despliegue automatizado con Jenkins	67
4.4	Testing continuo	68
4.4.1	BDD (Behavior Driven Development)	69
4.4.2	Cucumber y Gherkin	69
4.4.3	Pruebas Unitarias	72
4.4.4	Pruebas Automatizadas	73
CAPÍTULO 5: CONCLUSIONES Y TRABAJOS FUTUROS		76
5.1	Trabajo realizado	76
5.2	Lecciones aprendidas	76
5.3	Impacto del rediseño	77
5.4	Trabajo futuro	77
CAPÍTULO 6: BIBLIOGRAFÍA		78

ÍNDICE DE IMÁGENES

Figura 1. Ambientes y defectos detectados.....	4
Figura 2. Ontología de Ingeniería de negocios.....	6
Figura 3. Arquitecturas de macroprocesos	8
Figura 4. Tipos de Arquitectura	9
Figura 5. Diagrama de modelo tradicional. Entrega de valor al final.....	11
Figura 6. Diagrama de modelo ágil. Entrega de valor constante.	12
Figura 7. Diagrama del Proceso Scrum	13
Figura 8. Diagrama del Proceso Kanban	16
Figura 9. Beneficios de implementar SAFe	17
Figura 10: Diagrama SAFe.....	18
Figura 11. Diagrama de DevOps.....	19
Figura 12. Etapas de DevOps	20
Figura 13. Pirámide de Cohn.....	23
Figura 14. Arquitectura de Banco SCL	26
Figura 15. Macro Servicio Compartido Desarrollo de Software	28
Figura 16. Cadena de Valor del Servicio Compartido Desarrollo de Software	29
Figura 17. Administración de relación con el Usuario	30
Figura 18. Gestión de producción y Entrega de Software	31
Figura 19: Gestión de producción y entrega de software.....	32
Figura 20: Producción y entrega de software	33
Figura 21: Producción de Software.....	34
Figura 22: Entrega de Software	35
Figura 23: Proceso de despliegue	36
Figura 24. Procesos impactados.....	37
Figura 25. Diagrama Ishikawa.....	39
Figura 26: Proceso Administración de relación con el usuario	50
Figura 27: Proceso Gestión Producción y Entrega de Servicios TI	52
Figura 28: Proceso Producción y entrega de software.....	54
Figura 29: Evolución del flujo de despliegue de Software	57
Figura 30. Modelo de implementación	64
Figura 31. Rama Main	65
Figura 32. Rama Feature	65
Figura 33. Rama Release.....	66
Figura 34. Rama HotFix.	66

Figura 35. Ejecución de despliegue por Jenkins	67
Figura 36. Tiempos de ejecución de despliegue	67
Figura 37. Esquema de Testing Continuo.....	68
Figura 38. Flujo de trabajo BDD.....	69
Figura 39. Ejemplo de escenario escrito en Gherkin.....	70
Figura 40. Ejemplo de Gherkin.....	71
Figura 41. Ejemplo de defectos detectados en Pruebas Unitarias	72
Figura 42. Informe de ejecución de pruebas automatizadas en APIs.....	73
Figura 43. Informe de ejecución de pruebas automatizadas utilizando BDD	74
Figura 44. Informe de ejecución de pruebas automatizadas utilizando BDD	74

ÍNDICE DE TABLAS

Tabla 1: Ágil Vs Cascada	12
Tabla 2: Principios Agiles y DevOps fuertemente alineados	24
Tabla 3: Alternativas de solución.....	42
Tabla 4: Evaluación de alternativas.....	46
Tabla 5: Épica Vs Proyecto	49
Tabla 6: Matriz de proceso Administración de relación con el usuario	50
Tabla 7: Matriz de proceso Gestión Producción y Entrega de Servicios TI	52
Tabla 8: Matriz de proceso de producción y entrega de Software	55
Tabla 9: Plan de implementación	59
Tabla 10: Estructura Encuesta del Modelo de Madurez	60
Tabla 11: Preguntas del modelo de adopción:.....	60
Tabla 12: Descripción de herramientas para la implementación.....	63

CAPÍTULO 1: INTRODUCCIÓN

En Chile, durante el año 2019, las instituciones financieras generan ofertas agresivas de sus productos de consumo e hipotecario, esto con el objetivo de aumentar sus colocaciones y mejorar la posición de la institución en el mercado, además de fuertes campañas de fidelización con sus respectivas alianzas estratégicas para mantener y atraer nuevos clientes. Por otra parte, el cliente financiero ha sufrido una evolución considerable y es un consumidor cada vez más informado y por consecuencia es un cliente más educado y con mayores exigencias para los bancos e instituciones financieras en general.

Banco SCL es una institución financiera líder del mercado, con el objetivo de satisfacer las necesidades que los clientes le demandan como banco. Una de las principales características de Banco SCL es poner al cliente en el centro de todo lo que se propone para brindar una experiencia simple, personal y justa con el objetivo de ser el mejor banco. Dado el interés de banco SCL por mantenerse como líder en la industria financiera chilena, y ante la posibilidad de perder cuota de mercado frente a sus competidores más cercanos, es que se ha planteado cambiar su propuesta de valor para con el cliente, con la intención de anticipar todas las necesidades de estos, ya sea con nuevos productos financieros o con nuevos productos digitales que permitan generar una relación fácil y de calidad con los clientes.

Durante el último año, Banco SCL ha destinado gran parte de los esfuerzos y recursos en la adopción de metodologías ágiles como parte del programa de transformación digital. Este programa propone, rediseñar el proceso de construcción de software, que busca como objetivo principal, entregar de forma anticipada los nuevos productos financieros con el propósito de ofrecer el valor de negocio lo antes posible a los clientes y así contribuir a satisfacer las necesidades y expectativas.

Sumado a todo lo anterior, la pandemia originada por el virus COVID-19 se ha encargado de acelerar gran parte de la transformación digital que Banco SCL planificaba para los próximos 3 años. Bajo este nuevo escenario, cobra mayor relevancia generar y entregar nuevas alternativas digitales para que los clientes puedan operar de manera segura.

Dado todos estos antecedentes, es que Banco SCL considera de suma importancia realizar entregas de software confiable en el menor tiempo posible, con calidad y siempre considerando satisfacer a sus clientes y entregar los nuevos productos antes que los bancos competidores.

1.1 Problema del Proyecto

Tal como se describió en el punto anterior, Banco SCL es un banco con un fuerte foco en los clientes y enfocado en satisfacer las necesidades de ellos. En los últimos 10 años, el banco ha hecho una fuerte inversión en la red de sucursales con el objetivo de tener una infraestructura sobresaliente en donde los clientes se sientan a gusto. Por otra parte, durante mucho tiempo se mantuvo la fábrica de software como una empresa externa, separada de banco y dedicada a atender únicamente las necesidades de Banco SCL. Si bien es cierto, la empresa externa estaba dedicada a los requerimientos de Banco SCL, existía una brecha importante entre las necesidades del banco y los intereses de la empresa externa.

La empresa externa tenía su misión apartada de la misión del banco y un régimen en el que se medía la productividad por líneas de código entregadas, esto quiere decir que mientras más líneas de código entregadas, más dinero se cobraba, sin considerar si el software entregado satisfacía las necesidades de los requerimientos de las áreas comerciales del Banco SCL. Este modelo de operación generó una ineficiencia y retrabajo en los proyectos que finalmente terminaban costando tres veces más de presupuestado, dado que el software que se entregaba no cumplía con los requerimientos solicitados por el negocio, los plazos nunca se cumplían y el retraso generaba más presión por parte de las áreas interesadas para que el software llegara a producción. Esta situación de retraso en la entrega y constante presión del cliente implicaba que se construía un software apurado, de baja calidad y con escasa seguridad de que los cambios de software fueran los correctos. Bajo este escenario, cada vez que se requería realizar un cambio de software, se generaban incidentes que afectaban a los clientes. En otro frente, el área de producción del banco se oponía a los cambios de software generados por la empresa externa dado los incidentes productivos que se introducían. Todo este escenario generó áreas antagonistas que rara vez colaboraban para conseguir un objetivo en común, si no que estaban más preocupadas de culparse unas a otras. El desarrollo de software culpaba a la producción de poner trabas y generar retrasos en los proyectos, por otra parte, la producción culpaba al área de desarrollo de generar incidentes productivos que finalmente ellos debían resolver.

Llegada la pandemia y con ella las restricciones de movilidad, hicieron que las personas finalmente dejaran de asistir a las sucursales. Esto generó un grave problema a los directivos del banco ya que su inversión estaba fuertemente enfocada en la red de sucursales y dejó al descubierto una infraestructura tecnológica obsoleta, con quince años de antigüedad y software precario con aplicaciones monolíticas, construidas por distintos subcontratistas a lo largo del tiempo y sin un repositorio de software único, que lo hace muy difícil de modificar.

Sumado a todo lo anteriormente expuesto, en una época en que los servicios de TI son críticos para la operación de cualquier empresa y brindar nuevos productos digitales antes que la competencia es esencial para las pretensiones de un banco que busca ser el mejor de la industria chilena, el marco de trabajo de desarrollo de software utilizado en Banco SCL, agrega más complicaciones puesto que normalmente es un proceso manual, lento y burocrático, en el que realizar una promoción de software implica mover una serie autorizaciones en distintas aplicaciones que están desconectadas de proceso de construcción de software como tal y que por lo demás genera puntos de falla producto de la alta manualidad en cada una de sus etapas.

La motivación del proyecto es apoyar el proceso de transformación digital de banco SCL utilizando un rediseño de procesos de construcción de productos digitales aplicando un modelo de DevOps que permita la automatización en un entorno ágil de construcción de software y que permita mejorar los tiempos de entrega del software.

1.2 Objetivos del Proyecto

1.2.1 Objetivo General

Mejorar el tiempo de entrega de los nuevos productos digitales a los clientes, aplicando un rediseño al proceso de desarrollo de productos digitales, eliminando el modelo de trabajo por proyectos grande e incorporando un entorno de trabajo ágil, un modelo de testing y despliegue continuo con el propósito de disminuir el tiempo que transcurre desde el requerimiento hasta la puesta en producción.

1.2.2 Objetivos Específicos

- Rediseño del proceso de desarrollo de productos digitales, incorporando un entorno de trabajo ágil y un modelo de testing y despliegue continuo.
- Definir y formalizar la incorporación de casos de prueba estructurados con lenguaje natural.
- Diseñar e implementar prácticas de testing unitario y de integración para asegurar el nivel de calidad en las nuevas entregas.
- Implementar pipelines de despliegue automatizado desde etapas de desarrollo hasta despliegue a producción.
- Generar una base de información para crear un modelo predictivo que permita la detección temprana de fallas en producción.

1.2.3 Resultados esperados

- Diminución en un 40% los tiempos de entrega de software, desde que se comienza a desarrollar hasta que se entrega a la producción para ser explotado.
- Disminuir en un 40% los tiempos de construcción de pruebas automatizadas al escribir el diseño del software (Historia de usuario) en lenguaje natural Gherkin.
- Aumentar la calidad del software disminuyendo la tasa de errores, producto de mayor cobertura en las certificaciones. Se espera reducir en un 80% los tiempos de certificación de calidad de código de software.
- Aumentar en 70% la velocidad y en 90% calidad de los despliegues de software a producción.
- Una base sistematizada de datos para el desarrollo de un modelo predictivo.

1.3 Justificación y Relevancia del Proyecto

1.3.1 Justificación

En el año 2020 las instituciones bancarias y financieras avanzan fuertemente con el objetivo de digitalizar la mayor cantidad de productos, es así que existen una variedad de ofertas tales como: tarjetas de crédito de prepago, tarjetas digitales, apertura de cuentas y curse de créditos 100% on-line.

La situación expuesta anteriormente da cuenta de un gran problema para el Banco, dado que, por una parte, la industria altamente competitiva obliga a hacer muchos despliegues de nuevo software a la producción y por otra parte un contexto poco amigable para la realización de despliegue de nuevo software a la producción del Banco. Esto se explica, entre otras cosas, por el retrabajo que se genera debido a la manualidad de estos procesos y los distintos puntos de falla que esto implica, por ejemplo, la alta tasa de defectos que son detectados en ambiente Pre-Productivo.

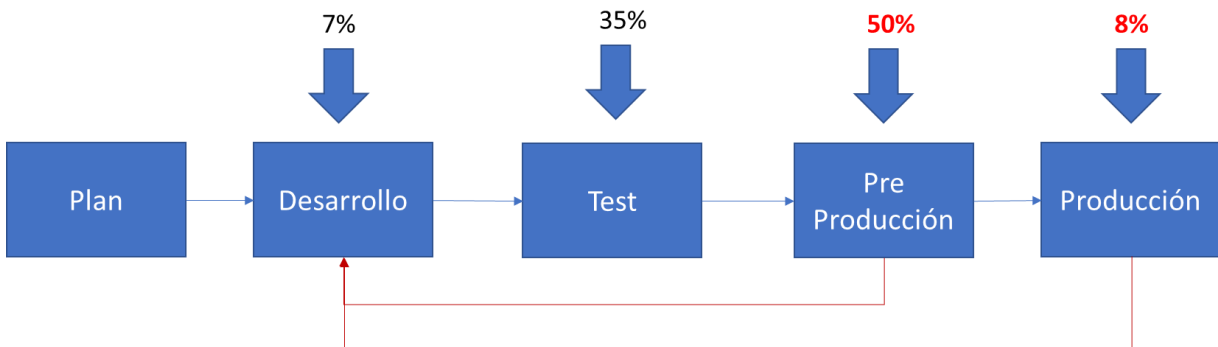


Figura 1. Ambientes y defectos detectados
Fuente: *(Elaboración propia)*

Llegar con un software a un ambiente Pre-Productivo es prácticamente casi tan costoso como llegar a producción y detectar defectos del software en esta instancia genera un alto retrabajo dado que el software debe volver a las instancias de desarrollo y luego volver a pasar por el entorno de Test y Pre-producción. Lo que finalmente genera retrasos en las fechas y costos adicionales al software.

El presente proyecto de grado busca mejorar el tiempo de entrega de los nuevos productos digitales para así anticiparse a los bancos competidores y no perder cuota de mercado frente a ellos, para conseguir este objetivo, se realizará un rediseño del proceso de desarrollo de productos digitales. Este trabajo consiste en identificar, medir y priorizar las etapas del proceso que son más repetitivas, así como también, las etapas que involucran una mayor cantidad de tiempo, con la finalidad de automatizar dichas etapas y eliminar desperdicios. Todo lo anterior busca convertir al Banco en una organización ágil con la finalidad de satisfacer a los clientes antes, mejor y con mayor calidad; y al mismo tiempo generar valor al Banco.

1.3.2 Relevancia

La relevancia de realizar este trabajo viene dada por cuatro puntos que se han descrito anteriormente:

- Banco SCL espera ser el mejor banco para sus clientes y ser el numero uno a nivel nacional.
- Se encuentra en una zona de alto riesgo dado por la alta competencia del mercado financiero.
- La fuerte tendencia, debido a la pandemia, de evolucionar las sucursales físicas a entregar productos digitales para que los clientes puedan operar desde sus dependencias sin necesidad de asistir a una sucursal.
- Mejorar el tiempo de entrega de productos digitales disminuyendo el retrabajo en el proceso de construcción de software.

1.4 Solución Propuesta

Este trabajo de tesis propone el rediseño de proceso incorporando arquitecturas de multinivel y la adopción de las prácticas DevOps en la organización, lo que se explicará en mayor profundidad en el siguiente capítulo.

CAPÍTULO 2: MARCO TEÓRICO

El presente capítulo describe el marco teórico utilizado para desarrollar este trabajo de tesis el cual se basa en la metodología de ingeniería de negocios del Magister en Ingeniería de Negocios con Tecnologías de la Información.

2.1 Metodología de Ingeniería de Negocios

La ingeniería de negocios entrega distintas herramientas para lograr descomponer y posteriormente obtener el diseño de las organizaciones y los servicios que éstas entregan (Barros, 2009). Generalmente las empresas poseen un diseño limitado en donde se puede encontrar la estructura, procesos, roles y sus funciones que poco representan a la realidad que se vive en el día a día de la organización.

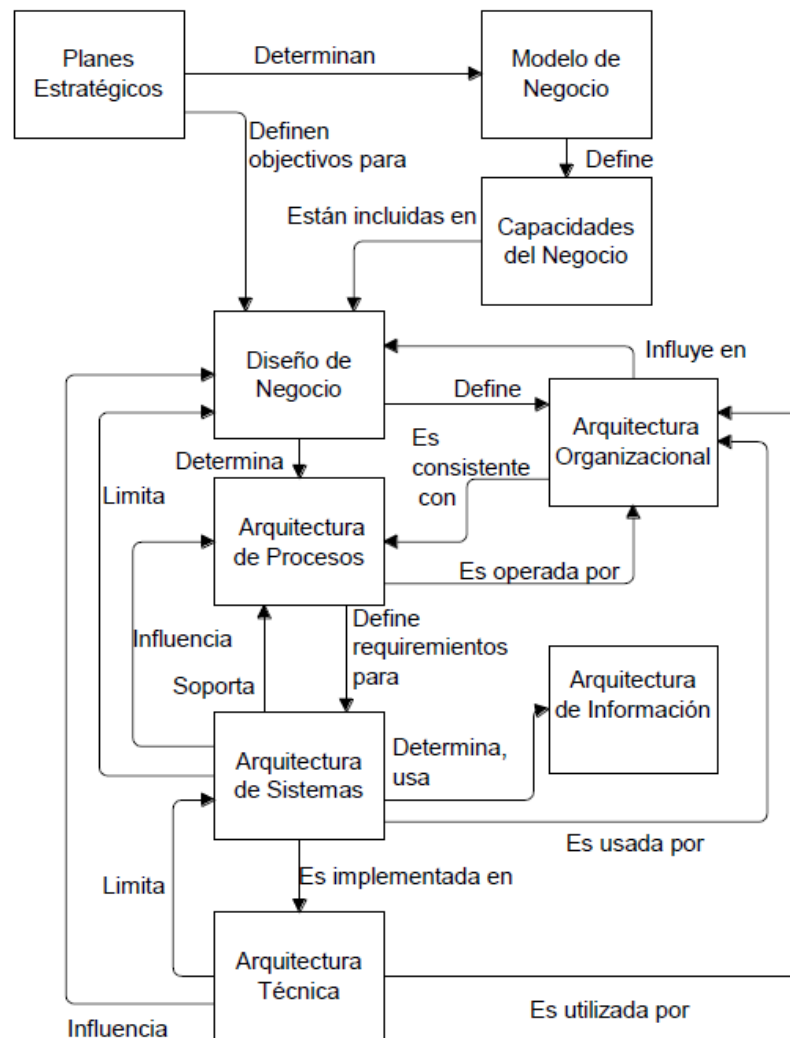


Figura 2. Ontología de Ingeniería de negocios
Fuente: (Barros, 2015)

Basado en la metodología de ingeniería de negocios y la arquitectura empresarial que se presenta en libro “Modelación Y Diseño De Una Arquitectura Empresarial (AE) Multinivel Compleja” (Barros, 2017) que considera escenarios en los que se puede apreciar una arquitectura que presenta varios niveles de servicios para una misma organización, como por ejemplo, una institución bancaria, en ella es posible encontrar servicios particulares orientado a los clientes específicos, que pueden ser: personas, personas de alto patrimonio, inversionistas, empresas de distintos tamaños e incluso grandes conglomerados. Además de estos servicios, se encuentran otros procesos de gestión especializados tales como Marketing, Gestión comercial, Sistemas y Tecnologías de información entre otros.

Esta metodología tiene como objetivo proporcionar herramientas para el diseño de arquitecturas complejas permitiendo modularizar dichos servicios, con el objetivo de separar en subestructuras que colaboran entre ellas. De esta manera es posible generar economía de escala derivada la estandarización de los servicios compartidos lo permite mejorar eficiencia.

2.2 Arquitectura de Procesos de Negocios

La arquitectura de procesos establece métodos para implementar las capacidades y el diseño de una organización. Esta arquitectura especifica las dependencias, las relaciones de los procesos y su conexión entre sí.

Para diseñar esta arquitectura se utilizan patrones que se basan en extensiva experiencia de diseño de procesos realizada en cientos de casos reales y comparten la idea que existen cuatro agrupaciones de procesos, llamados macroprocesos, que existen en cualquier organización, los cuales se detallan a continuación:

- **Cadena de Valor (Macro 1):** Conjunto de procesos que ejecutan la producción de bienes y/o servicios de la empresa u institución, el cual va desde que se interactúa con el cliente o usuario para generar requerimientos, hasta que estos han sido satisfechos.
- **Diseño de Nuevas Capacidades (Macro 2):** Conjunto de procesos que desarrollan las nuevas capacidades que la empresa requiere para ser competitiva: los nuevos productos y servicios, incluyendo modelos de negocios, que una empresa requiere para mantenerse vigente en el mercado. Este macroproceso se centra en la capacidad de innovación de la empresa.
- **Planificación del Negocio (Macro 3):** Conjunto de procesos que planifican el negocio, es decir, que definen el curso futuro de la organización en la forma de estrategias, que se materializan en planes y programas.

- **Recursos de Apoyo (Macro 4):** Conjunto de procesos de apoyo que manejan los recursos necesarios para que los anteriores operen. Existen cuatro grandes grupos: recursos financieros, humanos, infraestructura y materiales.

Las relaciones y dependencias de los macroprocesos mencionados se representan en la siguiente figura 3:

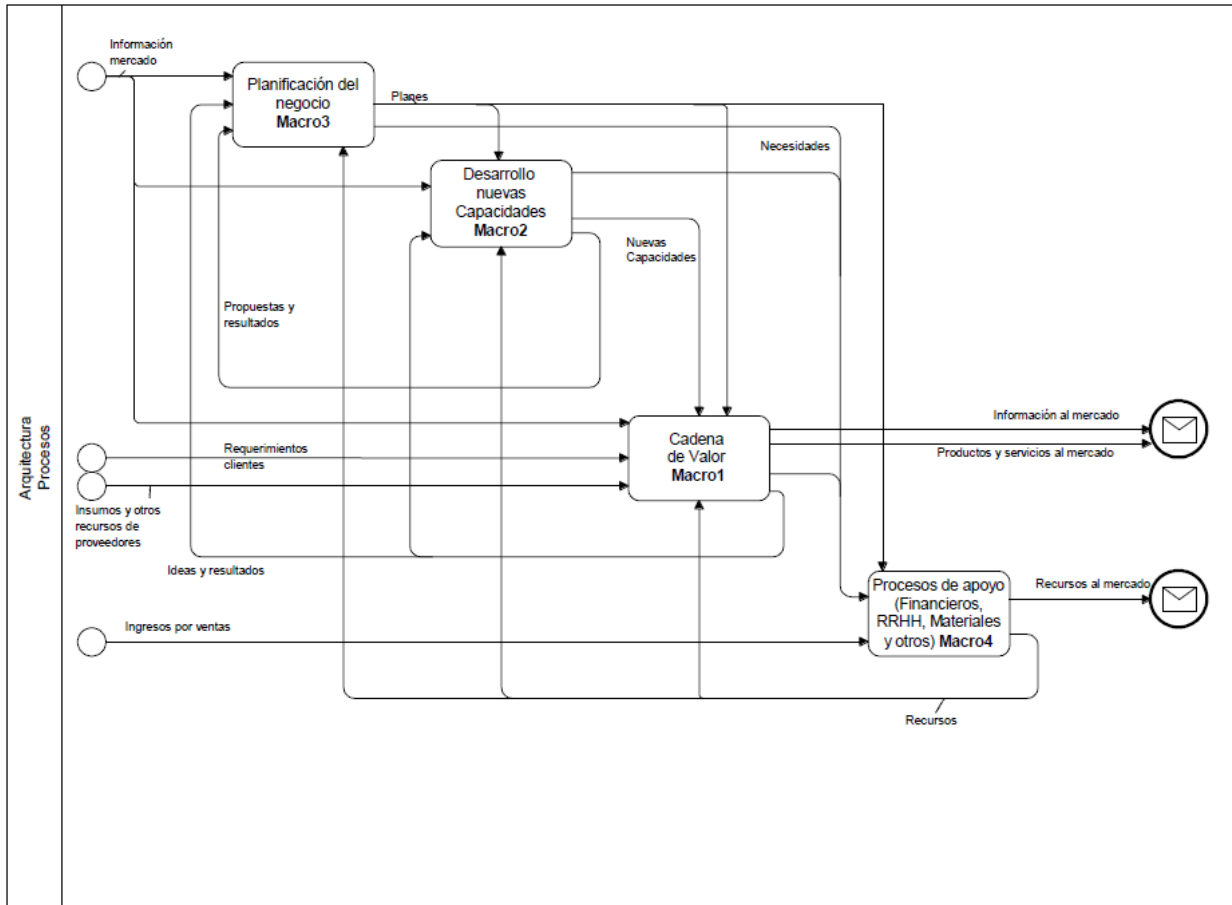


Figura 3. Arquitecturas de macroprocesos
Fuente: (Barros, 2017)

Las organizaciones pueden modelarse en base a estos cuatro macroprocesos, dicho modelo entrega una estructura integrada que permite identificar las relaciones entre los procesos, los flujos de información y los requerimientos entre ellos, que permiten una mejor gestión de la organización en su conjunto.

En la *Figura 4: Tipos de Arquitectura*, se presentan las cuatro alternativas principales definidas en:

- **Una cadena de valor** del tipo macro 1.
- **Varias cadenas de valor:** cada una de las cuales opera independientemente, esta se denomina Diversificación.

- **Varias cadenas de valor**, las que operan de manera independiente, pero comparten ciertos servicios centrales, se denomina Coordinación y Replicación.
- **Varias cadenas de valor**, que comparten varios de sus servicios internos y que también comparten servicios centrales, recibe el nombre de Unificación.

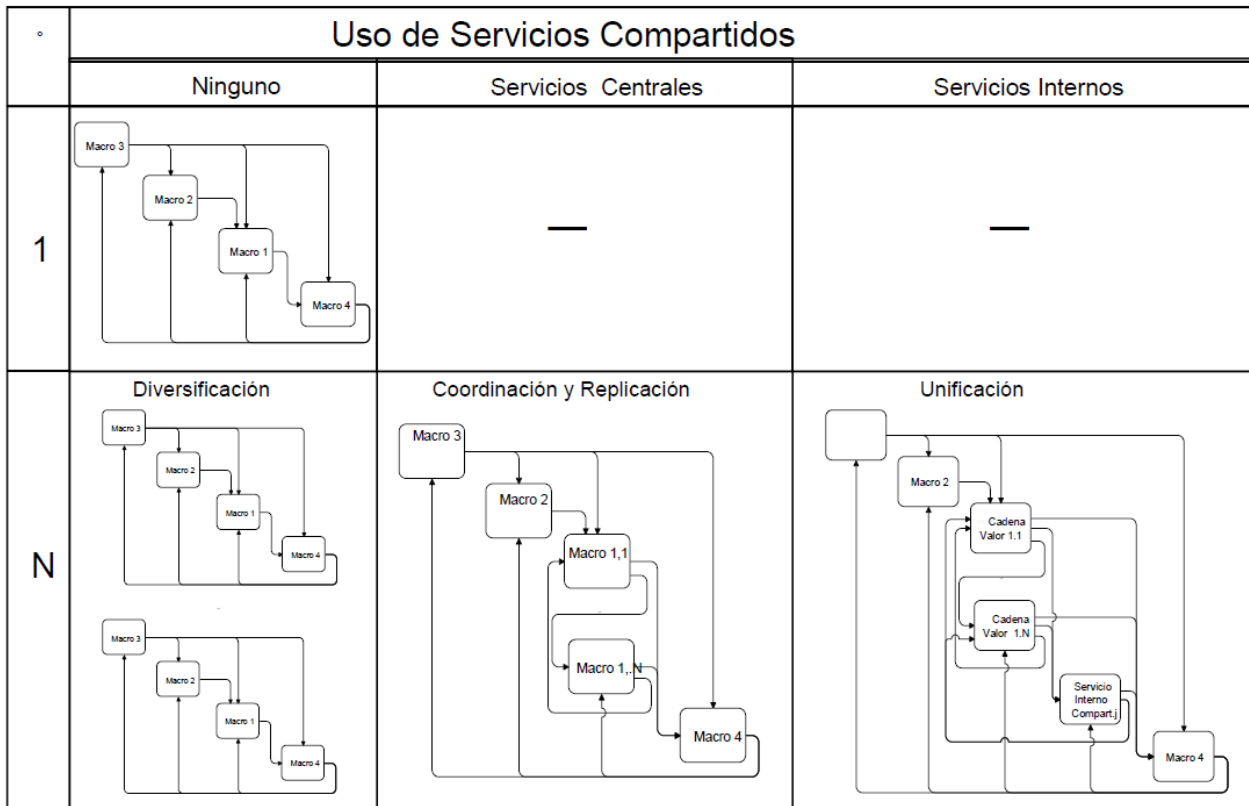


Figura 4. Tipos de Arquitectura
Fuente: (Barros, 2017)

2.3 Revisión de Literatura Relevante

2.3.1 Metodología ágil

La agilidad es la capacidad responder de manera rápida ante un evento inesperado, bajo esta premisa resulta más fácil entender la metodología ágil que tiene como por objetivo principal centrarse en las actividades más relevantes de la construcción del software, las que generan más valor para conseguir el resultado final y evitar las actividades burocráticas que no aportan un valor significativo al objetivo.

Tal como se declara en Manifiesto ágil (Agile Manifesto, 2001), se valoran más:

- **Las personas y sus interacciones** por sobre los procesos y las herramientas
- **El software funcionando** por sobre la documentación extensiva
- **Colaboración con el cliente** por sobre negociación contractual
- **Respuesta ante los cambios** por sobre seguir el plan

Vale la pena destacar que, de estos cuatro valores, el manifiesto ágil expresa que, aunque se aprecia la importancia de los elementos de la derecha, se valoran mucho más los elementos de la izquierda.

La metodología ágil es un término genérico para un conjunto de marcos de trabajo y buenas prácticas que se basan en los valores ya vistos, adicionalmente esta metodología se sustenta en los doce principios de la agilidad (Twelve Principles of Agile, 2001) que se detallan a continuación:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con referencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Dentro de las metodologías ágiles que utilizadas se encuentran Scrum y Kanban, éstas dos serán explicadas en detalle en los próximos puntos.

2.3.2 Modelo Tradicional Vs Ágil

En el modelo tradicional de construcción de software, el área de negocio entrega las especificaciones del requerimiento y el jefe de proyectos responsable del equipo de desarrollo de software, gestiona la construcción del software hasta entregarlo a producción. Normalmente este proceso podría llevar varios meses, incluso, en casos extremos, podrían pasar años hasta que la solución sea liberada a producción.

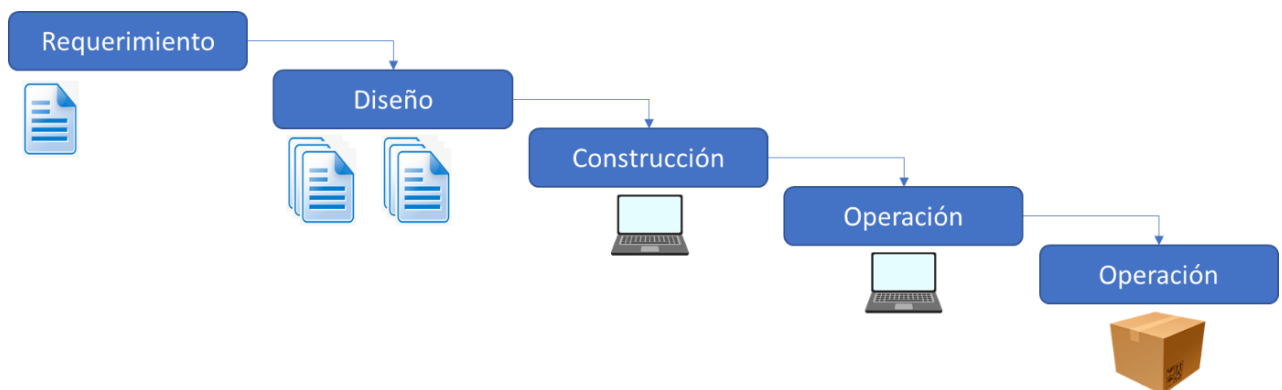


Figura 5. Diagrama de modelo tradicional. Entrega de valor al final.
Fuente: (Elaboración propia)

Por otra parte, la metodología ágil propone un modelo de entrega continua, tal como se explica en los doce principios de la agilidad visto anteriormente. Bajo este modelo, cada iteración es una entrega de valor constante e iterativo, Bajo este modelo es posible aprender, corregir y adaptarse rápidamente con cada una de las entregas liberadas a los clientes, lo que en teoría conlleva a soluciones de negocio más exitosas en un mercado competitivo.

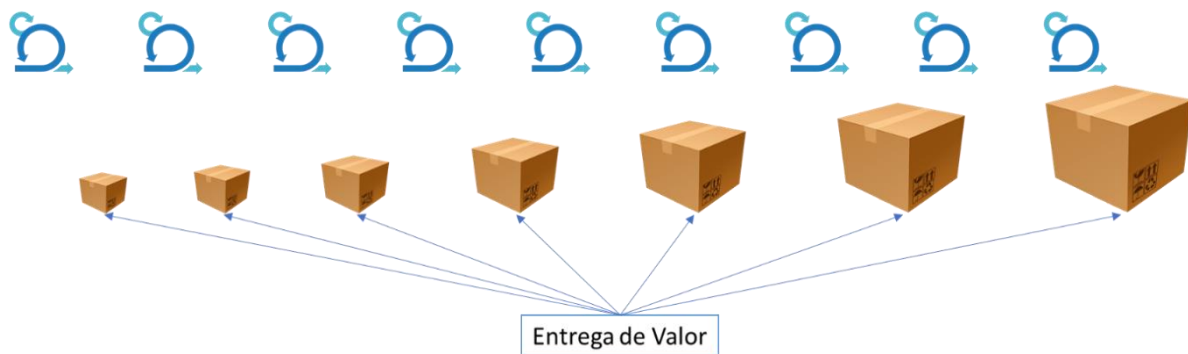


Figura 6. Diagrama de modelo ágil. Entrega de valor constante.
Fuente: (Elaboración propia)

Según describe el PMI (Fair, 2012), las principales ventajas de las metodologías ágiles, es que la mayoría se basan en principios LEAN y buscan eliminar del proceso las actividades que tienen un bajo aporte de valor, ya que son catalogadas como un desperdicio del proceso. Adicionalmente PMI describe ciertas diferencias que resultan ser clave entre la construcción de proyectos en cascada tradicional y el enfoque de producto que proponen las metodologías ágiles:

Tabla 1: Ágil Vs Cascada

Cascada	Ágil
Planes detallados a largo plazo con un cronograma único	Planificación más corta basada en iteraciones y entregas múltiples
Roles de equipo definitivos y rígidos	Equipo flexible y multifuncional
Cambios en los entregables son costosos	Se esperan cambios en los entregables y son menos impactantes
Producto terminado entregado al final	Producto entregado en etapas funcionales
El cliente suele participar solo al principio y al final de un proyecto	El cliente participa durante todo el sprint
El enfoque de fases crea dependencias	El enfoque reduce dependencias

2.3.3 Scrum

Scrum (Schwaber & Sutherland, 2017) es uno de los marcos de trabajo ágil más populares ya que centra la atención y el trabajo en el producto para los clientes, en este marco de trabajo resalta la colaboración efectiva que adoptan la personas que forman parte de dicho equipo. El Scrum consta de una serie de reglas, roles, artefactos y ceremonias que serán detalladas a continuación:

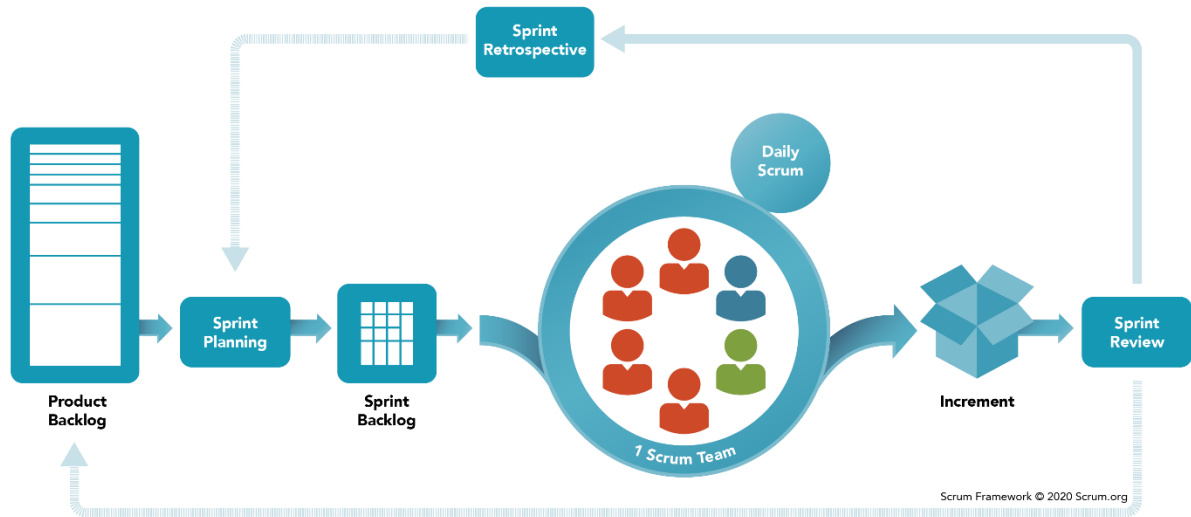


Figura 7. Diagrama del Proceso Scrum
Fuente: (Scrum guide, 2017)

▪ Roles

- **Scrum Master:** Es el facilitador del equipo, el encargado de hacer que las cosas ocurran fomentando la mentalidad y prácticas ágiles tanto dentro como fuera del equipo.
- **Product Owner:** Es el encargado de transmitir las necesidades del cliente y del negocio en el equipo. Define y prioriza las iniciativas que estas estén alineadas con el negocio.
- **Development Team:** Es un equipo multidisciplinario encargado de construir el software de calidad trabajando colaborativamente para realizar las entregas constantes y poder satisfacer las necesidades de los clientes.

- Ceremonias

- Sprint: Es la ceremonia principal del scrum, generalmente tiene un periodo de duración de dos o cuatro semanas, en este tiempo definido y constante se construye el incremento de valor. Un nuevo Sprint comenzará inmediatamente después de terminar el Sprint anterior.
- Sprint Planning: En esta ceremonia se planifica el trabajo que será realizado en el Sprint. Este plan es creado manera colaborativa por todo el equipo que forma parte de la célula scrum. El responsable de esta actividad es el Scrum Master.
- Daily Scrum: Es un evento diario con una duración ideal de 15 minutos y busca principalmente comunicar lo que se hizo el día anterior y lo que se busca terminar el día actual, lo que genera mayor colaboración y aumento del rendimiento del equipo.
- Sprint Review: Esta actividad se realiza al término del Sprint, en ella se efectúa una revisión del incremento realizado durante el periodo de tiempo definido y se valida si se cumplió el objetivo del Sprint o si quedó algo pendiente.
- Sprint Retrospective: Esta ceremonia genera la posibilidad para que el equipo se inspeccione a sí mismo, esto genera aprendizaje continuo y un plan de mejora para que el próximo Sprint se más efectivo.

- Artefactos

- Product Backlog: Es la lista ordenada de todos los requerimientos que se deben hacer para el producto. El Product Owner es el responsable de este artefacto. Es la única fuente de ingreso de requerimientos para el equipo y en esta lista se enumeran todas las características, funciones, requisitos, mejoras y correcciones que se deben construir para el producto.
- Sprint Backlog: es el conjunto de elementos del Product Backlog que han sido seleccionados para construir durante el sprint y así generar el incremento de valor al producto.
- Increment: El Incremento es el trabajo realizado durante Sprint que genera un aumento de valor para el producto.

2.3.4 Kanban

Kanban es un método de gestión de trabajo visual que se basa en el uso de un tablero con columnas que representan las distintas etapas del proceso y tarjetas que representan un elemento de trabajo. La tarjeta o elemento de trabajo se desplaza por las distintas columnas hasta su finalización, de esta manera es posible gestionar el flujo de trabajo de manera visual y continua, ya que uno de los principios del Kanban es reducir los desperdicios sin afectar la productividad.

Cuando se utiliza como metodología ágil para el desarrollo de software, el tablero Kanban se basa en las fases del ciclo de vida del desarrollo de software para representar las diferentes etapas del proceso. El objetivo es controlar y gestionar el flujo de los distintos desarrollos para que el número de entregables que entran en el proceso coincida con las que se están completando.

A diferencia de Scrum, Kanban es una metodología ágil que no es necesariamente iterativa. Procesos como Scrum tienen iteraciones cortas que imitan el ciclo de vida de un proyecto a pequeña escala, teniendo un comienzo y un final distintos para cada iteración. Kanban permite que el software se desarrolle en un solo ciclo de desarrollo. A pesar de esto, Kanban es un ejemplo de una metodología ágil porque cumple con los doce principios detrás del manifiesto ágil, porque, aunque no es iterativo, es incremental.

El principio detrás del Kanban, que permite que sea incremental y ágil, es que tiene un rendimiento limitado. Sin iteraciones, un proyecto Kanban no tiene puntos de inicio o final definidos, cada uno puede empezar y terminar independientemente del otro, y los elementos de trabajo no tienen una duración predeterminada. En cambio, se reconoce que cada fase del ciclo de vida tiene una capacidad limitada de trabajo en un momento dado. Se crea un pequeño elemento de trabajo a partir de la lista de requisitos priorizados y no iniciados y luego se inicia el proceso de desarrollo, generalmente con la elaboración de algunos requisitos. No se permite que un elemento de trabajo pase a la siguiente fase hasta que se abra una cierta capacidad. Al controlar el número de tareas activas en un momento dado, los desarrolladores todavía se acercan al proyecto global de forma incremental, lo que les da la oportunidad de aplicar principios ágiles.

Los proyectos Kanban tienen límites llamados Work In Progress (WIP), que son la medida de la capacidad que mantiene al equipo de desarrollo enfocado en una pequeña cantidad de trabajo a la vez. A medida que se completan las tareas, las nuevas tareas se incorporan al ciclo. Los límites del WIP deben ser ajustados en base a comparaciones del esfuerzo esperado versus el esfuerzo real para las tareas que se completan.

Kanban no impone ninguna definición de rol y junto con la ausencia de iteraciones formales, la flexibilidad de roles hace que Kanban sea atractivo para aquellos que han

estado usando modelos de desarrollo tradicionales en cascada y que quieren cambiar, pero que temen la conmoción inicial que algo como Scrum puede causar mientras son adoptados por un equipo de desarrollo.

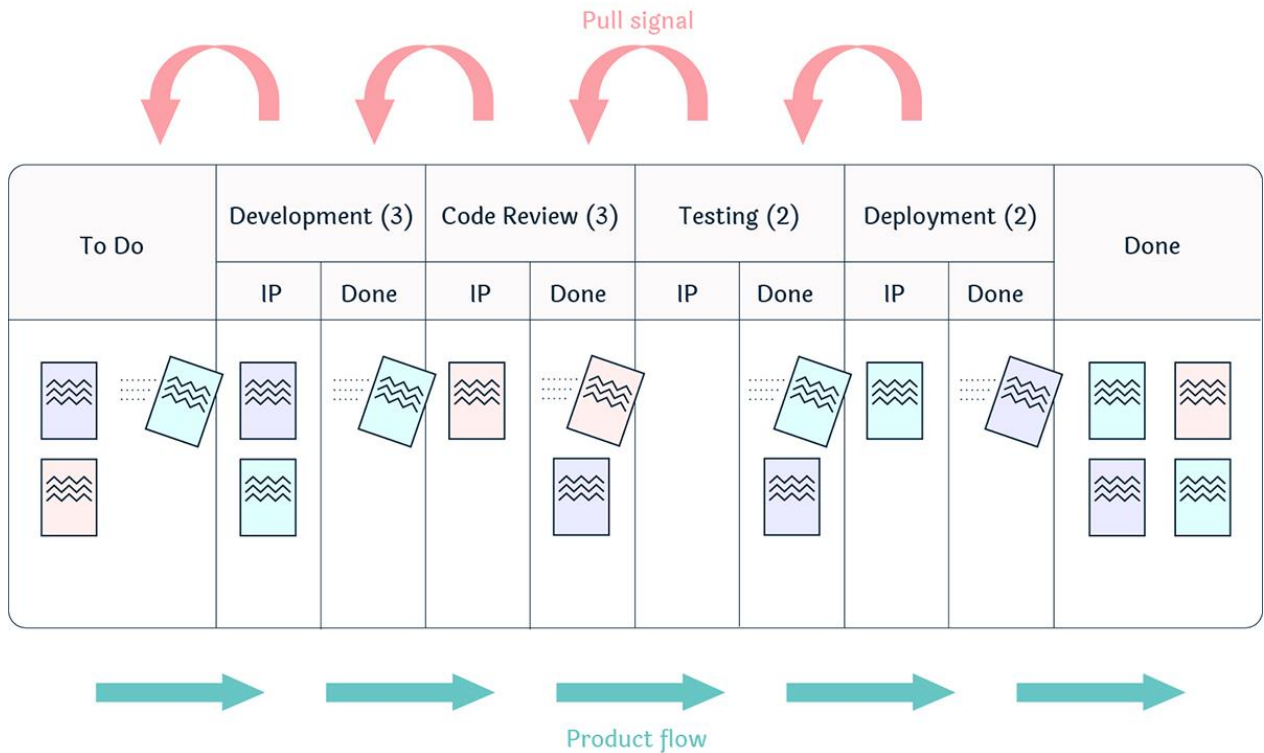


Figura 8. Diagrama del Proceso Kanban
Fuente: (Agile Alliance, s.f.)

2.3.5 SAFe

Es un marco de trabajo que permite generar ventajas en el proceso de desarrollo de software, incluye patrones y flujos que sirven para implementar la agilidad escalada dentro de una organización. Está diseñado para ayudar a las empresas a impulsar el compromiso de los trabajadores, mejorar el incremento en la productividad y la calidad. Lo anterior permite que a las empresas que lo adopten sean capaces de entregar valor a sus clientes de manera continua y en forma eficiente. Se basa principalmente en tres pilares: el desarrollo ágil de software, el desarrollo Lean y el pensamiento sistemático.

Conforme el tamaño de una organización, la versión 5 de SAFe propone una estrategia distinta para escalar la agilidad, en este contexto existen cuatro modelos de SAFe que se acomodan a las distintas estructuras las cuales son: Essential SAFe, Large Solution SAFe, Portfolio SAFe y Full SAFe.

El marco de trabajo SAFe se basa en planes de educación para los distintos roles que componen la agilidad escalada. Estas capacitaciones contemplan a todos los niveles de una empresa, tanto a los líderes como a los colaboradores, ya sean de equipos comerciales o técnicos. De esta manera logra que, en conjunto, las distintas áreas trabajen y desarrollen las prácticas ágiles y LEAN dentro de la organización.

SAFe reconoce la necesidad empresarial de transformar las organizaciones en organizaciones ágiles para que estas puedan responder de mejor manera a la transformación digital. En este escenario aparece el concepto de DevOps que fomenta la automatización de los procesos y desempeña una función clave para adoptar la agilidad empresarial debido a que DevOps es un habilitador para realizar liberaciones de producto de manera constante logrando superar los desajustes tradicionales en los flujos de valor.

Los beneficios de implementar este marco de trabajo se pueden ver reflejados en los casos de estudio de empresas que han implementado SAFe. Dentro de las principales cuatro mejoras, es interesante destacar la mejora del Tiempo de Entrega con un incremento del 50% y el aumento de la Calidad en un 50% (SAFe, 2020).

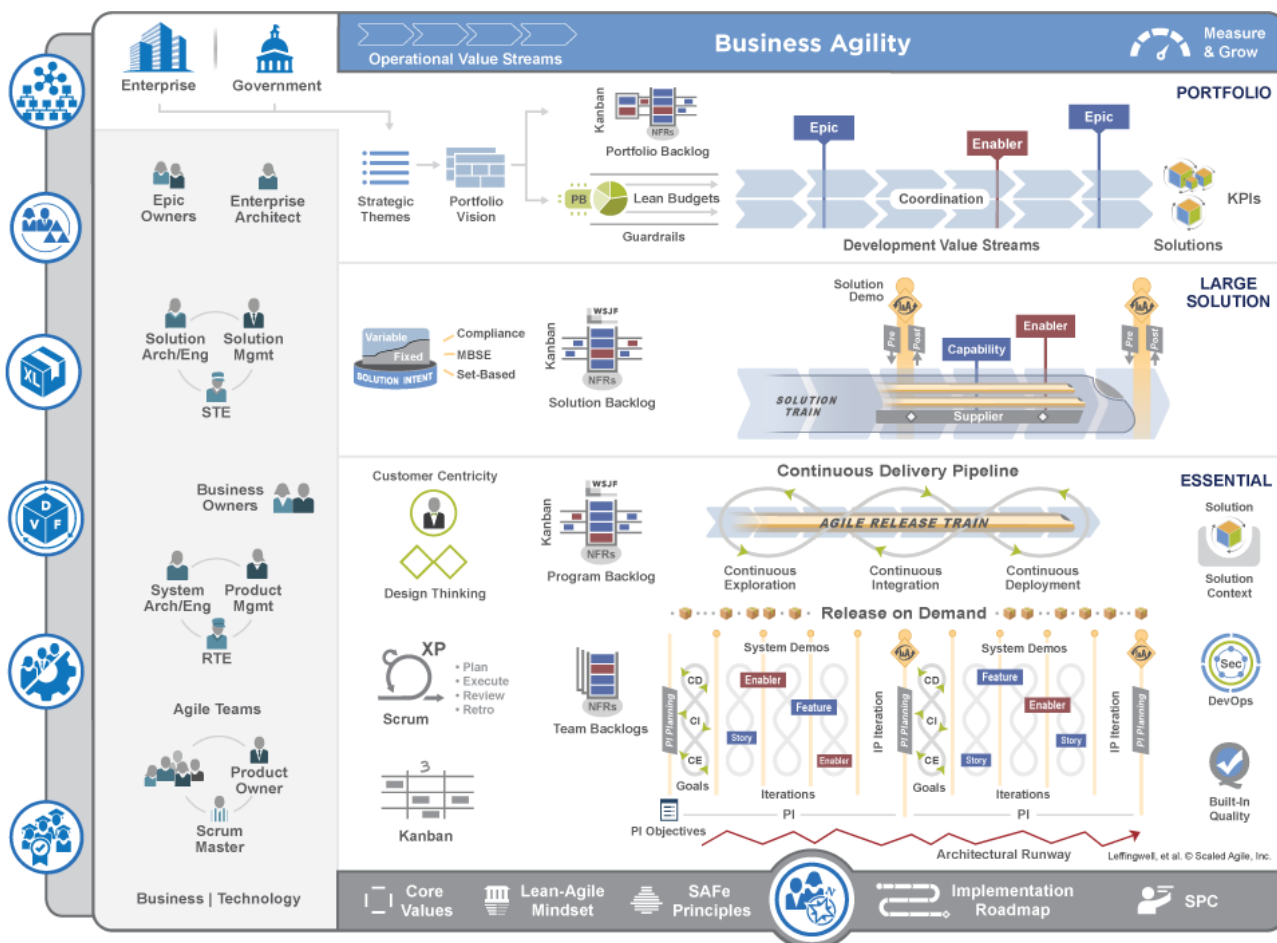


Figura 9. Beneficios de implementar SAFe
Fuente: (SAFe, 2020)

Las empresas que ha adoptado SAFe también han adoptado DevOps para romper los silos y potenciar las entregas constantes y de tal manera ofrecer continuamente nuevas funciones a sus usuarios finales.

En la siguiente figura 10 se explica visualmente el concepto de SAFe, en ella se puede apreciar la colaboración que existe entre los roles en todos los niveles. En los niveles

superiores se puede destacar la presencia de Epic Owner y Business Owner, además del ya mencionado Product Owner



Lean-Agile Leadership

Figura 10: Diagrama SAFe
Fuente: (SAFe Scaled Agile, s.f.)

2.3.6 DevOps

DevOps es la combinación de filosofías, prácticas y herramientas culturales que aumenta la capacidad de una organización para entregar productos y servicios a alta velocidad.

En un entorno DevOps, los distintos equipos de construcción de software y los de operaciones logran mantener un alto grado de fusión, incluso en algunos casos llegan a ser un solo equipo, en donde los especialistas de las distintas áreas trabajan en todo el ciclo de vida del producto, desde el desarrollo del software con sus respectivas pruebas hasta la implementación y la operación.

La cultura es el núcleo de las prácticas DevOps, la adopción de esta cultura instauro principios, prácticas, herramientas y métricas en gran parte de la organización y de este

modo de logran eliminar los silos entre las distintas áreas de desarrollo (Dev) y las áreas de operaciones (Ops). Algunas empresas han adoptado modelos en los que además se incorporan los equipos de aseguramiento de calidad y de seguridad a los equipos de desarrollo y operaciones.

- Principios: Son los conceptos y pautas que emergen naturalmente de la cultura DevOps por ejemplo, responsabilidad de extremo a extremo, comenzar a construir con el cliente en mente.
- Prácticas: actividades o hábitos técnicos concretos que nos ayudan a implementar los principios de DevOps por ejemplo, integración continua
- Herramientas: Aplicaciones utilizadas para las prácticas y métodos de DevOps.
- Métrica: Habilitadores de mejora. "Si no puedes medirlo, no puedes mejorarlo" (Peter Drucker). El monitoreo frecuente permite mejoras continuas.

Eliminando los silos entre los equipos, el desarrollo y las operaciones colaboran estrechamente y comparten responsabilidades. La cultura DevOps desdibuja la línea divisoria entre ambos y la automatización se convierte en un lenguaje común entre los equipos. Dichos quipos son autónomos para colaborar eficazmente, los desarrolladores y el personal de operaciones deben ser capaces de tomar decisiones y aplicar cambios sin complejos procesos de toma de decisiones. Esto implica confiar en los equipos, cambiar la forma en que se gestiona el riesgo y crear un ambiente de trabajo colaborativo como se representa en las figuras 11 y 12.

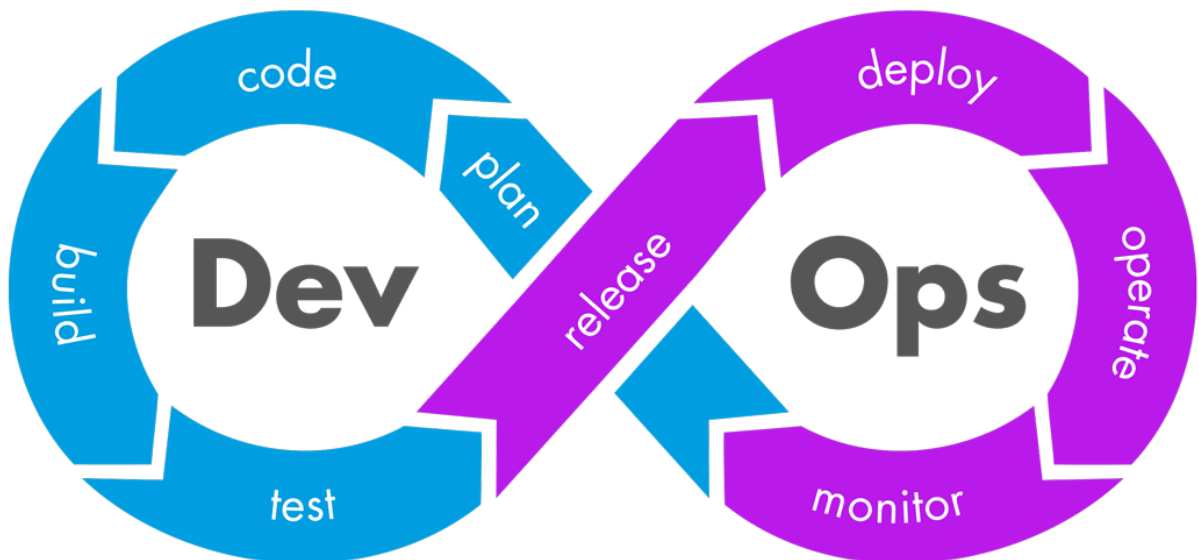


Figura 11. Diagrama de DevOps
Fuente: (*Best Practice DevOps, s.f.*)

La responsabilidad compartida, es la actitud que fomenta una estrecha colaboración entre los roles de desarrollo y operaciones, por otro lado, para completar este enfoque, los equipos de desarrollo aceptan el desafío y asumen la responsabilidad.

La piedra angular de DevOps es la automatización, se espera poder automatizar la mayor cantidad de etapas como las actividades de prueba, despliegues, entre otras. De esta manera se libera a los ingenieros para que se enfoquen en agregar valor y reducen la posibilidad de errores humanos.

Una mentalidad es la percepción que las personas tienen sobre cosas como ellas y sus habilidades. A diferencia de los negocios tradicionales, los digitales tienen una naturaleza cambiante continua e inmediata, que exige una mentalidad que promueve la adaptación. Se ha definido como "mentalidad de crecimiento" y tiene, como características principales:

- Acepta los desafíos.
- Persistir ante los contratiempos.
- El esfuerzo conduce al dominio.
- Aprender de la crítica.
- Encuentra lecciones e inspiración en el éxito de los demás

DevOps proporciona el soporte tecnológico para permitir el desarrollo de software ágil. Aplica la entrega de software iterativa e incremental, que es el enfoque ágil que permite a las empresas avanzar más rápido hacia las necesidades cambiantes del mercado.

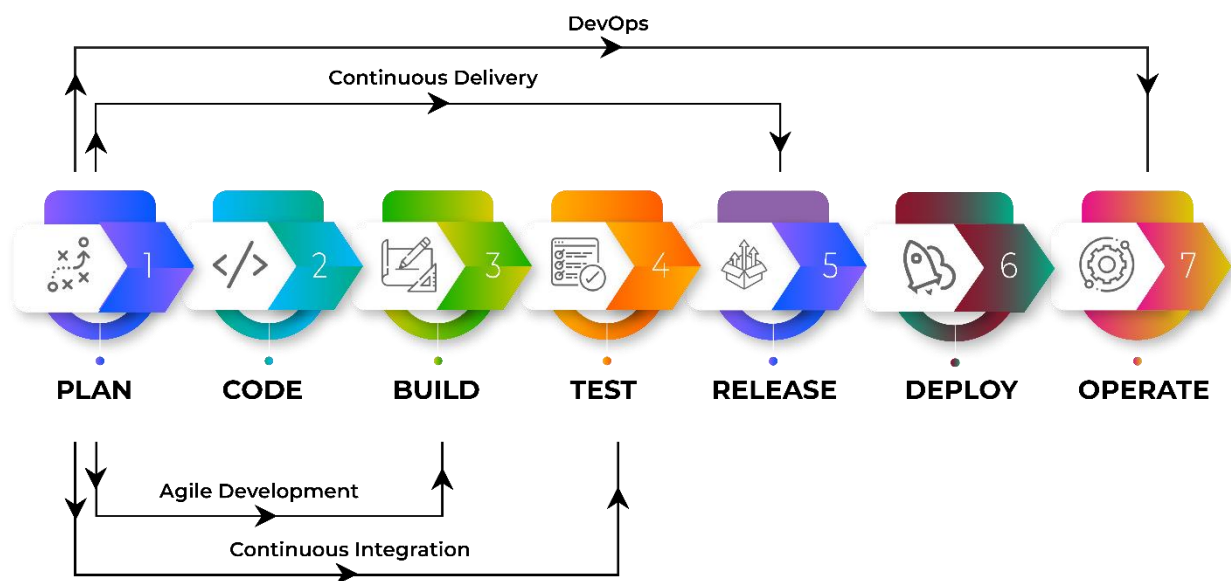


Figura 12. Etapas de DevOps
Fuente: (Elaboración propia)

La entrega continua es una práctica de DevOps que refleja claramente esta simbiosis. Permite el desarrollo de software ágil al reducir el costo de cada iteración y aumentar la calidad. También requiere una incorporación temprana de Operaciones al proceso de desarrollo, ya que el primer incremento del producto se integra y se entrega como candidato de lanzamiento.

2.3.7 Prácticas de DevOps

Las siguientes son prácticas necesarias y progresivas para implementar un modelo de DevOps:

- **Integración continua:**

La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software.

- **Entrega continua:**

La entrega continua es una práctica de desarrollo de software mediante la cual se compilan, prueban y preparan automáticamente los cambios en el código y se entregan a la fase de producción. Amplía la integración continua al implementar todos los cambios en el código en un entorno de pruebas o de producción después de la fase de creación. Cuando se la entrega continua se implementa de manera adecuada, los desarrolladores dispondrán siempre de un artefacto listo para su implementación que se ha sometido a un proceso de pruebas estandarizado.

- **Microservicios:**

La arquitectura de microservicios es un enfoque de diseño que sirve para crear una sola aplicación como conjunto de servicios pequeños. Cada servicio se ejecuta en su propio proceso y se comunica con otros servicios mediante una interfaz bien definida utilizando un mecanismo ligero, normalmente una interfaz de programación de aplicaciones basada en HTTP (API). Los microservicios se crean en torno a las capacidades empresariales. Cada servicio abarca un único propósito. Puede utilizar distintos marcos o lenguajes de programación para escribir microservicios e implementarlos independientemente, como servicio único, o como grupo de servicios.

- **Infraestructura como código:**

La infraestructura como código es una práctica mediante la que se aprovisiona y administra infraestructura con técnicas de desarrollo de código y de software, como

el control de versiones y la integración continua. El modelo orientado a la API de la nube permite a los desarrolladores y administradores de sistemas interactuar con la infraestructura mediante programación y a escala, en lugar de configurar y ajustar recursos manualmente. Así, los ingenieros pueden interactuar con la infraestructura con herramientas basadas en código y tratar la infraestructura de un modo parecido a como tratan el código de la aplicación. Como están definidos por el código, la infraestructura y los servidores se pueden implementar con rapidez con patrones estandarizados, actualizar con las últimas revisiones y versiones o duplicar de forma repetible.

- **Comunicación y colaboración:**

El incremento en la comunicación y la colaboración en una organización es uno de los aspectos culturales clave de DevOps. El uso de las herramientas de DevOps y la automatización del proceso de entrega de software establece la colaboración al reunir físicamente los flujos de trabajo y las responsabilidades de los equipos de desarrollo y operaciones. Además, estos equipos establecen normas culturales sólidas que giran en torno a compartir información y facilitar la comunicación mediante el uso de aplicaciones de chat, sistemas de seguimiento de proyectos o problemas y wikis. De este modo, se acelera la comunicación entre los equipos de desarrollo y operaciones e incluso con otros equipos, como marketing y ventas, lo que permite que todos los departamentos de la organización se alineen mejor con los objetivos y proyectos.

- **Testing Continuo:**

Las pruebas funcionales se distribuyen en diferentes niveles de la aplicación y se ejecutan durante todo el ciclo de vida de la aplicación: desde que se generan las primeras líneas de código (pruebas unitarias) hasta el despliegue de producción (E2E, pruebas de humo, regresión, etc). La economía es clave para evitar el exceso de trabajo, es por eso por lo que se creó la pirámide de pruebas, para explicar conceptualmente cuál es el enfoque de prueba de más eficiente. Las pruebas en la base son baratas de generar y ejecutar, además el encontrar defectos en estas pruebas es más eficiente puesto que aún se encuentra en etapa de construcción. Las capas superiores incrementan gradualmente la complejidad y la ineficiencia a la hora corregir los defectos. Los conjuntos de pruebas en la base deben ser completos y las capas superiores tan escasas como sea posible.

- **Pruebas unitarias:**

Valida que una unidad de código (método único, clase o función) aislada funcione de la forma esperada. Las pruebas unitarias son la base de la estrategia de prueba y deben cubrir todas las posibles rutas del software.

Estas pruebas se realizan en tiempo de desarrollo por lo que la corrección de defectos detectados en esta etapa son lo más eficientes en termino de costo y de tiempo.

Pruebas de componentes: prueba una unidad de compilación de forma aislada, validando como una caja blanca e ignorando todos los servicios externos.

Pruebas de integración: Este tipo de pruebas aseguran la correcta integración del componente desarrollado o modificado con los componentes ya existentes.

Pruebas de API: valida el contrato de API de componente.

Pruebas UI: Pruebas funcionales E2E que validan todo el sistema trabajando con dependencias, desde la perspectiva del usuario final. Estos son los más caros de ejecutar, ya que requieren todas las capas internas y externas para evitar falsos positivos ambientales.

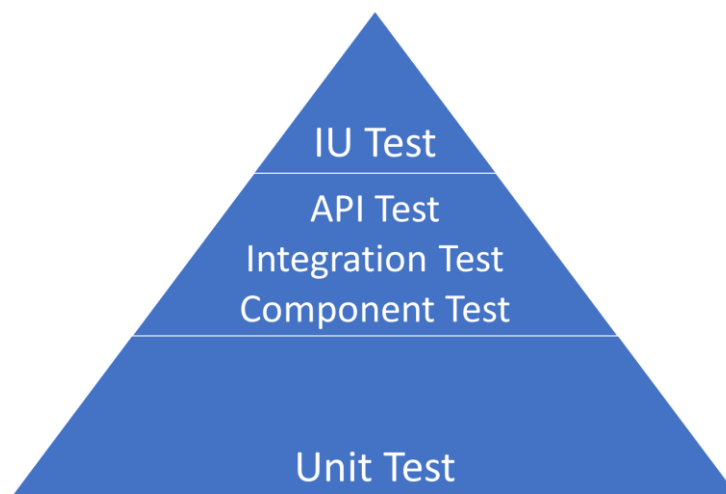


Figura 13. Pirámide de Cohn
Fuente: *(Elaboración propia)*

2.3.8 DevOps y agilidad

El DevOps Provee el soporte tecnológico para permitir el desarrollo de software ágil. Refuerza la entrega de Software iterativa e incremental, que es el enfoque ágil que permite a las empresas avanzar más rápido hacia las necesidades cambiantes del mercado.

Continuous Delivery es una práctica de DevOps que refleja claramente esta simbiosis. Permite el "desarrollo ágil de software" al reducir el costo de cada iteración y aumentar la calidad. También requiere una incorporación temprana de Operaciones al proceso de desarrollo, ya que el primer incremento del producto se integra y se entrega como candidato de lanzamiento.

Principales beneficios ágiles al adoptar DevOps:

- Ciclos más rápidos
- Incrementos más pequeños
- Eliminar desechos e impedimentos
- Mejorar a través de comentarios
- Incrementar la colaboración
- Empoderar a los equipos
- Mejor visibilidad del trabajo

Tabla 2: Principios Ágiles y DevOps fuertemente alineados

PRINCIPIOS AGILES	PRINCIPIOS DEVOPS
Satisfacción del cliente mediante la entrega temprana y continua de valor	Acciones centradas en el cliente
Entrega frecuente de software funcionando (semanas en lugar de meses)	Crear con el cliente en mente
El software funcionando como principal medida de progreso	Responsabilidad de punta a punta
Atención continua de la excelencia técnica	Automatizar todo lo que sea posible
Las mejores arquitecturas y diseños surgen de equipos autoorganizados	Equipos autónomos y multifuncionales
El equipo reflexiona sobre como ser más eficiente y se ajusta para ello	Mejora continua

CAPÍTULO 3: PROYECTO

3.1 Antecedentes de la empresa

Banco SCL tiene como objetivo principal ser el mejor banco de la plaza chilena, operando de forma comprometida con la sociedad y captando la lealtad de sus clientes, accionistas, colaboradores y comunidades. El propósito es de ayudar a las personas y empresas a prosperar. Para esto propone a sus clientes productos fáciles de comprender, siendo ágiles y resolutivos en las respuestas a sus necesidades. Para ello, el banco, requiere que los sistemas y procesos sean fáciles de operar, permitiendo a los empleados entregar respuestas claras con la agilidad que los clientes requieren.

Banco SCL detalla en sus valores la orientación al cliente, entendiendo que, como toda empresa de servicios, el cliente es la razón de ser. Trabajando para buscar y entregar soluciones que simplifiquen su vida de este y le permitan vivir la mejor experiencia de servicio con el objetivo de poder cumplir sus sueños y hacer realidad sus proyectos en una relación duradera basada en la confianza y excelencia.

Tiene un fuerte foco en la banca de personas ya que aproximadamente dos tercios del resultado neto de la operación se basa en la banca comercial, liderando el mercado con créditos de consumo e hipotecarios, además de ser uno de los líderes en números de cuentas corrientes, tarjetas de crédito y depósitos. Banco SCL se define como un banco para todos, ya que atiende necesidades de todo tipo de clientes, personas de diferentes niveles de ingreso como también a empresas de distintos tamaños, organismos públicos y corporaciones privadas.

La industria bancaria es reglamentada por la Comisión para el Mercado Financiero (CMF). La CMF es el organismo que se encarga de fiscalizar a las entidades bancarias, las actividades que se generan en el mercado financiero y también a las actividades de los seguros que operan en Chile.

3.2 Levantamiento de la Situación Actual

Para realizar el levantamiento de la situación actual e identificar el proceso en donde se centrará el proyecto, se utilizará lo expuesto en la metodología de la ingeniería de negocios, específicamente el framework de arquitectura de procesos que fue revisado en el capítulo anterior, en donde se explica que toda organización puede ser representada por los 4 macroprocesos.

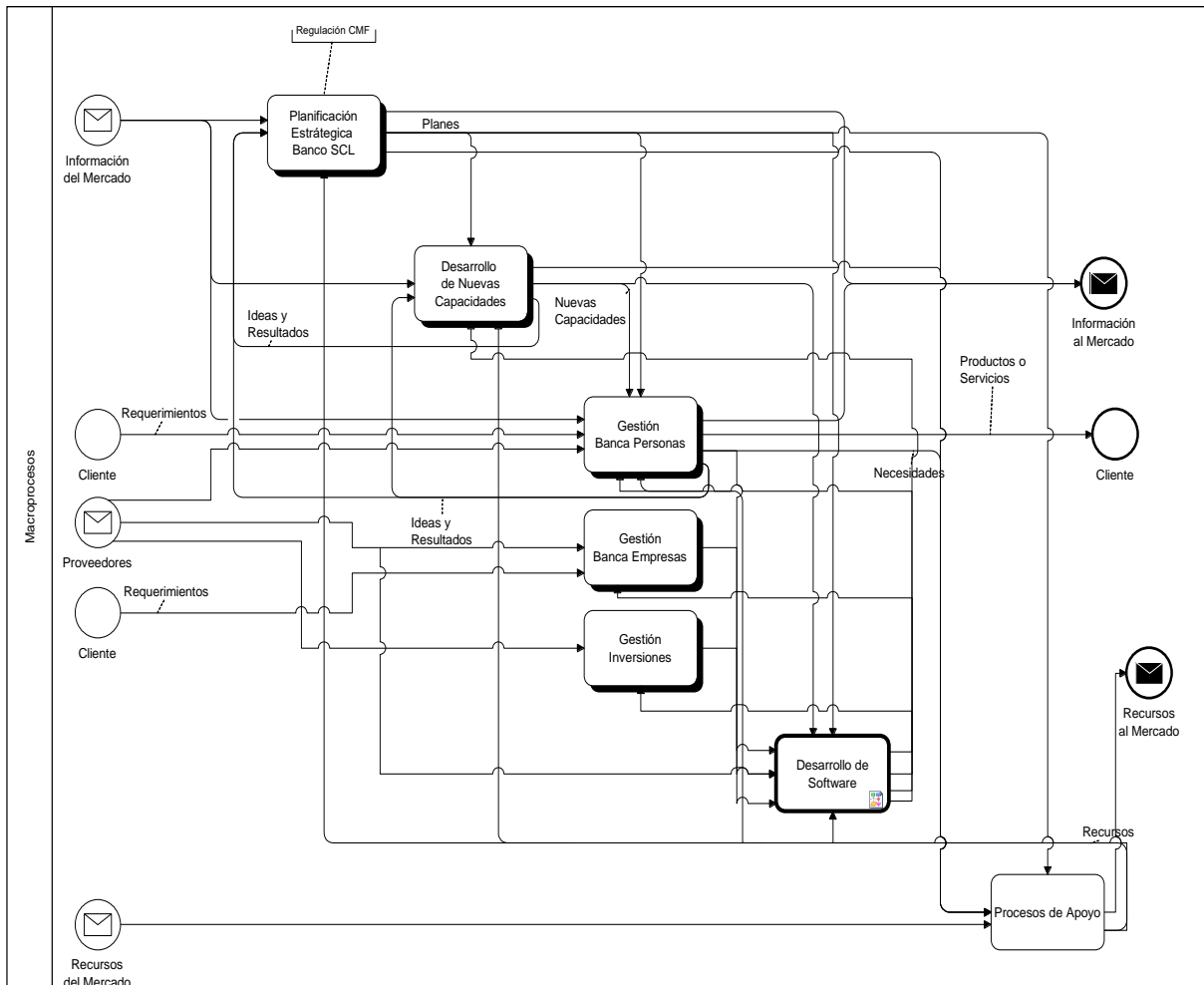


Figura 14. Arquitectura de Banco SCL
Fuente: (Elaboración propia)

Para este trabajo utiliza la arquitectura Unificación de Servicio compartido que consiste en una arquitectura compuesta por varias cadenas de valor que comparten servicios de apoyo y también los servicios compartidos. En el caso del Banco SCL, el servicio compartido corresponde al Desarrollo de Software que presta sus servicios a todas las cadenas de valor del banco, tal como se puede apreciar en la figura 14.

A continuación, se realiza una descripción de los macroprocesos identificados en el levantamiento realizado desde la perspectiva de ingeniería de procesos:

- **Cadena de Valor Gestión Banca Personas (Macro 1):**
En este Macroproceso se encuentran el conjunto de procesos que establece la producción de la Gestión de Banca Clientes
- **Cadena de Valor Gestión Banca Empresas (Macro 1)**
En este Macroproceso se encuentra el conjunto de procesos que establece la producción de la Gestión de Banca Empresas
- **Cadena de Valor Gestión Inversiones (Macro 1)**
En este Macroproceso se encuentra el conjunto de procesos que establece la producción de la Gestión de Inversiones
- **Desarrollo de Nuevas Capacidades (Macro 2)**
En este Macroproceso se encuentra el conjunto de procesos que desarrollan las nuevas capacidades de Banco SCL para ser un banco competitivo, por ejemplo, el desarrollo de nuevos productos digitales.
- **Planificación estratégica (Macro 3)**
En este Macroproceso se encuentra el proceso de planificación estratégica del banco, que define el futuro de la organización en base a programas, proyectos y planes que son aplicados para conseguir los objetivos definidos.
- **Gestión de recursos (Macro 4)**
En este Macroproceso se encuentra el conjunto de procesos de apoyo que son ejecutados para que los otros procesos puedan operar de manera satisfactoria.

Este trabajo se enfoca en la Macro de Servicio Compartido Desarrollo de Software que será revisado en mayor detalle a continuación

3.2.1 Servicio compartido Desarrollo de Software

Al observar únicamente la macro de Servicio Compartido Desarrollo de Software, que se representa de manera aislada en la figura 16, se puede apreciar la relación con las cadenas de valor Gestión Banca Persona, Gestión Banca Empresas y Gestión Inversiones y sus respectivas respuestas.

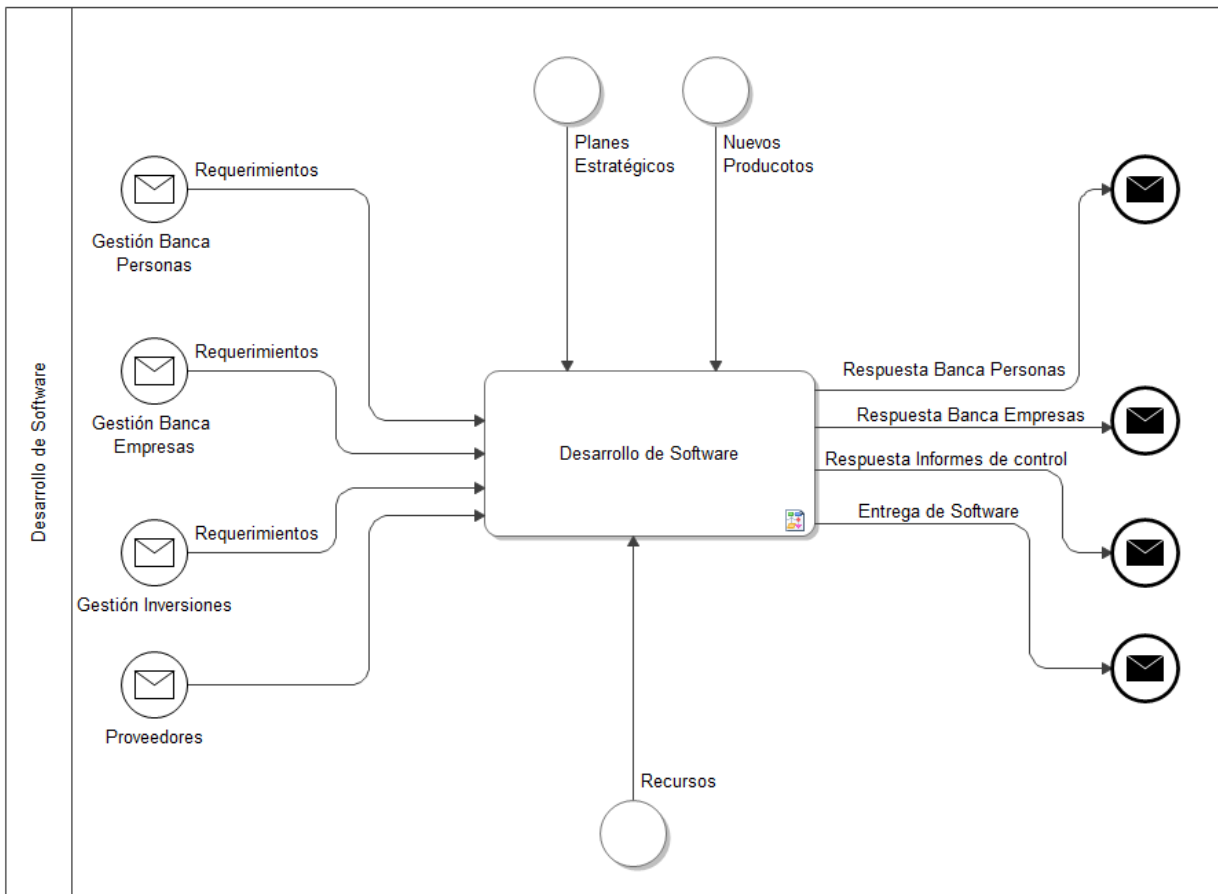


Figura 15. Macro Servicio Compartido Desarrollo de Software
Fuente: (Elaboración propia)

3.2.2 Cadena de Valor del servicio compartido Desarrollo de Software

La realización de este trabajo se concentra únicamente en la Macro 1 del servicio compartido Desarrollo de software, en donde se pueden observar los siguientes procesos que son representados en la figura 16.

- **Administración de Relación con el usuario:**
El servicio compartido Desarrollo de Software se relaciona con Banco SCL a través de los Usuarios designados quienes son los encargados de realizar requerimientos de software a Desarrollo de Software

- Administración de Relación con Proveedores:**
 El servicio compartido Desarrollo de Software cuenta con varios proveedores que le permiten ampliar su capacidad de desarrollar y entregar servicios a Banco SCL.
- Gestión de Producción y Entrega Software:**
 Preparación, planificación, control y entrega de nuevos desarrollos de software que son solicitados por los distintos usuarios de las Cadenas de Valor de Banco SCL
- Producción y Entrega de Software:**
 Producción del desarrollo de software y respectiva entrega.

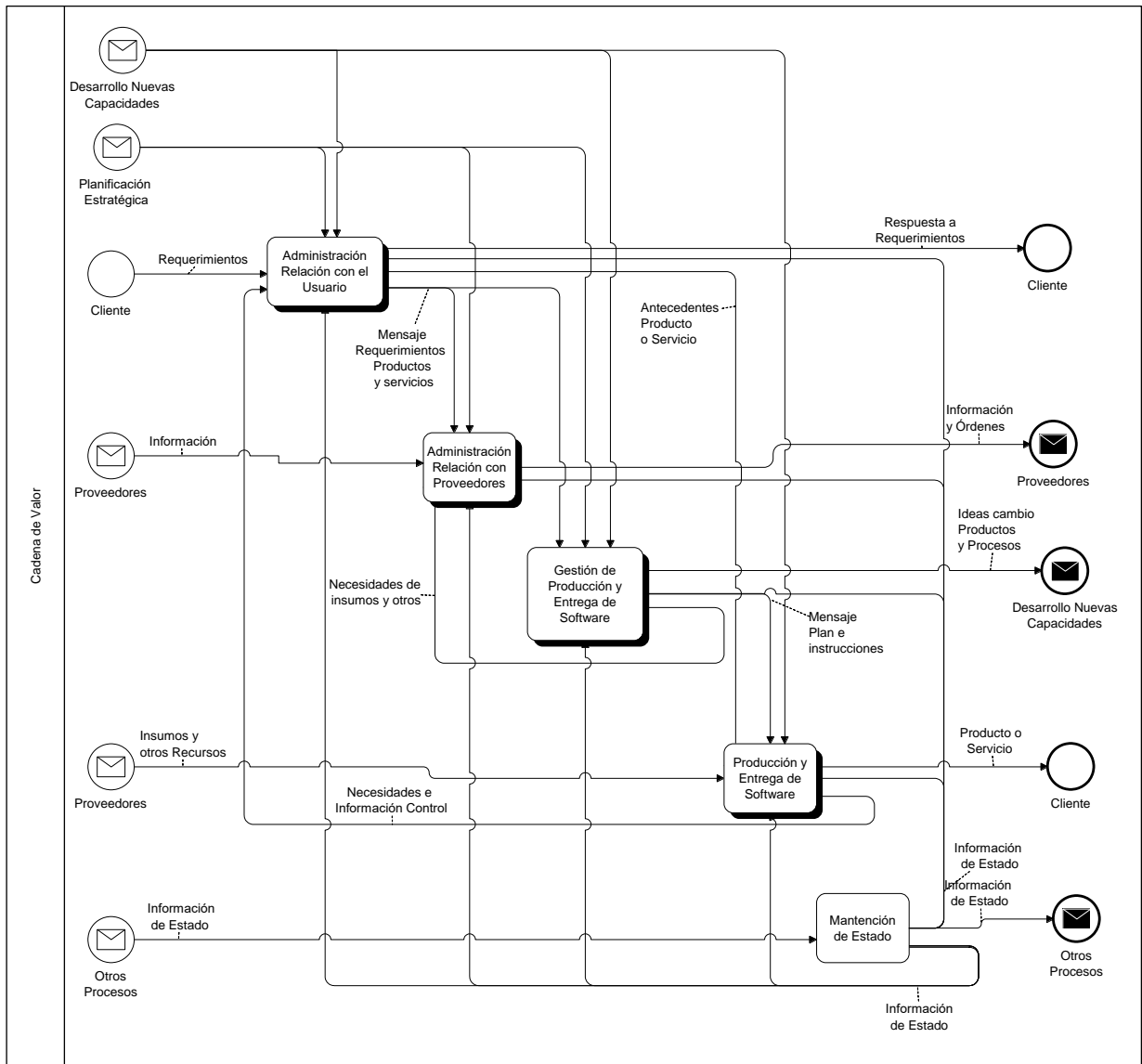


Figura 16. Cadena de Valor del Servicio Compartido Desarrollo de Software
 Fuente: (Elaboración propia)

3.2.3 Administración de Relación con el Usuario

La gerencia de desarrollo de software se relaciona con sus usuarios a través la gestión de requerimientos que atiende a las necesidades del usuario, para ello el usuario confecciona un documento de especificación de requerimientos con el cual el equipo de desarrollo puede comenzar con la construcción del software. Durante el proceso de construcción se realizan reuniones de avance hasta la entrega del prototipo funcionando con el que se decide si se satisfacen las necesidades del usuario.

- **Atención al usuario:**
Se reciben los requerimientos ingresados por el usuario, los que luego son validados en el siguiente proceso para validar si son satisfactorios.
- **Decidir satisfacción de requerimientos:**
En esta etapa se toma la decisión de si el requerimiento es apto para pasar a la siguiente fase de construcción de software o debe ser devuelto al usuario para su completitud.

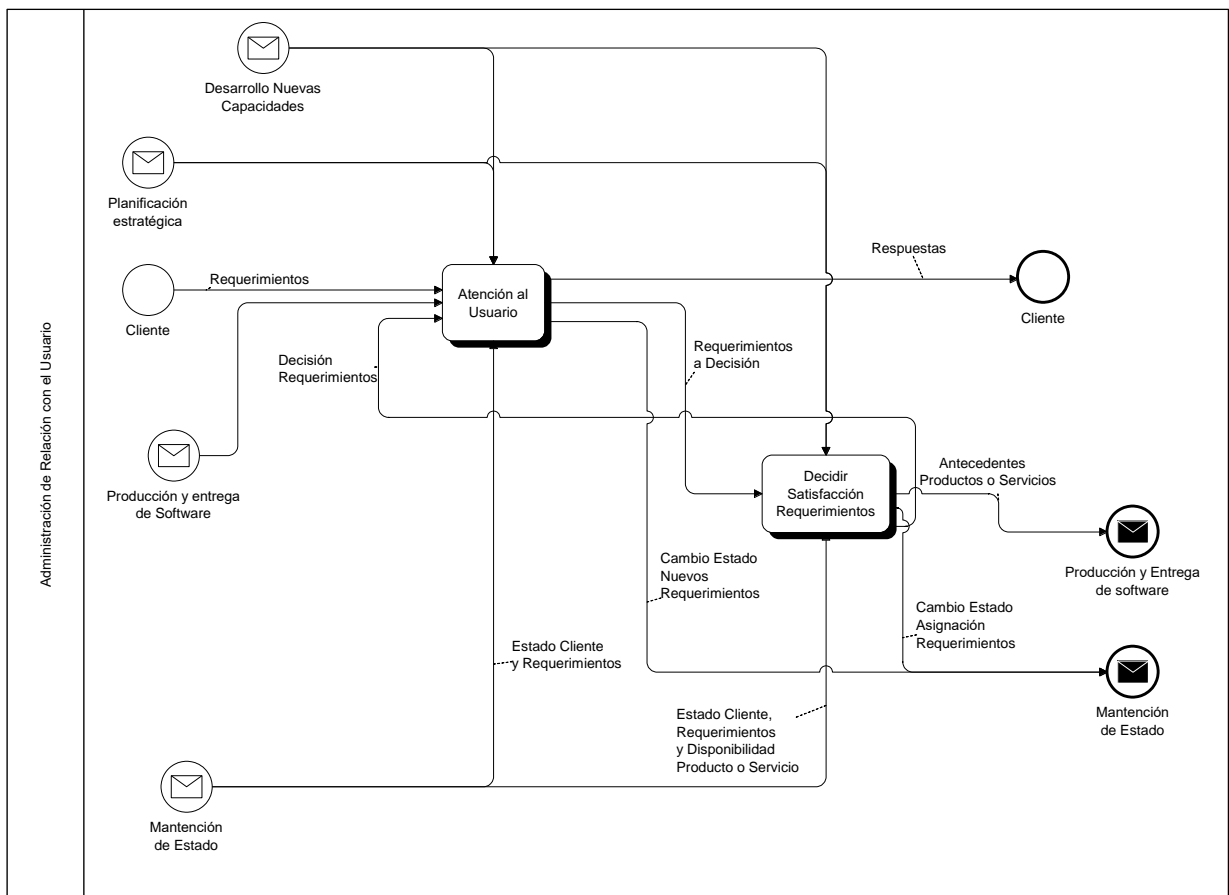


Figura 17. Administración de relación con el Usuario
Fuente: (Elaboración propia)

3.2.4 Gestión de Producción y Entrega de Software

En este proceso, el equipo de gestión de desarrollo de software prepara y planifica la construcción del software a partir del requerimiento recibido del proceso anterior. Se define el alcance, definen los plazos de seguimiento y fecha de entrega de todo el producto terminado.

- **Implementación desarrollo de software:**
Se reciben los requerimientos ingresados por el usuario, en base a ellos se realiza una estimación de la implementación del software.
- **Planificación y control de producción:**
Se reciben la estimación de los requerimientos, se realiza la planificación con su respectivo cronograma de trabajo para ser asignado a un equipo.
- **Decidir entrega Desarrollo de software:**
En base a la estimación y planificación se realiza la decisión de entrega del requerimiento a un equipo para la construcción del software.

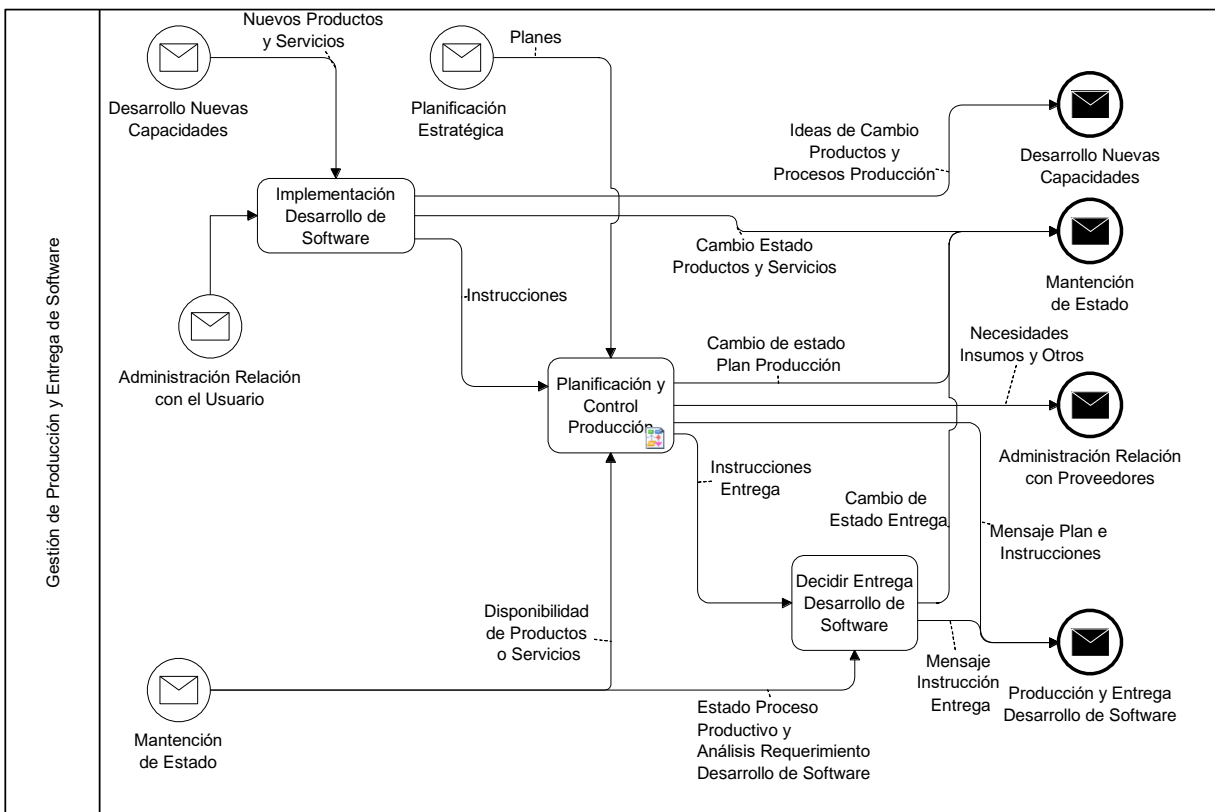


Figura 18. Gestión de producción y Entrega de Software
Fuente: (Elaboración propia)

3.2.5 Planificación y Control Producción

En este proceso se realiza seguimiento del avance para validar si se cumple la planificación acordada con el usuario, para lo cual se utiliza una carta Gantt. Dependiendo del porcentaje de avance de las piezas de software construidas vs las planificadas, se puede inferir si el proyecto está bien avanzado o con algún retraso. También es posible detectar posibles bloqueos de las actividades y gestionar soluciones para poder avanzar con los hitos del proyecto.

- **Preplanificación:**
Se realiza la definición de métricas para medir el avance del proyecto y los umbrales de desvío permitidos para cada proyecto.
- **Planificación Métricas de Software:**
Se realiza planificación de métricas para medir el avance o desvío del proyecto.
- **Control y seguimiento de software:**
Se realiza seguimiento del avance del desarrollo de software para asegurar que se cumple con la planificación.

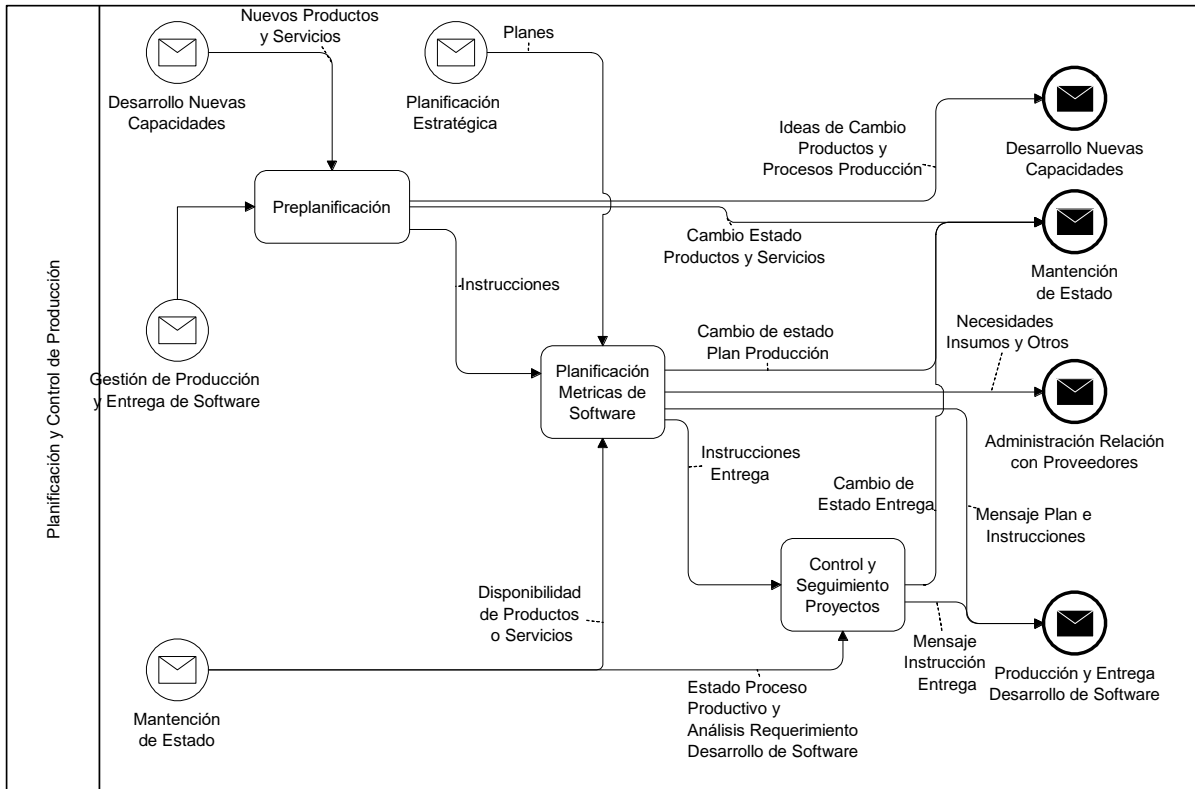


Figura 19: Gestión de producción y entrega de software
Fuente: (Elaboración propia)

3.2.6 Producción y Entrega de Software

En este proceso se realiza la coordinación de todas las etapas del proceso de construcción del software y la coordinación del proceso de entrega.

- **Producción de Software:**
En esta etapa se realizan todas las actividades propias del ciclo de vida de construcción de software.
- **Entrega de Software:**
Se realiza el proceso de entrega de software a la producción

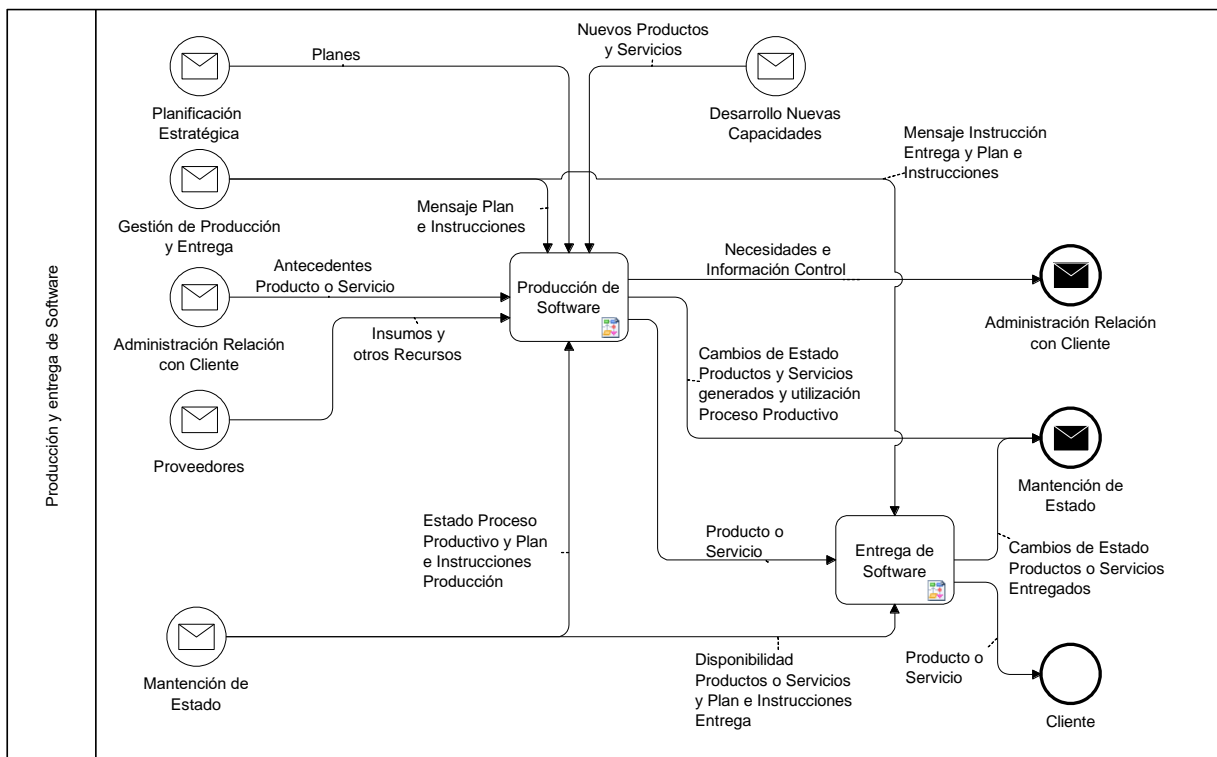


Figura 20: Producción y entrega de software
Fuente: (Elaboración propia)

3.2.8 Entrega de Software

En este proceso se entrega el software ya construido y probado al usuario, para que este realice sus pruebas de aceptación, en donde validará si la entrega cumple con sus requerimientos.

- **Pruebas de aceptación:**
El software construido es entregado al usuario para que realice las pruebas de aceptación de usuario.
- **Capacitación a Usuario:**
Se realiza capacitación del nuevo software a todos los usuarios que participan del flujo.

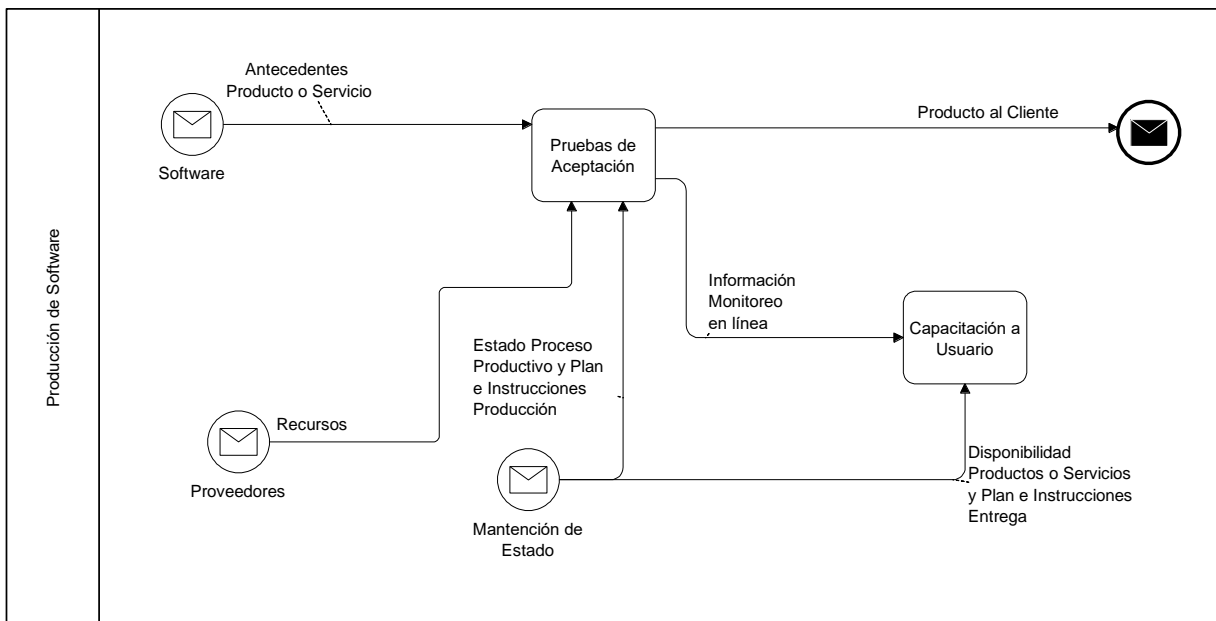


Figura 22: Entrega de Software
Fuente: (Elaboración propia)

3.3 Conclusión del levantamiento

El proceso actual de Desarrollo de Software está diseñado para realizar grandes proyectos en periodos largos de tiempo. Se recibe un gran requerimiento y se trabaja en ellos durante un prolongado periodo de tiempo hasta tenerlo finalizado por completo.

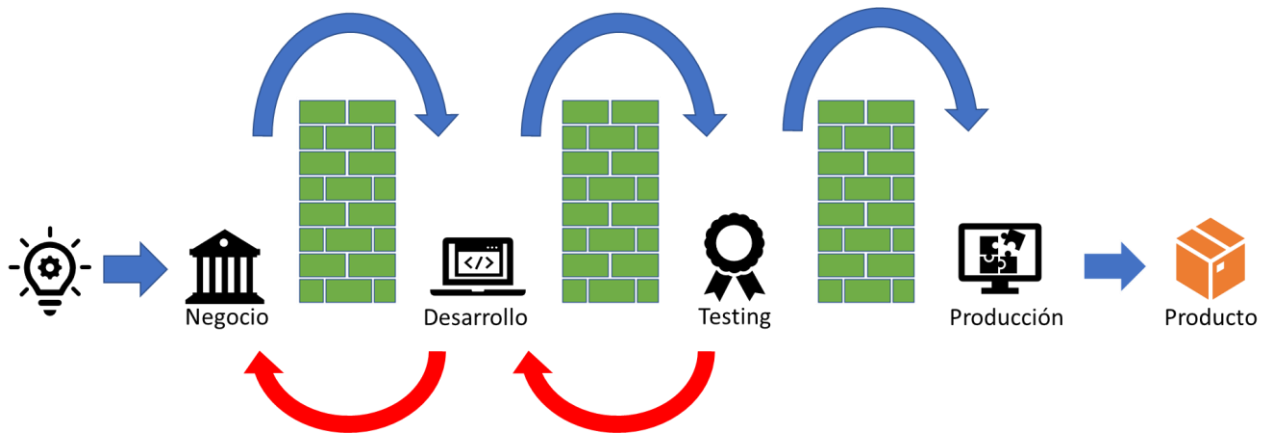


Figura 23: Proceso de despliegue
Fuente: *(Elaboración propia)*

Es un proceso en silos, rígido y secuencial, con escasa participación de los usuarios y prácticamente en su totalidad, las actividades son manuales. En las etapas del proceso se generan iteraciones y retrabajo que finalmente redundan en ineficiencias y atentan con el objetivo actual de realizar cambios pequeños en periodos cortos de tiempo.

3.4 Modelo conceptual de los procesos impactados

La siguiente figura representa de manera conceptual, en un esquema de árbol, los procesos que son impactados en este trabajo de rediseño.

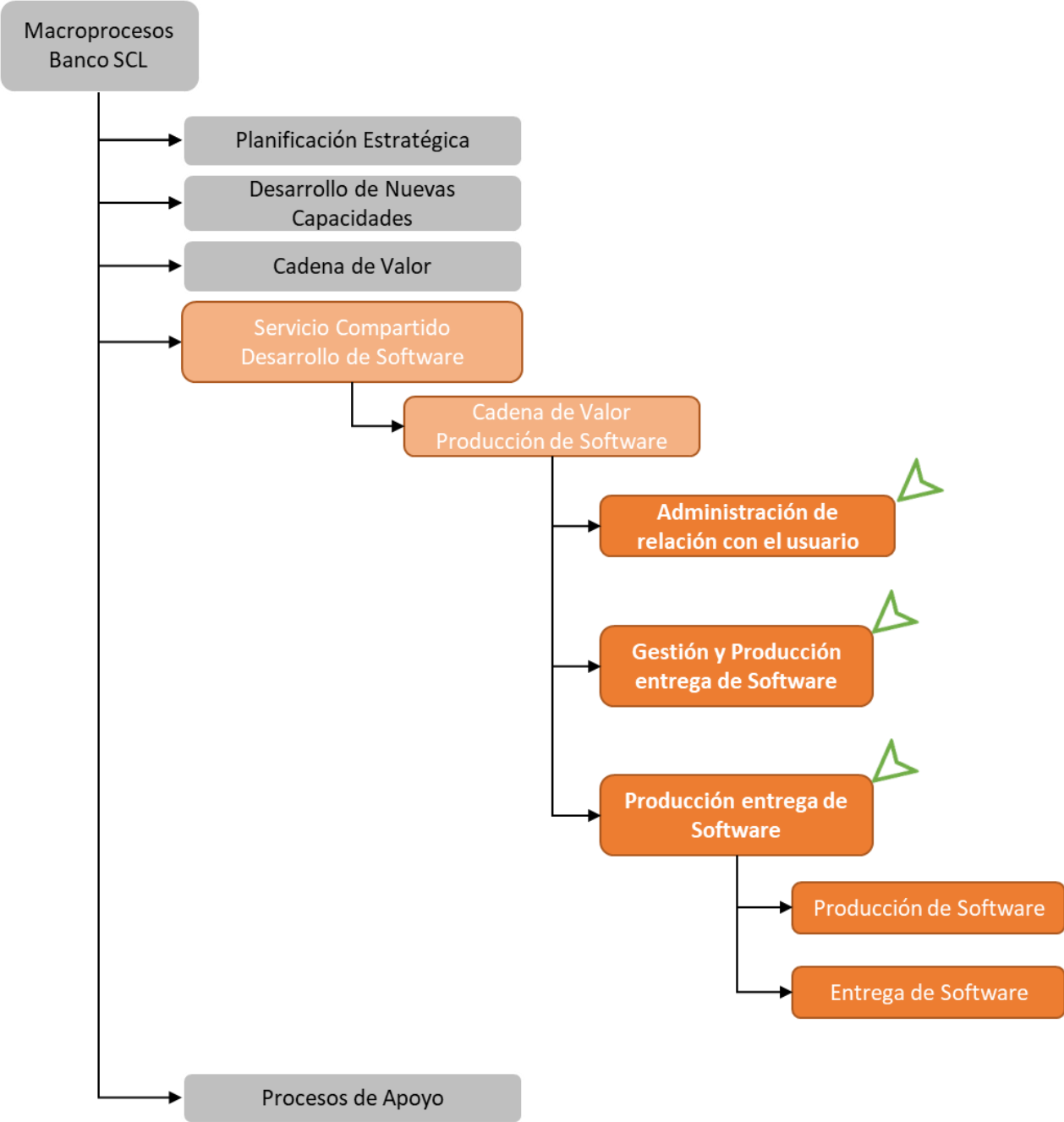


Figura 24. Procesos impactados
Fuente: *(Elaboración propia)*

- **Administración de Relación con el Usuario**

En este proceso se trabajará el modelo de relación con el usuario ya que, en el modelo actual, el usuario es un requirente y un receptor de la solución terminada, pero no es parte del proceso.

- **Gestión y producción entrega de software**

Este proceso se verá impactado en la manera que se gestiona el desarrollo de construcción, incorporando las ceremonias ágiles para la priorización y seguimiento de software.

- **Producción entrega de Software**

Este proceso será el más impactado ya que es aquí en donde se construye y entrega el software.

3.5 Diagnóstico de la Situación Actual

3.5.1 Problemas Identificados

El proceso de despliegue de nuevo software y de nuevas versiones del software existente se realiza utilizando un proceso manual en el que se producen constantes fallas que genera retrabajo y un sobre costo de los proyectos. Según los registros del área de despliegues de Banco SCL, un 13% de los pasos a producción tienen algún defecto mayor.

En caso de que el defecto sea detectado en la certificación realizada posterior al paso a producción, se realiza una vuelta atrás del cambio de software, esto equivale al 10,3% de los pasos a producción. Por otra parte, el 2,7% corresponde a defectos que no fueron detectados y generan un incidente productivo que afecta a los clientes.

Para una institución financiera como lo es Banco SCL y con las pretensiones que persigue, tener un 2,7% de incidentes productivos que implican que un software deje de operar o que afecta al servicio de cara a los clientes y daña la reputación como entidad bancaria líder, es un problema considerable. Además del retraso en la entrega al mercado de las soluciones digitales

Al realizar el análisis de los fallos, se pueden identificar distintas falencias del proceso de desarrollo de software y la implementación de este, siendo los más representativos los siguientes:

- El repositorio de código no siempre contiene la última versión del software lo que genera que defectos que fueron corregidos anteriormente vuelvan a aparecer.
- La definición de los requerimientos y los criterios de aceptación generan problemas de interpretación.
- El proceso de despliegues es manual, siguiendo las instrucciones de una pauta de instalación en documento escrito a mano y que es reutilizado. Esto genera errores de confección y posterior interpretación del operador que ejecuta el proceso.
- Ausencia de testing continuo que permite realizar test en todas las etapas del desarrollo de software, lo que finalmente redundando en un proceso de pruebas concentrado con tiempo acotado y apurado que no permite la cobertura de pruebas apropiada.

Para profundizar en los problemas descritos anteriormente, se realiza un diagrama de causa y efecto, más conocido como diagrama de Ishikawa con el fin de representar el problema junto con las causas que lo generan, para luego describir cada una de ellas y posteriormente trabajar sobre las posibles soluciones

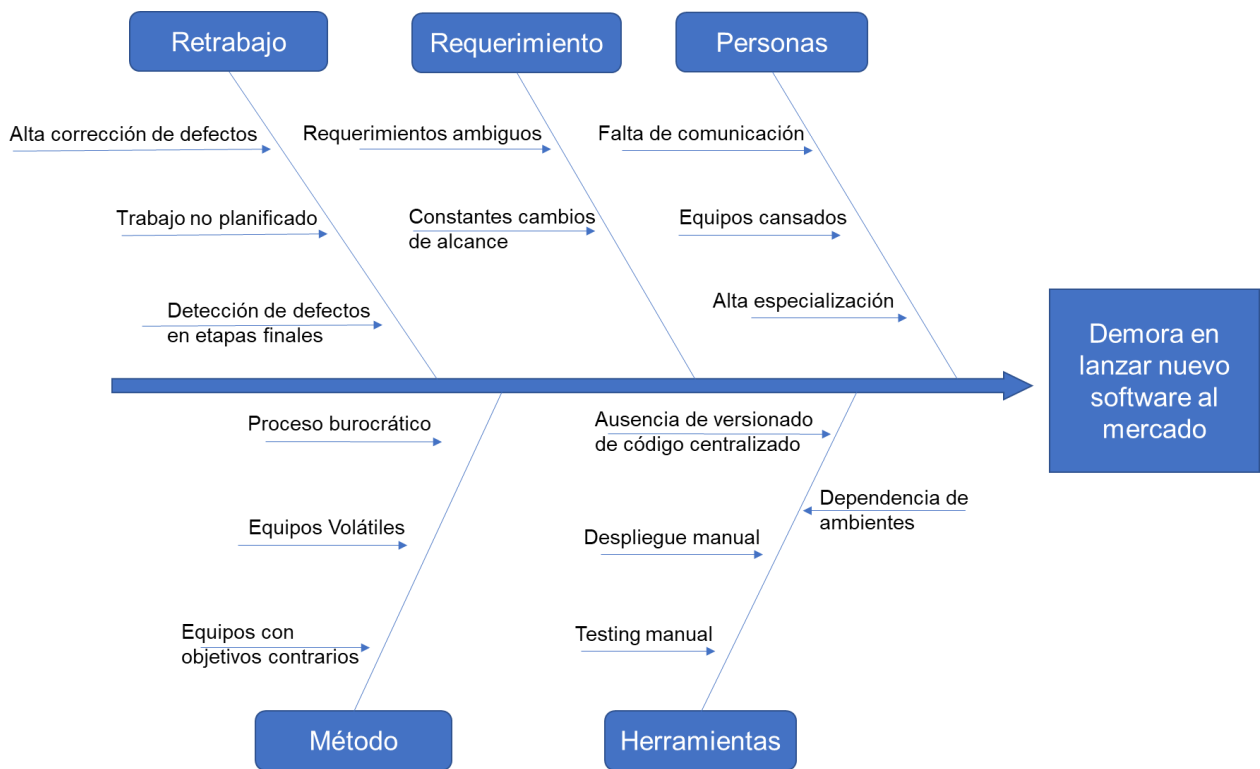


Figura 25. Diagrama Ishikawa
Fuente: (Elaboración propia)

- **Personas**

- Fata de comunicación: Este problema de comunicación ocurren en todos los equipos que participan del ciclo de vida del software. Los equipos trabajan en silos y comienzan la construcción de sus proyectos sin darse cuenta de que otros equipos que previamente construyeron herramientas que pueden ser útiles para su proyecto, lo que terminan siendo un problema por duplicar componentes y además porque se invierte tiempo valioso en construir algo que ya existía.
- Alta especialización: Dentro de toda la gerencia de desarrollo, sólo algunos ingenieros son los especialistas en la tecnología MainFrame de Banco SCL, esto implica que cualquier proyecto que deba trabajar con MainFrame debe esperar que alguno de estos profesionales tenga disponibilidad para atender los requerimientos del proyecto.
- Equipos cansados: El círculo poco virtuoso en que los productos salen con retraso, la presión que esta situación genera y, además, el retrabajo que se crea dentro de todo el proceso ha impactado fuertemente en los equipos de construcción de software, quienes demuestran signos de agotamiento y fallas que redundan en incidentes productivos.

- **Requerimiento**

- Requerimientos Ambiguos: Los requerimientos son grandes documentos con mucha información que no expresan con claridad el requerimiento, esto ocurre principalmente por la complejidad del documento y criterios de aceptación poco claros.
- Cambios de alcance: Conforme avanzan las semanas, es habitual que los usuarios constantemente soliciten cambios en los requerimientos del software, lo que muchas veces implica desechar e trabajo previamente realizado lo genera retrasos en la entrega.

- **Retrabajo**

- Alta corrección de defectos: La ausencia de un proceso de testing continuo implica que las pruebas se concentren en una etapa posterior al desarrollo de software lo que genera un alto volumen de defectos en estas etapas.

- Trabajo no planificado: La alta rotación de los equipos implica asignar tiempo no considerado para capacitar y acompañar a los nuevos profesionales.
- Detección de efectos en etapas finales: La detección de defectos en las etapas finales generan un gran problema debido a que para llegar a esta etapa se deben pasar varios controles obligatorios, los cuales se tendrán que volver a realizar dado que el software volverá a la etapa de desarrollo para corregir los defectos detectados.

- **Herramientas**

- Ausencia de versionado de código centralizado: La ausencia de un repositorio de código fuente centralizado implica que las versiones del software se encuentran repartida en distintos equipos y no sé sabe con certeza que si se está trabajando con la última versión del software que ya se encuentra en producción.
- Ausencia de despliegue automatizado: La ausencia de una herramienta de despliegue de software hacia los distintos ambientes incorpora múltiples puntos de falla con la realización manual de esta tarea, esto ocurre porque el proceso actual implica una pauta de instalación creada por el desarrollador y una interpretación y ejecución del paso a paso realizada por un operador de despliegue.
- Ausencia de Testing Continuo: La ausencia de un proceso de testing continuo durante todo el proceso, implica que la etapa de testing manual se realiza y concentra todas las pruebas en la etapa posterior al desarrollo generando que el software debe volver a etapas anteriores para la corrección de los defectos.

- **Método**

- Metodología no estandarizada: La ausencia de un framework metodológico genera que los distintos equipos trabajen bajo sus prácticas y propias definiciones, las que luego, no son compatibles con otros equipos de trabajo y dificulta la integración.
- Proceso burocrático: Dada la tasa de incidentes detectados, el proceso para realizar un despliegue a producción es pesado y engorroso, lo que complica flujo de lanzar nuevo software en el mercado.

- Equipos volátiles: Dada la urgencia y atraso con ciertos proyectos estratégicos, los equipos de desarrollo son intervenidos en pro de apoyar proyectos estratégicos y genera retraso en el proyecto que es intervenido.
- Equipos con objetivos contrarios: Los distintos equipos que participan en el proceso de entregar software a la producción tienen objetivos que se contraponen, dado que el equipo de desarrollo debe cumplir con entregar el software en la fecha estipulada, el equipo de calidad debe garantizar que el software está correcto sin importar la fecha y el equipo de operaciones de la producción debe garantizar la estabilidad y continuidad operacional de los sistemas del banco.

3.6 Generación de Alternativas

En base al trabajo realizado para detectar y organizar las causas del problema, se evalúan distintas alternativas de solución para solucionar las causas y resolver el problema detectado: “Demora en lanzar nuevos productos al mercado”. La siguiente tabla contiene alternativas para cada causa detectada:

Tabla 3: Alternativas de solución

Nro	Ámbito	Causa	Alternativa de solución
1	Personas	Falta de comunicación	Generar reuniones masivas para que todos los equipos tomen conocimiento de los trabajos que se están iniciando.
2	Personas	Falta de comunicación	Crear un portal de información para publicar el trabajo que cada equipo está abordando y los otros proyectos tomen conocimiento.
3	Personas	Falta de comunicación	Adoptar un modelo de trabajo que fomente la colaboración y entre los distintos equipos de trabajo
4	Personas	Equipos Cansados	Incorporar más personas a los equipos para distribuir las tareas entre los nuevos profesionales con el objetivo de descomprimir la presión de las personas del equipo.

5	Personas	Equipos Cansados	Incorporar automatizaciones en el proceso para disminuir el número de actividades de las personas.
6	Personas	Alta especialización	Crear un equipo dedicado al trabajo con plataformas Mainframe y contratar más profesionales para potenciar a dicho equipo y sea capaz de atender los requerimientos de los otros equipos.
7	Retrabajo	Alta corrección de defectos	Incorporar un modelo de revisión de pares entre los desarrolladores para detectar los defectos cuando se están construyendo.
8	Retrabajo	Alta corrección de defectos	Anticipar la etapa pruebas, incorporándolas desde el inicio del desarrollo y en todas las etapas hasta llegar a producción.
9	Retrabajo	Detección de defectos en etapas finales	Incorporar un modelo de revisión de pares entre los desarrolladores para detectar los defectos cuando se están construyendo.
10	Retrabajo	Detección de defectos en etapas finales	Anticipar la etapa pruebas, incorporándolas desde el inicio del desarrollo y en todas las etapas hasta llegar a producción.
11	Retrabajo	Trabajo no planificado	Crear una plataforma de auto capacitación para no invertir tiempo de los miembros del equipo en capacitar a los nuevos profesionales.
12	Requerimiento	Requerimientos ambiguos	Rediseñar los documentos para evitar ambigüedad en su construcción.
13	Requerimiento	Requerimientos ambiguos	Incorporar a los usuarios como parte continua del del proceso de desarrollo de software para

			generar feedback constante entre los equipos.
14	Requerimiento	Constantes cambios de alcance	Limitar el número permitido de cambios de alcances por requerimientos y crear un modelo que desincentive los cambios propuesto por el usuario, agregando días adicionales a la fecha de entrega establecida.
15	Requerimiento	Constantes cambios de alcance	Incorporar a los usuarios como parte continua del proceso de desarrollo de software para generar feedback constante entre los equipos.
16	Requerimiento	Constantes cambios de alcance	Incorporar una metodología que permita construcciones pequeñas y constantes que faciliten la construcción y la entrega de valor de manera continua.
17	Herramienta	Ausencia de versionado de código centralizado	Implementar un portal de información para registrar que Equipo/Desarrollador está utilizando el código fuente para coordinar los cambios de software.
18	Herramienta	Ausencia de versionado de código centralizado	Implementar un modelo de trabajo DevOps que permita el versionado de código y la automatización de procesos y pruebas.
19	Herramienta	Ausencia de despliegue automatizado	Desarrollar una herramienta para implementar un modelo de despliegue de software en base a la parametrización de cada aplicación.
20	Herramienta	Ausencia de despliegue automatizado	Implementar un modelo de trabajo DevOps que permita el versionado de código y la automatización de procesos y pruebas.

21	Herramienta	Ausencia de Testing Continuo	Incorporar un modelo de revisión de pares entre los desarrolladores para detectar los defectos cuando se están construyendo.
22	Herramienta	Ausencia de Testing Continuo	Implementar un modelo de trabajo DevOps que permita el versionado de código y la automatización de procesos y pruebas.
23	Método	Metodología no estandarizada	Construir un modelo de integración que permita la compatibilidad entre todas las metodologías existentes.
24	Método	Equipos volátiles	Crear un equipo extra, dedicado a prestar apoyo a los proyectos estratégico que se atrasan para evitar desarmar los equipos de los proyectos.
25	Método	Equipos con objetivos contrarios	Definir metas consolidadas a los distintos equipos para evitar se generen situaciones no colaboración.

3.7 Evaluación de Alternativas

Junto con el proceso de generar alternativas, surgen hipótesis para resolver cada una de las causas del problema detectado. Dichas hipótesis son representadas y evaluadas en la siguiente Tabla:

Tabla 4: Evaluación de alternativas

Nro	Hipótesis	Nivel de impacto	Nivel de esfuerzo
1	Los equipos que participan de las reuniones estarían informados de todos los proyectos y podrían colaborar.	Bajo	Medio
2	Creando un portal de información con el trabajo que cada equipo está realizando, los otros proyectos tomarían conocimiento.	Alto	Alto
3	Adoptando un modelo de colaboración que fomente el trabajo entre los equipos se generará sinergia y mejora continua	Alto	Alto
4	Incorporando más profesionales a los proyectos, las personas podrían estar menos presionadas y cometer menos errores.	Alto	Medio
5	Incorporando automatizaciones las personas podrían liberar su carga con menores cantidad de tareas que realizar	Alto	Alto
6	Si se crea un equipo dedicado para atender plataformas mainframe se podrían atender de mejor manera estos requerimientos.	Alto	Bajo
7	Incorporando un modelo de revisión de pares para los desarrolladores se podría disminuir la cantidad de defectos	Alto	Medio
8	Anticipando las pruebas desde las etapas tempranas implica corrección inmediata evitando retrabajo posterior.	Alto	Medio
9	Incorporando un modelo de revisión de pares para los desarrolladores	Alto	Medio

	se podría evitar llegar con defectos en las etapas finales		
10	Anticipando las pruebas desde las etapas tempranas implica evitar la detección de defectos en etapas finales.	Alto	Medio
11	Creando una plataforma de auto capacitación se podría eliminar el tiempo que se pierde al capacitar a nuevos profesionales.	Alto	Bajo
12	Rediseñando los documentos se podría erradicar los requerimientos ambiguos.	Alto	Bajo
13	Incorporando a los usuarios como parte continua del proceso se lograría generar retroalimentación constante evitando ambigüedades	Alto	Medio
14	Limitando los cambios de alcance o generando un castigo por cada cambio, se lograría disminuir que se genere un alto número de cambios en la etapa de construcción	Bajo	Alto
15	Incorporando a los usuarios como parte continua del proceso se lograría generar retroalimentación constante evitando ambigüedades	Alto	Medio
16	Incorporando un modelo que permita construcciones pequeñas y constantes lograría generar una estrategia de entrega de valor continua.	Alto	Medio
17	Implementando un portal de consulta para informar que código fuente está siendo utilizado por los equipos lograría evitar que distintos equipos trabajen sobre la misma versión código	Bajo	Alto
18	Implementando herramientas que permitan el modelo de DevOps se lograría obtener un repositorio de código como única fuente de la verdad	Alto	Alto

19	Construyendo una herramienta que permita el despliegue del software en base a la parametrización eliminaría los errores manuales	Alto	Alto
20	Implementando herramientas que permitan el modelo de DevOps se lograría obtener automatización en los despliegues y eliminar errores producto de la manualidad	Alto	Alto
21	Incorporando un modelo de revisión de pares entre los desarrolladores se podría disminuir la cantidad de defectos	Medio	Medio
22	Implementando herramientas que permitan el modelo de DevOps se lograría obtener testing Continuo	Alto	Alto
23	Creando un modelo de compatibilidad permitiría la fluida integración entre todas las metodologías	Alto	Medio
24	Creando un equipo para atender los proyectos estratégicos que se retrasan, evitaría que se utilicen profesionales de los otros proyectos	Medio	Medio
25	Definiendo metas comunes se generaría mayor sinergia entre los equipos	Alto	Bajo

Con la información levantada fue posible realizar el análisis correspondiente y determinar cuáles hipótesis son factibles de implementar en base al Alto Impacto y el beneficio transversal que estas agregaría al proceso Desarrollo de Software.

Sumado a lo anterior, las hipótesis también han sido seleccionadas puesto que permitirían incorporar mejoras en los procesos impactados en el rediseño, permitiendo un entorno colaborativo, alta participación de los usuarios en el proceso, incorporar automatizaciones en distintas etapas y anticipar el proceso de pruebas para disminuir iteraciones producto de corrección de defectos. Finalmente, las hipótesis seleccionadas para ser realizadas son:

H3, H5, H8, H10, H13, H15, H16, H18, H20 y H22.

3.8 Propuesta de Solución

Para lograr el propósito de este trabajo, se rediseña el proceso de Cadena de Valor del Servicio Compartido Desarrollo de Software, previamente señalados en la Figura 24. Dentro de la Cadena de Valor de Desarrollo de software, se trabajará en el rediseño de los procesos Administración de relación con el Usuario, Gestión de producción y entrega de software y, por último, Producción y Entrega de Software.

3.8.1 Rediseño del proceso Administración de relación con el Usuario

El rediseño de este proceso tiene como principal objetivo incorporar al usuario como parte de la solución tecnológica, haciéndolo participe de todo el proceso de la construcción del producto digital, desde la etapa de captura de necesidades estratégicas que luego serán transformadas en un Backlog de Portfolios y posteriormente será dividido en iniciativas llamadas Épicas. Las Épicas tienen como responsable a un nuevo rol llamado Epic Owner y tiene como objetivo descomponer el Portfolio en entregas de valor continuas e incrementales. Otro cambio importante que ocurre en este proceso es que quedan obsoletos los proyectos como se conocían originalmente y se comienza a trabajar bajo el concepto de Flujo de Valor a través iniciativas llamadas épicas. Las Épicas luego serán descompuestas en iniciativas más pequeñas llamadas Historias de usuario. Algunas de las grandes diferencias importantes de destacar:

Tabla 5: Épica Vs Proyecto

Épica	Proyecto
Equipo de trabajo estable en el tiempo, continúa trabajando en el valor continuo.	Equipos de trabajo temporal, se desmantela una vez terminado el proyecto.
Objetivos variables y ajustables en el tiempo en busca de entregar valor.	Objetivos fijos con un inicio y fin definido, todo es implementado sin importar si entrega valor.
Progreso medido según el beneficio entregado.	Progreso medido en tareas terminadas.

Para realizar este rediseño se trabaja directamente en el proceso *Administración Relación con Usuario* en donde el principal cambio, es que el usuario comienza a ser parte del proceso de construcción de soluciones tecnológicas, tomando un rol protagónico para lograr el objetivo de acoplar la estrategia de la compañía con el Portfolio de soluciones. Con esta unión se crea el concepto de Value Stream que permite transformar las necesidades estratégicas en soluciones.

Una vez implementado el rediseño será posible distinguir las siguientes competencias:

- Un Portfolio alineado con la estrategia para resolver los problemas de los clientes.
- Una clara visibilidad de los Values Stream y sus respectivas soluciones.
- Un gobierno basado en la agilidad empresarial.

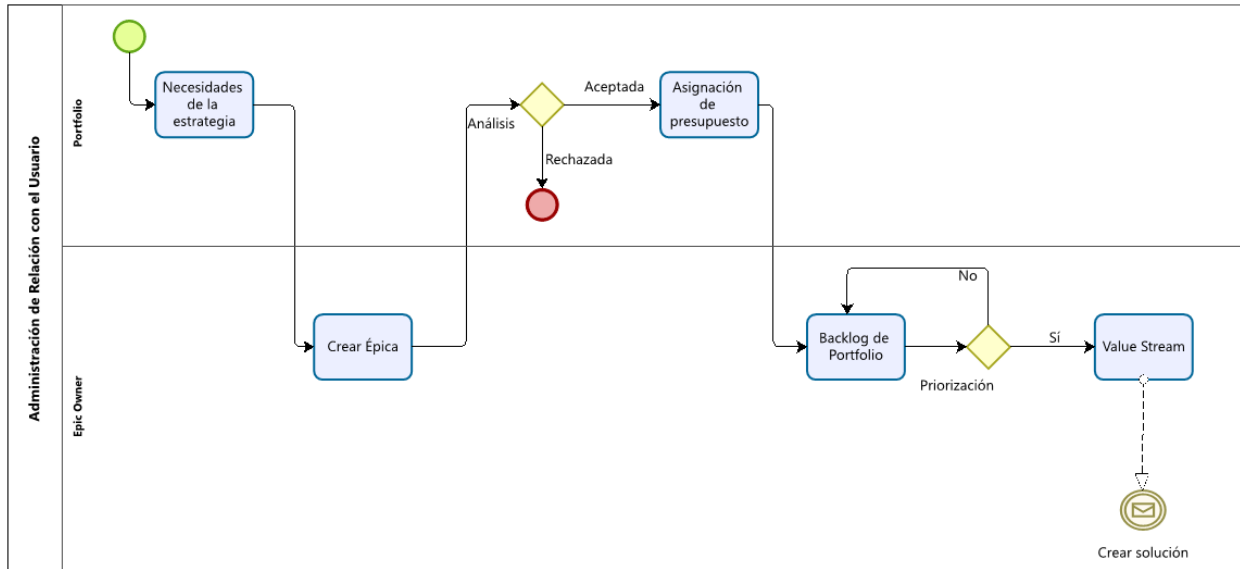


Figura 26: Proceso Administración de relación con el usuario
Fuente: (Elaboración propia)

Tabla 6: Matriz de proceso Administración de relación con el usuario

Nro.	Qué	Quién	Cuando	Cómo se hace
1	Necesidades estratégicas	Portfolio Management	Al definir la estrategia	La alta dirección y los stakeholders definen la estrategia comercial de la compañía
2	Crear Épica	Epic Owner	Con las necesidades estratégicas	Con la estrategia definida es posible tener un contexto claro para definir las iniciativas y con ellas lograr los objetivos claves
3	Asignación de presupuesto	Portfolio Management	Luego de aceptada la Épica	Existe un presupuesto para definido para financiar los habilitadores de la estrategia, una vez

				aceptada la Épica se le asigna un presupuesto para cada flujo de valor
4	Agregar al Backlog	Epic Owner	Una vez el presupuesto sea asignado	Una vez las Épicas han sido aprobadas son agregadas al backlog en donde serán dimensionadas, clasificadas y priorizadas para su implementación
5	Value Stream	Epic Ower	Cuando la Épica sea seleccionada para su desarrollo	Una vez comienza el desarrollo de la épica, sigue una secuencia actividades para convertir una necesidad estratégica en una solución

3.8.2 Rediseños Gestión de producción y entrega de Servicios TI

El rediseño de este proceso tiene como objetivo principal, incorporar las prácticas del marco metodológico SAFe y principios de DevOps. Aplicando estas prácticas y principios en los distintos equipos que participan del Servicio Compartido Servicios IT, es posible eliminar los objetivos antagónicos de cada uno de estos equipos, permitiendo así, la colaboración y sinergia para realizar entregas más rápidas de soluciones TI y con mayor calidad.

Es preciso recordar que el modelo de trabajo tradicional, sumado a los objetivos contrarios de los distintos equipos tecnológicos, generan un escenario que no logra satisfacer las necesidades de los negocios del banco ni los objetivos comerciales estratégicos.

Luego del rediseño del proceso será posible distinguir las siguientes capacidades:

- Desarrollo de soluciones TI alineados con los objetivos comerciales estratégicos.
- Programas divididos en requerimientos pequeños y específicos que faciliten la construcción y la entrega de valor de manera continua.

- Mejor organización de las personas para promover el enfoque hacia generar y entregar valor de manera constante.
- Un entorno de trabajo que fomente la colaboración, la innovación y la mejora continua por parte de los equipos de trabajo.

En la figura, se representa gráficamente el rediseño del proceso Gestión Producción y Entrega de Servicios TI:

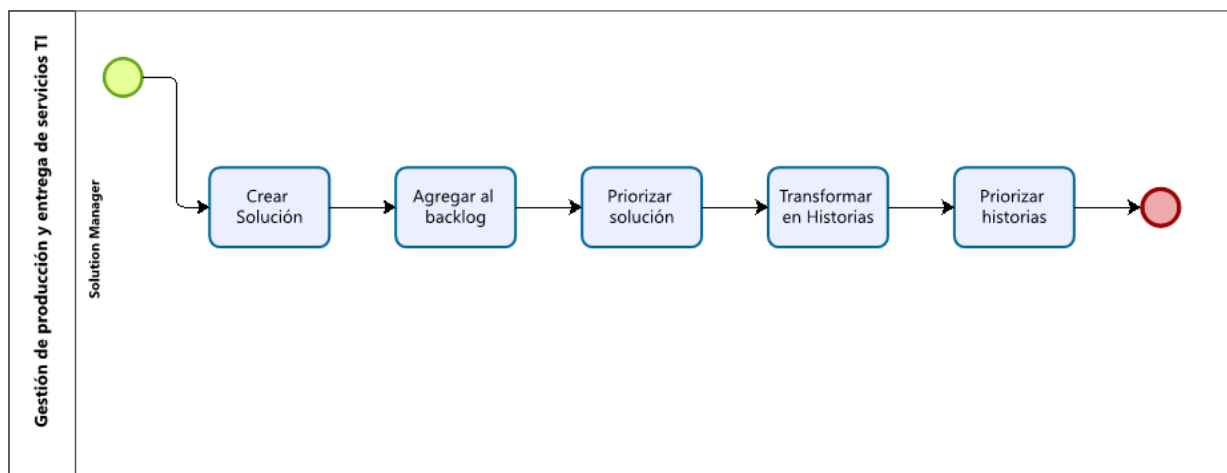


Figura 27: Proceso Gestión Producción y Entrega de Servicios TI
Fuente: (Elaboración propia)

En la siguiente Tabla se describe el proceso Gestión Producción y entrega de Servicios TI:

Tabla 7. Matriz de proceso Gestión Producción y Entrega de Servicios TI

Nro.	Qué	Quién	Cuando	Cómo se hace
1	Crear Solución	Solution Manager	Recibida la solicitud de solución del proceso anterior	Se debe definir exactamente lo que se construirá, se debe definir claramente la intención de la solución y cómo se construirá
2	Agregar al backlog	Solution Manager	Definida la solución	Una vez la solución ha sido creada es agregada al backlog en donde serán dimensionadas, clasificadas.

3	Priorizar solución	Solution Manager	Luego de ser clasificadas	Luego de ser clasificada, la solución debe ser priorizada para comenzar a trabajar en el desarrollo de la solución
4	Transformar en Historia	Solution Manager	Una vez priorizada	La solución es priorizada para comenzar el desarrollo y se debe descomponer en historias y se arma el roadmap de la solución
5	Priorizar historias	Solution Manager	Creadas las historias	Definidas las historias de usuario son entregadas al siguiente proceso en donde se realizará la construcción de los MVP de manera incremental y continua.

3.8.3 Rediseño del proceso Producción y entrega de Software

El rediseño de este proceso tiene como objetivo principal incorporar prácticas y herramientas que permitan agilizar la construcción de soluciones tecnológicas, permitiendo realizar entregas continuas e incrementales para así poder llegar antes al mercado con los nuevos productos digitales y sus mejoras.

Los requerimientos generados por los usuarios llegan a los diversos equipos de construcción de soluciones tecnológicas para comenzar el largo proceso desde el análisis de los requerimientos, pasando por el desarrollo y las pruebas, hasta llegar a la producción. Según lo revisado en el punto anterior, todo este proceso es burocrático y manual.

Para realizar este rediseño, se trabaja directamente en los dos procesos que son parte de Producción y Entrega de Software, los cuales son

Al proponer un nuevo proceso de Producción y Entrega de Software que incorpore métodos ágiles y herramientas para la automatización de las tareas manuales, que

posteriormente generan retrabajo, luego del rediseño se dispondrá de las siguientes capacidades:

- Tener un proceso de despliegue automatizado para promover el software a los distintos ambientes.
- Anticipar la etapa pruebas, incorporándolas desde el inicio del desarrollo y en todas las etapas hasta llegar a producción. Esto trae por consiguiente la disminución de los defectos reportados en las etapas de Preproducción y producción
- Generar un repositorio centralizado de versionado de código de software para asegurar que siempre se está trabajando con la última versión.
- Se dispondrá de un proceso ágil, con mayor velocidad y que además aumenta la calidad.

En la figura, se representa gráficamente el rediseño del proceso producción y entrega de software:

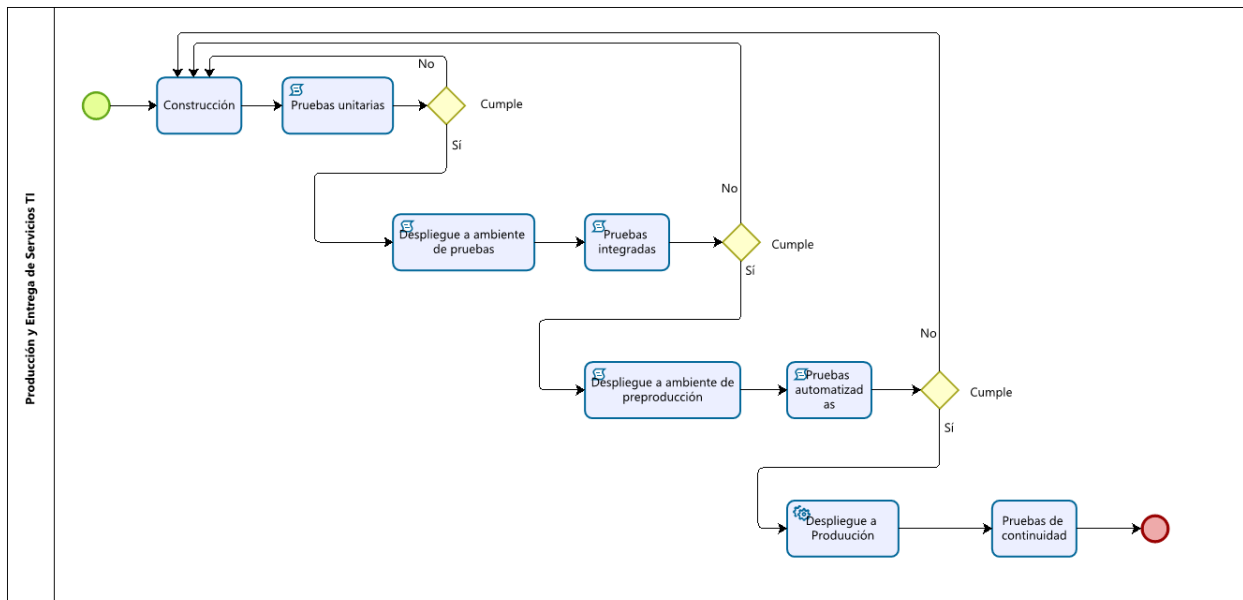


Figura 28: Proceso Producción y entrega de software
Fuente: (Elaboración propia)

En la siguiente Tabla se describe el proceso producción y entrega de software

Tabla 8. Matriz de proceso de producción y entrega de Software

Nro.	Qué	Quién	Cuando	Cómo se hace
1	Construcción	Célula de trabajo	Al iniciar un nuevo sprint	En la ceremonia de Sprint Planing se priorizan los requerimientos en conjunto con el Product Owner que se van a atender en sprint.
2	Pruebas unitarias	Desarrollador	Luego de escribir el código de desarrollo	Desde Jenkins, la herramienta que orquesta los despliegues a los distintos ambientes. Jenkins recibe la instrucción y ejecuta los scripts previamente creados para las pruebas unitarias
3	Despliegue a Ambiente de pruebas	Desarrollador	Luego de las pruebas unitarias	Desde Jenkins, la herramienta valida que el resultado de las pruebas unitarias ha sido satisfactorio y gatilla el despliegue de los componentes al ambiente de pruebas.
4	Pruebas integradas Automatizadas	Ingeniero de testing	Luego del despliegue a ambientes de test	Luego de realizado el despliegue al ambiente de test, Jenkins ejecuta las pruebas integradas automatizadas en dicho ambiente
5	Despliegue a ambiente de preproducción	Desarrollador	Luego de las pruebas integradas	Jenkins valida que el resultado de las pruebas integradas ha sido satisfactorio y ejecuta el despliegue de los componentes al

				ambiente preproductivo.
6.1	Pruebas de regresión automatizadas	Ingeniero de testing	Luego del correcto despliegue al Preproducción	Una vez realizado el despliegue al ambiente de preproducción, Jenkins ejecuta del set de pruebas de regresión automatizadas
6.2	Pruebas incrementales manuales	Ingeniero de Testing	Al finalizar pruebas automatizadas	El Ingeniero de Testing ejecuta las pruebas incrementales manuales para completar el flujo de testing
7	Despliegue a producción	Operador de despliegue	Resultado OK de las pruebas	El Operador de despliegue ejecuta manualmente el proceso automatizado de despliegue a producción
8	Pruebas de continuidad	Ingeniero de Testing	Al recibir la confirmación del operador de despliegue	El Ingeniero de testing realiza un set de pruebas de continuidad en el ambiente de producción para asegurar el correcto funcionamiento de las aplicaciones impactadas.

3.8.4 Conclusión del rediseño

Al mismo tiempo que ocurren los rediseños mencionado en los puntos anteriores, esta propuesta de solución incorpora una transformación del actual ciclo de vida del desarrollo de software, que va desde un modelo de desarrollo en cascada hacia un modelo de DevOps, pasando por sus etapas intermedias.

La evolución apunta a realizar cambios de software cada vez más pequeños y de manera constante que permitan obtener feedback y mejora continua.

Esta medida permite la participación constante de todos los involucrados en el ciclo de desarrollo de software, de forma que cada defecto o indefinición puede ser corregida prácticamente en el momento. Esta evolución se explica visualmente en la figura 29.

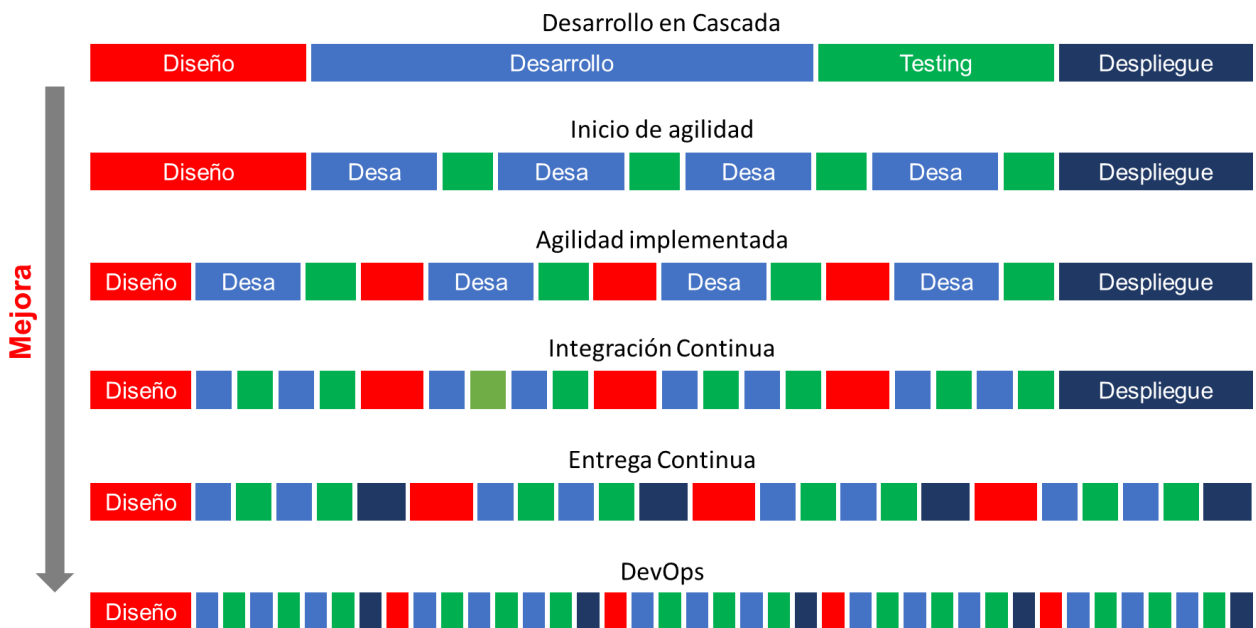


Figura 29: Evolución del flujo de despliegue de Software
Fuente: (Elaboración propia)

3.9 Plan de Implementación y Acción

Para la implementación de los rediseños de procesos se trabajará en los pilares básicos de DevOps. Tal como se hizo referencia en el Capítulo 2, DevOps es una forma distinta de pensar, un conjunto de prácticas y herramientas que entregan integración, automatización y una participación cercana entre todos los equipos que son parte en el desarrollo de una solución digital.

Para realizar este cambio cultural se utilizará el enfoque CALMS (Culture, Automation, Lean, Measurement y Sharing) que incorpora estos cinco elementos de excelencia que actúan como guía para conseguir la entrega continua de valor.

3.9.1 Gestión del cambio

Tal como se ha visto en capítulos anteriores, DevOps es la combinación de filosofías, prácticas y herramientas culturales en donde la cultura es el núcleo de las prácticas DevOps y la piedra angular es la automatización. Dicho lo anterior, este proyecto conlleva un cambio significativo desde el punto de vista cultural de los trabajadores de Banco SCL, esto viene dado, especialmente, por la forma en que se ha trabajado durante tantos años y de cómo se relacionan las distintas áreas entre ellas debe cambiar para que el proyecto sea exitoso.

Peter Drucker dijo: “La cultura se come como desayuno a la estrategia”, esta frase resume perfectamente la trascendencia de la cultura y que un trabajo de este nivel no se puede considerar como trivial. Es por ello que se propone realizar un proceso continuo de gestión del cambio, que permita incorporar de manera adecuada el cambio de mentalidad colaborativa y las nuevas prácticas.

Para saber si el proceso de gestión de cambio está avanzando de la manera deseada se propone incorporar un modelo de madurez de adherencia que permita reflejar la internalización de las prácticas en los distintos equipos. La etapa de medición de adherencia a los cambios es transcendental para enfocar las mejoras que se desean y para que los equipos avancen de manera uniforme en el proceso de cambio.

Para poder enfocar el proceso de gestión de cambio y su adherencia, se debe identificar claramente cuáles son los roles y áreas que se verán afectadas por las distintas prácticas que deben incorporar. En primera instancia resolver esta pregunta es relativamente sencillo ya que son todas las áreas que participan en el proceso de entrega de soluciones tecnológicas.

Finalmente, no sólo se trata de aplicar cambios, también es importante identificar y resaltar todas las prácticas que se debe conservar, como, por ejemplo, la proactividad de las personas para hacer que las cosas sucedan.

Para lograr este proceso de transformación, los equipos deben incorporar nuevas prácticas, procesos y herramientas, para ello es necesario crear un plan de capacitación a las personas de cada uno de los equipos que participan en el proceso de construcción de software, este plan debe ser transversal a todos los equipos y debe permitir entregar

los conocimientos de manera homogénea e incremental. Dicho plan se explica a continuación:

El plan propuesto para la implementación del rediseño de procesos consta de dos años, en este periodo de tiempo se realizará un proceso incremental de implementación y capacitación, para ello se realizarán Olas de trabajo en donde participarán grupos de personas de cada una de las gerencias. Esta selección de equipos para las Olas de trabajo tendrá como objetivo enfocar los esfuerzos del cambio en aquellos equipos que más impacto tendrían en el flujo continuo de entrega de valor.

Tabla 9: Plan de implementación

	Y1				Y2			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Gestión del cambio								
Capacitación Fundamentos DevOps								
Selección Herramientas								
Implementación herramientas								
Capacitación Orquestador								
Capacitación Versionado								
Capacitación Unit Test								
Capacitación Automatización								
Medición								

Fiel a las metodologías ágiles presentadas en este trabajo, este plan considera realizar el proceso de gestión de cambio en entregas tempranas considerando en primera instancia a los equipos que desarrollan APP Móviles, pues la tecnología utilizada para desarrollar APP Móviles es de las que más se adapta para implementar el rediseño. Por otra parte, la estrategia de comenzar con un grupo permite un proceso de aprendizaje y mejora continua para incorporar en las siguientes Olas de trabajo.

Como segundo grupo para participar en la transformación se consideran los equipos que desarrollan aplicaciones Web para clientes. En el tercer grupo se consideran aplicaciones Web para usuarios internos de atención a clientes y como cuarto grupo se consideran todos los equipos que desarrollan aplicaciones internas de Banco.

Finalmente, para medir la adherencia de adopción al cambio, se creará un modelo de madurez que considera cinco niveles y permite que los equipos puedan ir avanzando en dichos niveles dependiendo de las prácticas y herramientas que han sido incorporadas en su proceso de construcción de software. Se realizará una encuesta trimestral en donde cada uno de los equipos deberá responder si ha adoptado las prácticas DevOps o aún está en proceso de adopción. En base resultados obtenidos por dicha encuesta se podrá determinar el nivel de madurez de cada uno de los equipos.

En la tabla 10 se explica la estructura que tendrá la encuesta del modelo de adopción de prácticas DevOps para los equipos de desarrollo de software. Por cada una de las practicas necesarias para la implementación de DevOps se realizará un numero definido de preguntas para cada uno de los cinco niveles que propone el modelo de adopción. Cada equipo irá superando dichos niveles en cuanto sus respuestas sean positivas en su totalidad por cada nivel.

Tabla 10: Estructura Encuesta del Modelo de Madurez

Modelo de Adopción					
Prácticas	Nivel 1	Nivel 2	Nivel 3	Nivel 4	Nivel 5
Cultura	3	3	2	1	1
Desarrollo	3	3	2	2	2
Pruebas	3	3	2	2	1
Despliegue	3	2	2	1	1

En la siguiente tabla de describen las preguntas del modelo de adopción:

Tabla 11: Preguntas del modelo de adopción:

Nro	Práctica	Nivel	Pregunta
1	Cultura	1	¿Todos los miembros del equipo conocen el proceso de desarrollo DevOps?
2	Cultura	1	¿El equipo realiza alguna reunión de feedback periódica y constante en el tiempo?
3	Cultura	1	¿El equipo posee un entorno seguro y de confianza en donde expresar sus ideas y puntos de vista?
4	Cultura	2	¿El equipo está conformado como un equipo multifuncional?
5	Cultura	2	¿El equipo cuenta con el resultado de las reuniones de feedback en el tiempo?
7	Cultura	2	¿El equipo es autónomo para tomar decisiones sobre cómo de construir el producto?
7	Cultura	3	¿El equipo es autónomo para decidir cuándo realizar la entrega de software a producción?
8	Cultura	3	¿El equipo siempre está proponiendo ideas nuevas de cómo hacer las cosas?
9	Cultura	4	¿Los líderes del equipo permiten el concepto de fallar rápido?
10	Cultura	5	¿Dentro del equipo se patrocina la experimentación y la innovación?








11	Desarrollo de Software	1	¿El equipo conoce y aplica los patrones de diseño existentes?
12	Desarrollo de software	1	¿Por cada defecto que se detecta es posible identificar la línea de código rápidamente?
13	Desarrollo de software	1	¿Cualquier miembro del equipo puede identificar la línea de código y corregir el defecto?
14	Desarrollo de software	2	¿Se reutilizan las soluciones ya existentes de manera ahorrar tiempo?
15	Desarrollo de software	2	¿El equipo puede rechazar tarea de Bajo Valor y Alto Esfuerzo en la planificación?
16	Desarrollo de software	2	¿El Equipo revisa las funciones con el Product Owner más de una vez durante el sprint?
17	Desarrollo de software	3	¿Por cada nueva funcionalidad el equipo construye las pruebas unitarias?
18	Desarrollo de software	3	¿El equipo tiene la práctica de identificar y eliminar el código muerto?
19	Desarrollo de software	4	¿El equipo monitorea los tiempos de las tareas de desarrollo?
20	Desarrollo de software	4	¿El equipo mejora los tiempos en cada fase?
21	Desarrollo de software	5	¿El equipo investiga nuevas tecnologías / evoluciones para mejorar las soluciones?
22	Desarrollo de software	5	¿El equipo "hace experimentos" para validar y medir nuevas tecnologías?
23	Pruebas	1	¿La estrategia de prueba la definen y la aplican los equipos?
24	Pruebas	1	¿Hay un plan de pruebas definido y cubre todas las funciones del producto?
25	Pruebas	1	¿Las pruebas son consistentes, no permiten ambigüedad de resultados y son repetibles?
26	Pruebas	2	¿El seguimiento de defectos incluye versión de origen y versión de corrección?
27	Pruebas	2	¿Los desarrolladores y probadores trabajan en estrecha relación?
28	Pruebas	2	¿La estrategia de pruebas concuerda con la pirámide de pruebas?
29	Pruebas	3	¿Las pruebas de regresión son automatizadas y actualizadas de manera constante?
30	Pruebas	3	¿Las pruebas automatizadas se ejecutan en cada liberación?
31	Pruebas	4	¿Las pruebas automatizadas son parte del proceso de entrega continua?
32	Pruebas	4	¿El equipo puede generar datos de prueba validos de manera autónoma?
33	Pruebas	5	¿Las pruebas se crean y automatizan utilizando prácticas de BDD?
34	Despliegue	1	¿Existe un modelo de gestión de versiones de código fuente definido?

35	Despliegue	1	¿El equipo posee un proceso de entrega con un procedimiento determinado?
36	Despliegue	1	¿Se realiza el despliegue del mismo producto en todos los entornos?
37	Despliegue	2	¿El proceso de entrega de software es auditable y sin espacios de tiempos muertos?
38	Despliegue	2	¿El proceso de entrega de software se realiza sobre la plataforma definida?
39	Despliegue	3	¿El proceso de implementación esta automatizado para promover software entre los entornos?
40	Despliegue	3	¿Las pruebas de calidad son validadas automáticamente en el proceso de despliegue?
41	Despliegue	4	¿El despliegue posee un proceso de mejora continua, respaldado por indicadores?
42	Despliegue	5	¿El equipo es autónomo a la hora de desplegar el software en producción

CAPÍTULO 4: IMPLEMENTACIÓN APLICADA

En el universo de DevOps, existe una variada oferta de distintas herramientas, cada una de ellas especializada para las distintas etapas de construcción de software. Luego de realizar el trabajo de investigación para identificar las herramientas que mejor se adaptan a las necesidades de las tecnologías que actualmente existen y considerando las nuevas tecnologías que deben implementar en el proceso de construcción de software, se estableció el siguiente listado de herramientas:

Tabla 12: Descripción de herramientas para la implementación

Herramienta	Descripción
 Jenkins	Es la herramienta Open Source encargada de orquestar todo el proceso automatizado. Jenkins realiza la ejecución de compilación, instalación, despliegue y testing.
 GitLab	Esta herramienta permite realizar el control de versiones de código fuente en un entorno colaborativo basado en realizar copias del repositorio de código fuente.
 Nexus	Nexus es un repositorio de artefactos, en esta herramienta se almacenan los objetos binarios que utiliza Jenkins para realizar la compilación.
 Protractor	Protractor es una herramienta para el desarrollo de pruebas End to End de aplicaciones Web basadas en Angular.
 Jasmine	Esta herramienta permite implementar pruebas unitarias en Angular.
 Cucumber	Cucumber es una herramienta de software que permite el desarrollo de pruebas automatizadas basado en el comportamiento (BDD)
 Gherkin	Es un pseudo lenguaje de programación utilizado para describir el comportamiento del software en lenguaje natural

Para realizar este piloto de implementación, lo primero fue enfocar la tecnología en el que será aplicado, en este caso se utilizará para las aplicaciones Web y App móviles desarrolladas en Angular.

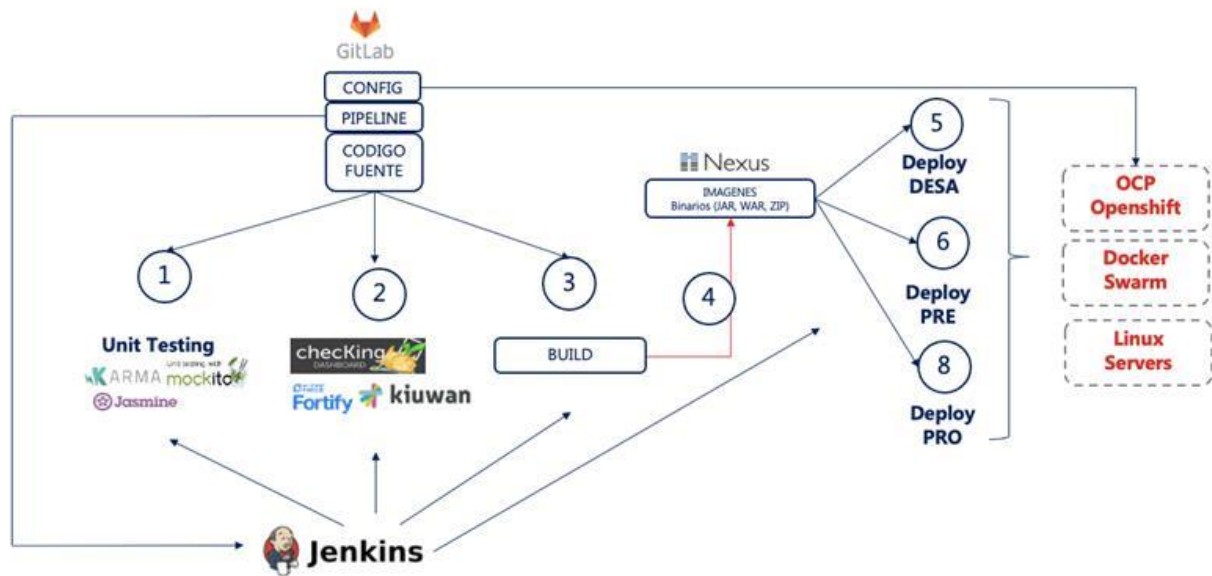


Figura 30. Modelo de implementación
Fuente: (Elaboración propia)

4.1 Adopción de GitLab

Una etapa clave para implementación del nuevo modelo de desarrollo de software es la adopción de GitLab como repositorio de código, esta herramienta es la encargada de gestionar las distintas versiones código en las que se está trabajando. Una de sus principales características es que permite el trabajo colaborativo, es decir varios desarrolladores pueden estar trabajando en distintas versiones del código utilizando el concepto de ramas, que finalmente son distintas copias del repositorio de código fuente que luego se fusionan en la rama principal. Existen varios métodos para gestionar estas ramas de código fuente, en este caso se utilizará GitFlow.

La implementación de esta herramienta y el modelo de versionamiento Gitflow permite solucionar directamente los problemas descubiertos en el diagnóstico de la situación actual, en donde se detectó que no siempre se contaba con la última versión de código fuente al momento de comenzar un desarrollo, lo que finalmente provocaba que errores que ya habían sido corregidos volvieran a aparecer en producción.

4.2 Gitflow

Es el modelo de gestión de código fuente recomendado para ser utilizado en un entorno DevOps, para ello se crean ramas designadas para funciones específicas definiendo en qué casos se deben utilizar cada una de ellas. A continuación, se explica el modelo.

4.2.1 Rama Main y Develop

Este flujo de trabajo posee dos ramas de trabajo en vez de una única rama, La rama Principal o Main almacena toda la historia de la publicación oficial del código fuente y la rama Desarrollo o Develop se utiliza para integrar las nuevas funciones en las que se está trabajando.

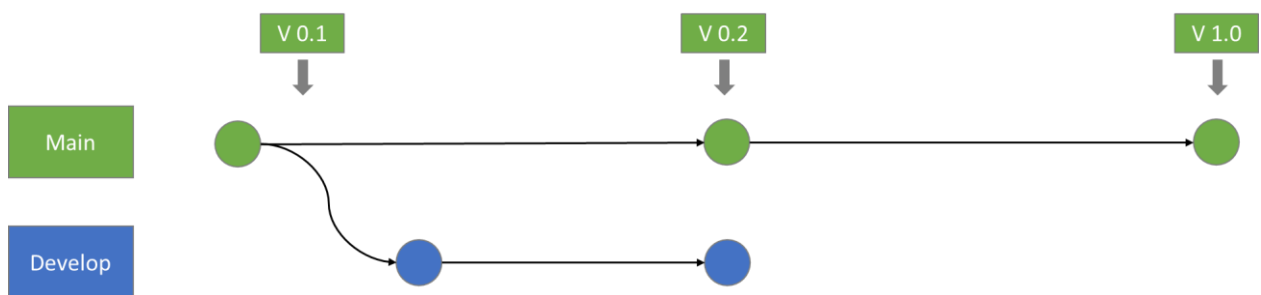


Figura 31. Rama Main
Fuente: (Elaboración propia)

4.2.2 Rama Feature

Se utiliza para trabajar en nuevas funcionalidades de la rama Develop, todas las nuevas funciones deben tener su propia rama asignada. Las ramas Feature se ramifican de la rama Develop y cuando la función está terminada, esta se fusiona de vuelta con la rama Develop.

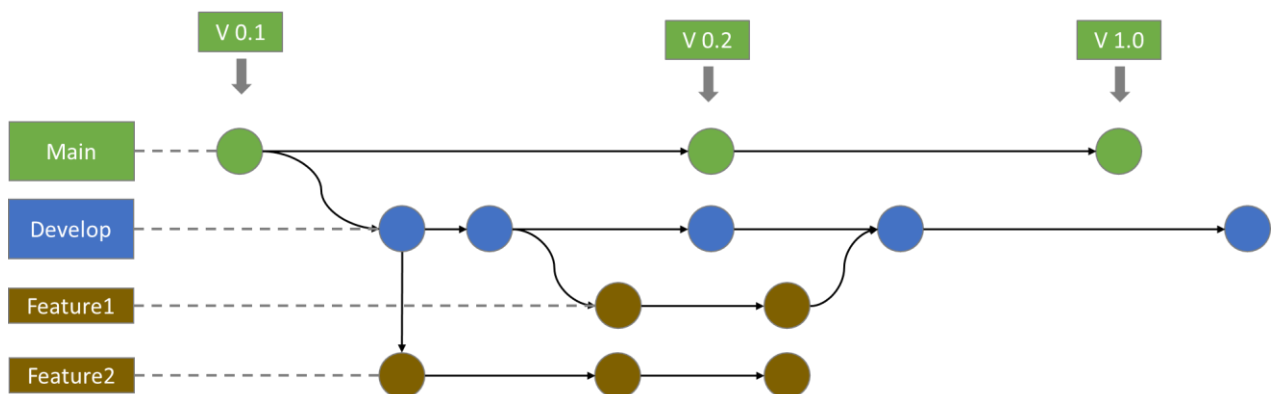


Figura 32. Rama Feature
Fuente: (Elaboración propia)

4.2.3 Rama Release

Una vez que la Rama Develop haya obtenido las funciones necesarias para realizar una publicación del software en producción, se debe crear la rama Release desde la rama Develop, con la rama Release creada se inicia el proceso de publicación a producción y dado esto no puede agregarse ninguna otra funcionalidad desde este punto. Sólo es posible agregar modificaciones para corregir algún error.

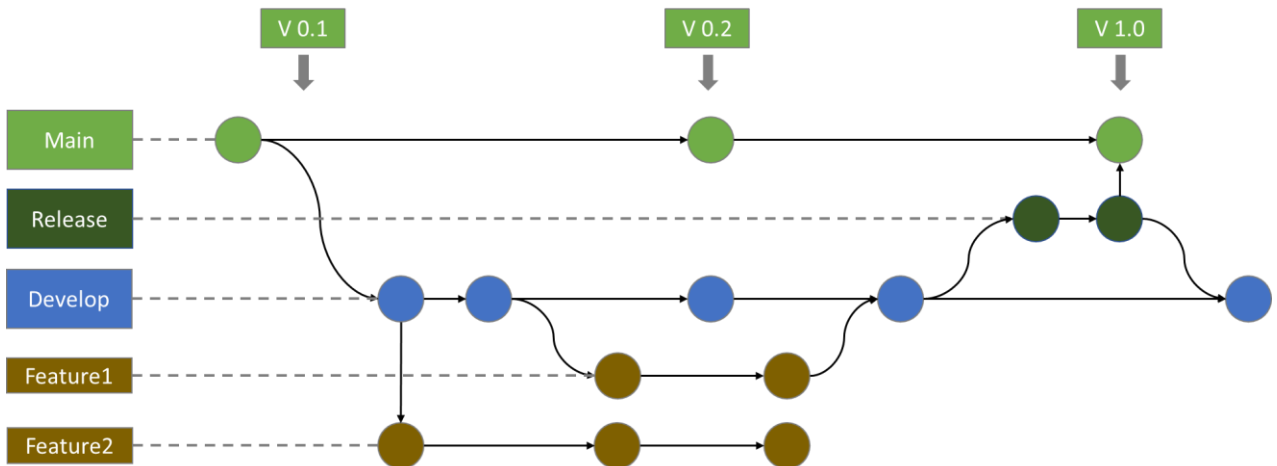


Figura 33. Rama Release
Fuente: (Elaboración propia)

4.2.4 Rama HotFix

La rama de corrección de Errores o Hotfix sirve para reparar un error descubierto en la producción. Esta rama a diferencia de las anteriores trabaja directamente sobre la rama Main. Cuando se termina la corrección, esta rama se fusiona con rama Main y con la rama Develop.

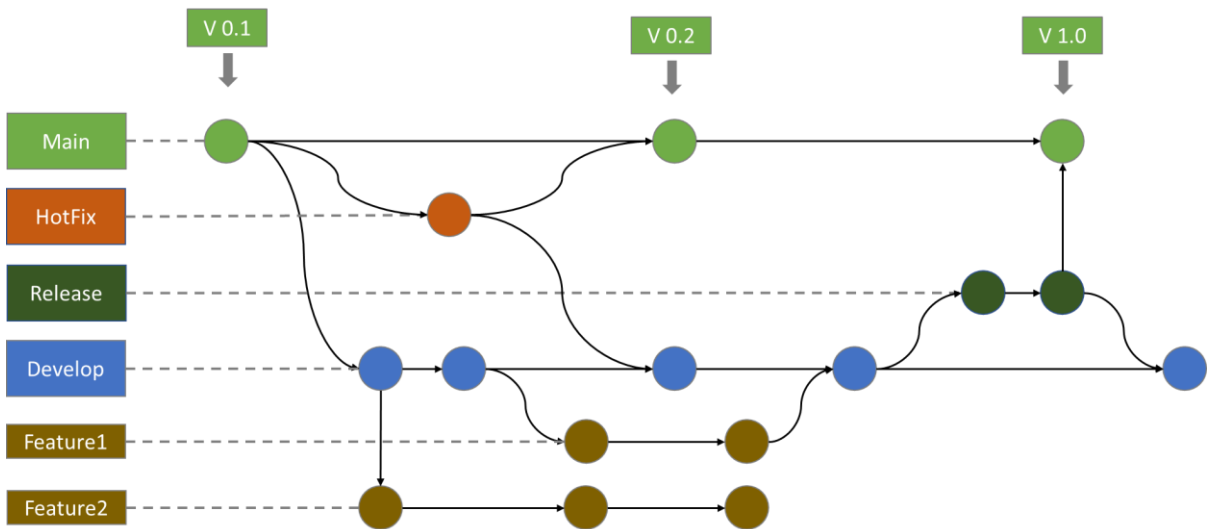


Figura 34. Rama HotFix.
Fuente: (Elaboración propia)

4.3 Despliegue automatizado con Jenkins

Otra pieza fundamental dentro del framework de solución propuesto, es la herramienta Jenkins, quien es el encargado de realizar la integración continua dentro del ciclo de desarrollo de software. Jenkins actúa como un orquestador de todo el modelo propuesto permitiendo la automatización de todo el ciclo. Realiza la compilación de las piezas de software, ejecuta el testing automatizado mediante pruebas unitarias, pruebas de integración y pruebas funcionales automatizadas, genera reportes de las ejecuciones y realiza el despliegue de las piezas software en los distintos ambientes.

Todas las tareas del proceso mencionado anteriormente son realizadas en cosas de minutos dada la integración de Jenkins con las otras herramientas del framework. Tal como se puede observar en las imágenes a continuación, en donde se ejecuta un despliegue, incluyendo todas las pruebas, al ambiente de desarrollo.

Esta implementación soluciona directamente los problemas detectados en el diagnóstico de la situación actual, dado que elimina la manualidad del proceso, además de anticipar las pruebas para evitar volumen de defectos en etapas cercanas a la producción.

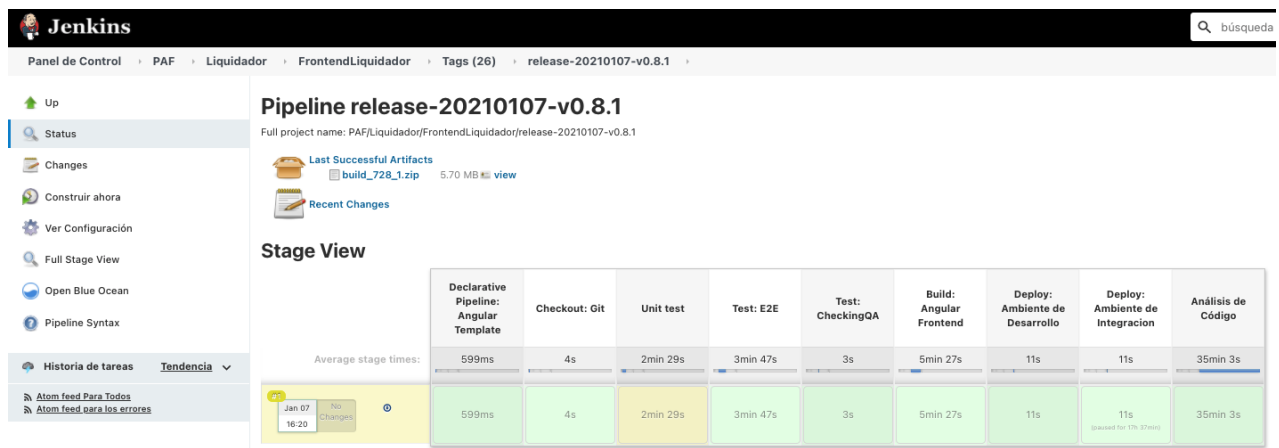


Figura 35. Ejecución de despliegue por Jenkins
Fuente: (Elaboración propia)

Declarative Pipeline: Angular Template	Checkout: Git	Unit test	Test: E2E	Test: CheckingQA	Build: Angular Frontend	Deploy: Ambiente de Desarrollo	Deploy: Ambiente de Integracion	Análisis de Código
599ms	4s	2min 29s	3min 47s	3s	5min 27s	11s	11s	35min 3s
599ms	4s	2min 29s	3min 47s	3s	5min 27s	11s	11s (paused for 17h 37min)	35min 3s

Figura 36. Tiempos de ejecución de despliegue
Fuente: (Elaboración propia)

4.4 Testing continuo

La necesidad de contar con este proceso se origina en el objetivo de evitar el retrabajo que significa detectar defectos en etapas cercanas al ambiente de producción, esto último viene dado por dos causas, primero porque la mayor cantidad de pruebas se realiza en etapas posteriores a la construcción del software y segundo por la escasa participación del usuario en el proceso.

El proceso de Testing continuo propone principalmente realizar las pruebas de forma temprana, idealmente desde el desarrollo, incorporando pruebas unitarias. Además, que puedan ser incrementales. También es necesario que se ejecuten con mayor frecuencia, cada vez que se agregue una nueva funcionalidad o se realice un despliegue a los distintos ambientes y finalmente, deben ser automatizadas ya sea para ser ejecutadas por el ingeniero de calidad o como parte del proceso de despliegue por Jenkins.

Para la implementación de este proceso se utilizará una estrategia de desarrollo basado en el comportamiento llamado BDD (*Behavior Driven Development*). Aunque no se trata de una técnica de pruebas como tal, tiene fuerte implicancia en mejorar el resultado del testing ya que incorpora un idioma común para equipos técnicos y no técnicos, de esta manera se logra mejorar la comunicación y colaboración entre los equipos de desarrollo, de pruebas y los usuarios.

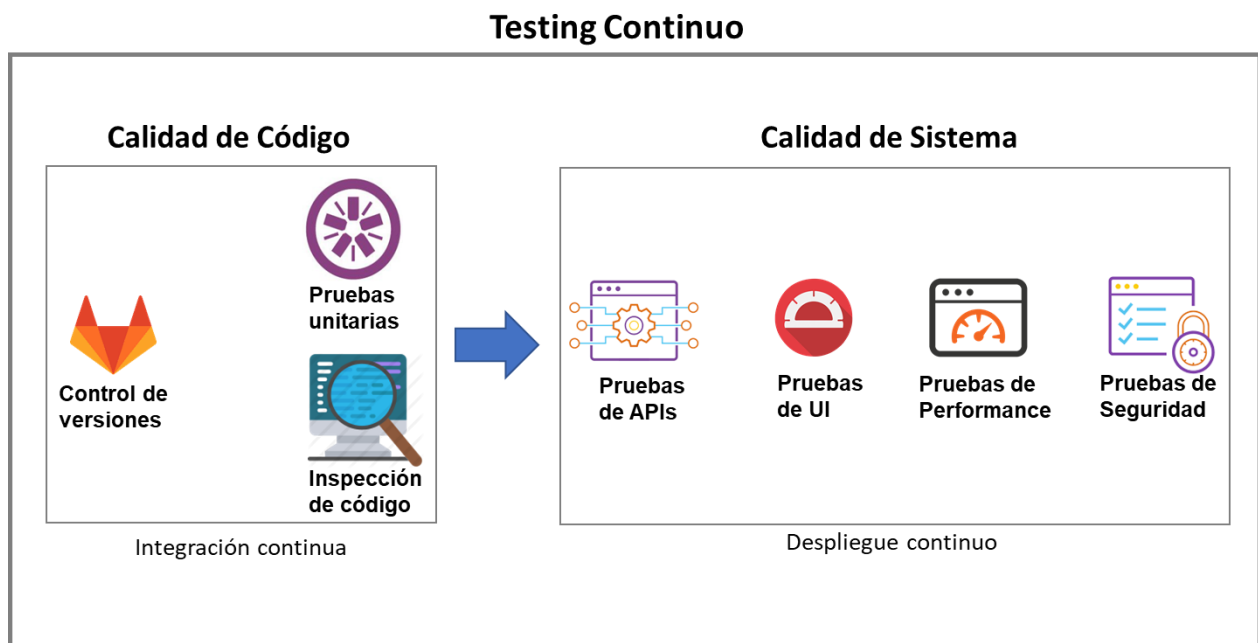


Figura 37. Esquema de Testing Continuo
Fuente: (Elaboración propia)

4.4.1 BDD (Behavior Driven Development)

Una de las grandes ventajas de BDD es que las definiciones se crean en un lenguaje común y las pruebas se comienzan a definir antes de comenzar el desarrollo. Una característica que hace más interesante usar BDD es que se basa en el comportamiento del software, en este sentido el equipo debe definir los detalles de cómo se deberá comportar el software que se va a desarrollar, de esta manera se hace mucho más comprensible para todos.

Como parte de este proyecto de implementación se incorpora BDD, utilizando un modelo de trabajo en donde existe una mayor participación por parte del usuario y todo el equipo participa de manera colaborativa tal como se aprecia en la siguiente figura.

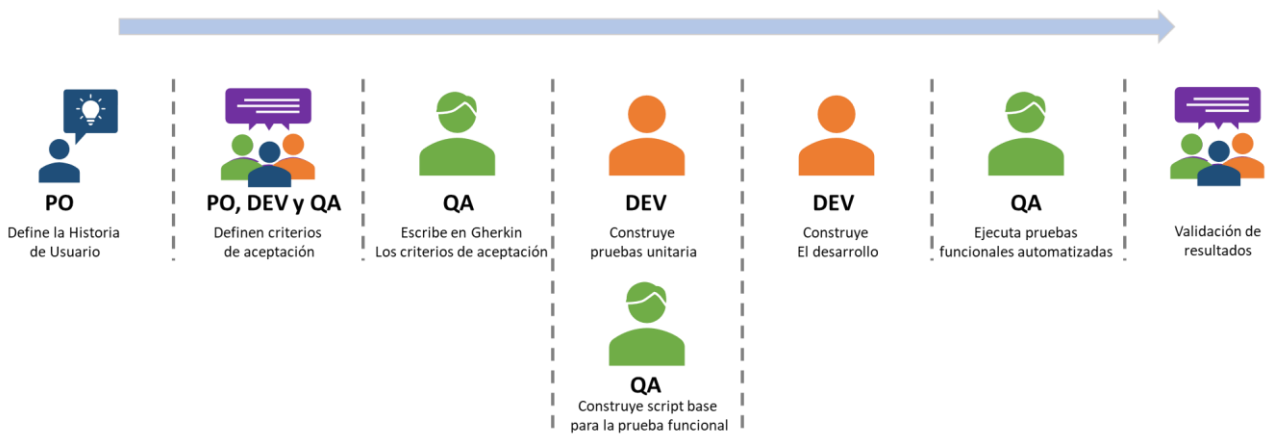


Figura 38. Flujo de trabajo BDD
Fuente: (Elaboración propia)

El desarrollo de los escenarios descritos en BDD ayudan al entendimiento de todo el equipo y en base a estas definiciones se deben construir las pruebas unitarias escritas por el desarrollador, así como también las pruebas funcionales automatizadas escritas por el ingeniero de QA.

Esta implementación soluciona directamente los problemas detectados en el diagnóstico de la situación actual, dado que incorpora al usuario en el proceso de construcción de software, evitando el retrabajo provocado por las ambigüedades en los requerimientos.

4.4.2 Cucumber y Gherkin

Para llevar a cabo el uso de BDD es necesario contar con herramientas que se utilizaran en la implementación de esta técnica, la más popular y la que utilizará en esta implementación es Cucumber que permite ejecutar las descripciones del comportamiento como pruebas funcionales automatizadas, Cucumber se basa en un lenguaje común llamado Gherkin.

Gherkin es un lenguaje común comprensible tanto por humano como por computadores con el que se describen las funcionalidades que definen el comportamiento del software. Se trata de un lenguaje fácil de entender, fácil de escribir y fácil de leer tal como se puede apreciar en el ejemplo a continuación.

```
LiqCourse.feature 1.49 KB
1 Feature: Funcionamiento del Curse
2
3 Como analista de operaciones financieras
4 Necesito una pantalla para poder identificar y activar distintos
5 flujos de operaciones, dependiendo de los estados y condiciones de los pagos
6 Para poder identificar estados, condiciones de las operaciones y acciones posibles
7
8 Background:
9   Given Obtengo acceso al Liquidador de Operaciones
10
11 Scenario: Apartado BUSCADOR DE OPERACIONES
12   When Ingreso valores en los campos del Buscador de Operaciones
13   Then Valido funcionalidad exitosa de los campos del Buscador de Operaciones
14
15 Scenario: Botón BÚSQUEDA DE PRODUCTOS
16   When Activo productos en la lista desplegable Búsqueda de Productos
17   Then Valido funcionalidad exitosa del botón Búsqueda de Productos
18
19 Scenario: Botón DESCARGAR EXCEL en Grilla de Operaciones
20   When Descargo Excel en múltiples páginas de la grilla
21   Then Valido funcionalidad exitosa del botón Descarga Excel de la grilla
22
23 Scenario: Apartado GRILLA DE OPERACIONES
24   When Navego en todas las funcionalidades de la Grilla de Operaciones
25   Then Valido funcionalidad exitosa de la Grilla de Operaciones
26
27 # WIP
28 # Scenario: Botón ENVIAR A CURSAR (No Duplicada)
29 #   When Navego en todas las funcionalidades del botón Enviar a Cursar
30 #   Then Valido funcionalidad exitosa del botón Enviar a Cursar
31
32 Scenario: Botón EDITAR OPERACIÓN (lápiz)
33   When Selecciono botón lápiz para modificar una operación
34   Then Valido funcionalidad exitosa del botón Editar Operación
```

Figura 39. Ejemplo de escenario escrito en Gherkin
Fuente: (Elaboración propia)

Luego de descritos los escenarios recién vistos, se escribe el paso a paso en mayor detalle utilizando el mismo lenguaje Gherkin. Este paso a paso servirá de guía para realizar las automatizaciones. En la siguiente imagen es posible ver como el escenario “Apartado BUSCADOR DE OPERACIONES” es descrito en su respectivo paso a paso.

```

1 Feature: Funcionamiento del Course
2
3 # INICIO HISTORIA DE USUARIO #
4 Como analista de operaciones financieras
5 Necesito una pantalla para poder identificar y activar distintos
6 flujos de operaciones, dependiendo de los estados y condiciones de los pagos
7 Para poder identificar estados, condiciones de las operaciones y acciones posibles
8 # FIN HISTORIA DE USUARIO #
9
10 Background:
11 Given Obtengo acceso al Liquidador de Operaciones
12
13 Scenario: Apartado BUSCADOR DE OPERACIONES
14 When Seleccione campo EMPRESA
15 And Ingreso rut inválido "16378972-3"
16 And Ingreso valor máximo "0123456789"
17 And Ingreso valor mínimo "k"
18 And Ingreso valor formato inválido "-*/**-*/++[*"
19 And Ingreso rut válido "97036000K"
20 Then Valido funcionalidad exitosa de campo EMPRESA
21
22 When Seleccione campo CANAL ENTRADA
23 And Seleccione en lista desplegable el valor "Produban"
24 Then Valido funcionalidad exitosa de lista CANAL ENTRADA
25
26 When Seleccione campo RUT CLIENTE
27 And Ingreso rut inválido "5656563-7"
28 And Ingreso valor mínimo "k"
29 And Ingreso valor formato inválido "-*/**-*/++[*"
30 And Ingreso rut válido con K "20901792-K"
31 And Ingreso rut válido sin K "8378972-7"
32 Then Valido funcionalidad exitosa de campo RUT CLIENTE
33
34 When Seleccione campo ID AAMM
35 And Ingreso valor inválido "3xt0D3Pru3b@?!"
36 And Ingreso valor válido numérico "56629"
37 Then Valido funcionalidad exitosa de campo ID AAMM
38
39 When Seleccione campo OP SISTEMA ORIGEN
40 And Ingreso valor inválido "3xt0D3Pru3b@?!"
41 And Ingreso valor válido numérico "12365"
42 Then Valido funcionalidad exitosa de campo OP SISTEMA ORIGEN
43
44 When Seleccione campo CANAL DE PAGO
45 And Seleccione en lista desplegable el valor "Banco Externo"
46 Then Valido funcionalidad exitosa de lista CANAL DE PAGO
47
48 When Seleccione botón BÚSQUEDA DE PRODUCTOS
49 And Activo en lista desplegable el checkBox del producto "Depositos"
50 Then Valido funcionalidad exitosa botón BÚSQUEDA DE PRODUCTOS
51

```

Figura 40. Ejemplo de Gherkin
Fuente: (Elaboración propia)

Finalmente se escribirá el código de la prueba automatizada que utilizará como entrada todo lo descrito en los escenarios anteriores.

4.4.3 Pruebas Unitarias

Esta práctica se realiza en la etapa de desarrollo y es responsabilidad del desarrollador, es una de las más importante con respecto a disminuir el tiempo de clico de desarrollo de software puesto que aquí es donde se deben comenzar a realizar las primeras pruebas anticipando considerablemente la detección de defectos y facilitando la corrección de estos.

Las pruebas unitarias, tal como su nombre lo indica, aíslan una parte del código para probarlo de manera unitaria, de esta forma es posible asegurar que una funcionalidad en particular funciona correctamente y en caso de detectar algún defecto, el desarrollador tiene la posibilidad de corregirlo en el instante, de esta manera es posible evitar la detección de este defecto en la etapa de preproducción.

Una vez implementada la herramienta Jasmin, que permite realizar pruebas unitarias para Angular, es posible obtener resultado de ejecución indicando los errores que deben ser corregidos por el programador según se muestra en la siguiente imagen.

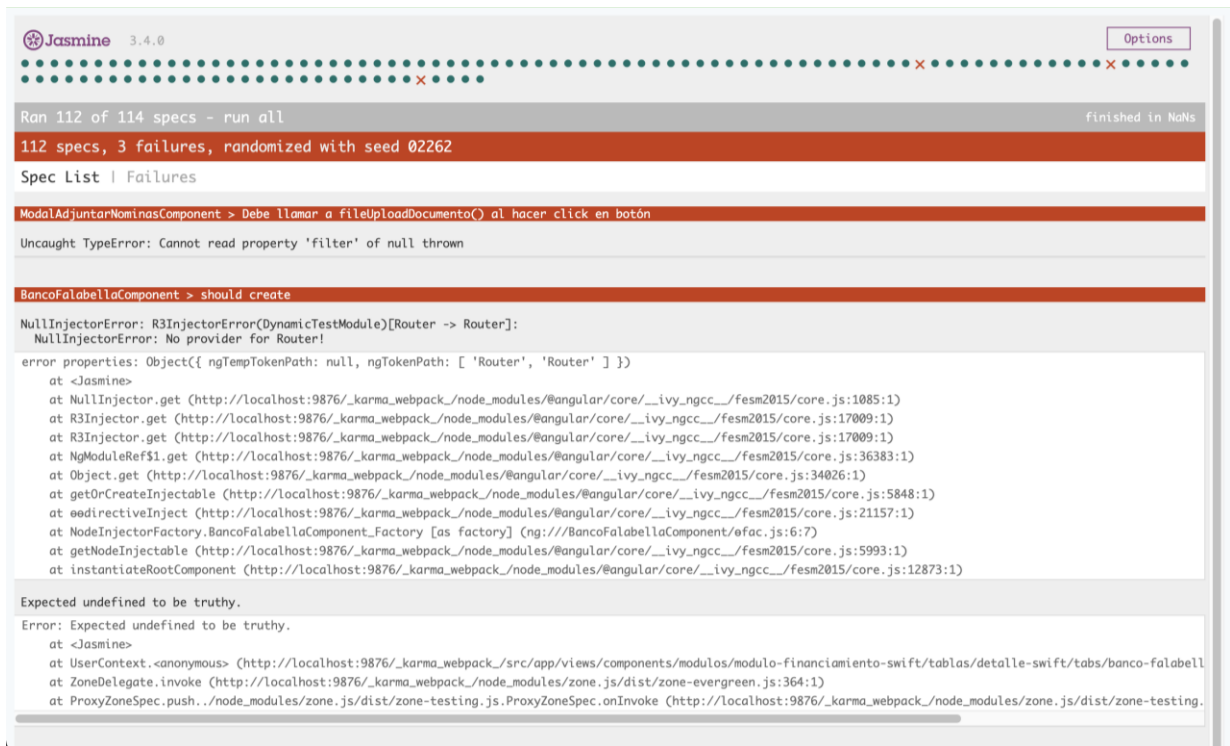


Figura 41. Ejemplo de defectos detectados en Pruebas Unitarias
Fuente: (Elaboración propia)

4.4.4 Pruebas Automatizadas

Como parte del testing continuo se encuentran las pruebas funcionales automatizadas, estas tienen la finalidad de acelerar el proceso de pruebas para mejorar los tiempos de lanzamiento del nuevo software al mercado, además proveen una mayor cobertura de pruebas del nuevo software en un menor tiempo.

Las pruebas automatizadas son incrementales, esto quiere decir que se crean nuevas pruebas conforme las nuevas funcionalidades del software en desarrollo se van implementando, este no es el único criterio que se utiliza, también es necesario identificar los escenarios críticos para la aplicación y con esto pueden aparecer nuevas pruebas.

Como parte de esta implementación se incorporan pruebas automatizadas de APIs y pruebas automatizadas funcionales, las que forman parte del proceso de despliegue y se podrán ejecutar mediante Jenkins en cada iteración de software. Es importante mencionar que las pruebas automatizadas funcionales son incrementales conforme se avanza en los distintos ambientes, vale decir, una porción de ellas es posible de ejecutar en un despliegue a desarrollo, una porción mayor es posible ejecutar en ambiente de pruebas y finalmente es posible ejecutar la totalidad en ambiente pre-productivo.

El resultado de la ejecución terminará con un informe visual y navegable para cada una de las funcionalidades ejecutadas, permitiendo identificar de forma clara y precisa los posibles defectos detectados en las ejecuciones del set de pruebas, tal como se puede apreciar en la siguiente imagen.

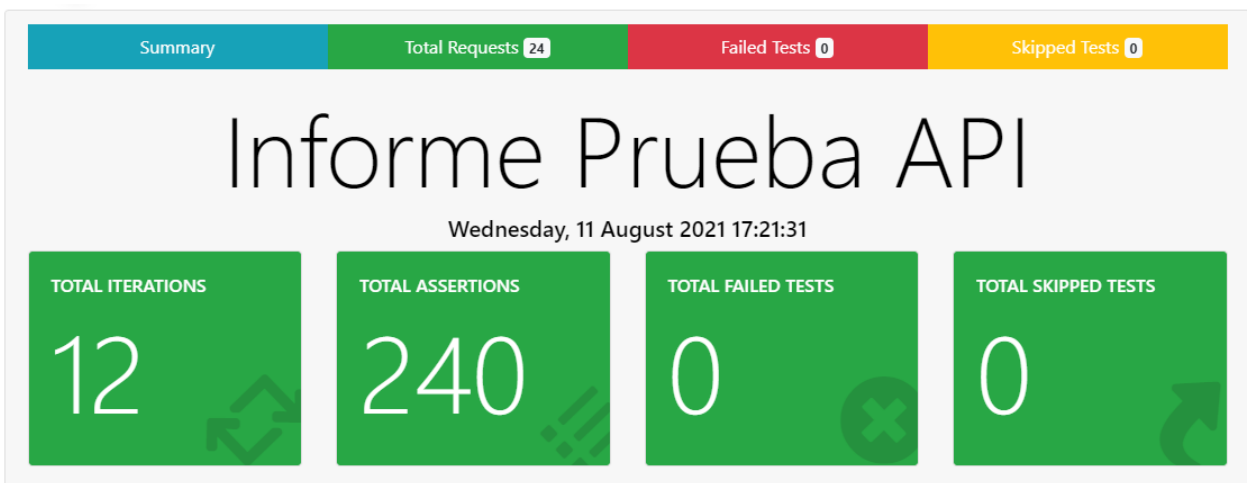


Figura 42. Informe de ejecución de pruebas automatizadas en APIs
Fuente: (Elaboración propia)

Feature: Funcionamiento del Course

Como analista de operaciones financieras Necesito una pantalla para poder identificar y activar distintos flujos de operaciones, dependiendo de los estados y condiciones de los pagos Para poder identificar estados, condiciones de las operaciones y acciones posibles

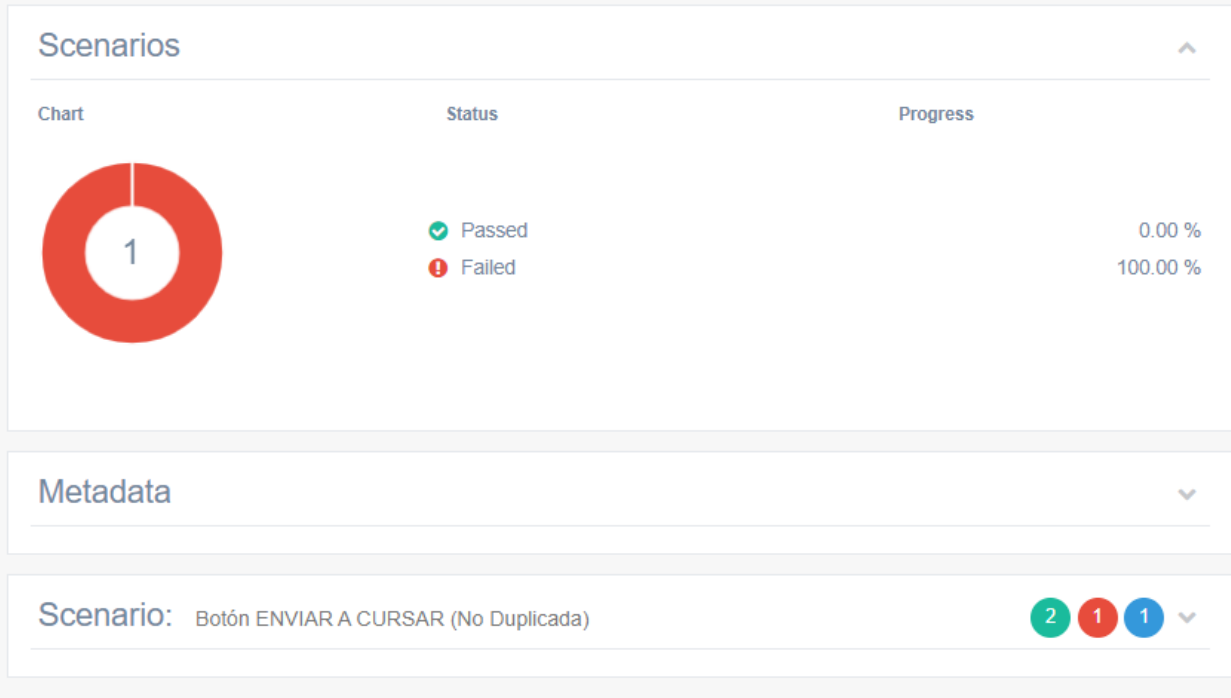


Figura 43. Informe de ejecución de pruebas automatizadas utilizando BDD
Fuente: (Elaboración propia)

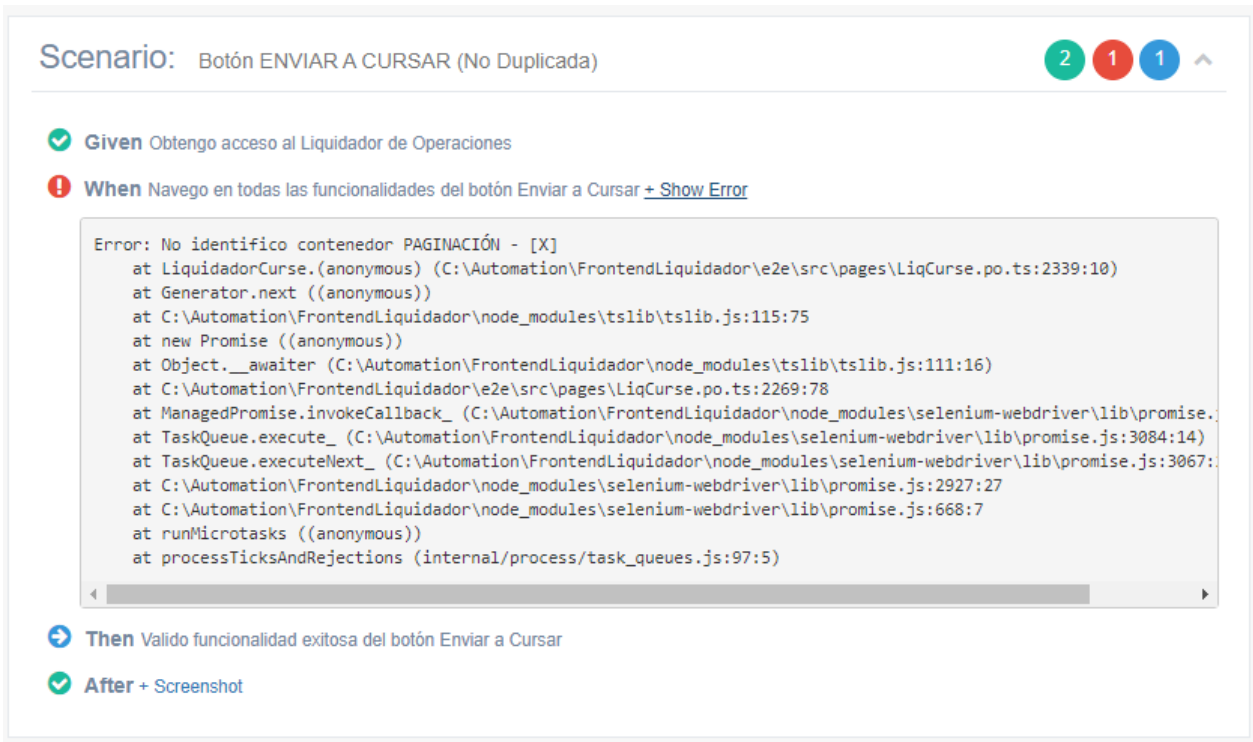


Figura 44. Informe de ejecución de pruebas automatizadas utilizando BDD
Fuente: (Elaboración propia)

Esta práctica permite detectar defectos de manera temprana y de esta manera solucionar los problemas manifestados en el diagnóstico de la situación actual dado que elimina la manualidad del proceso de pruebas y además anticipa las pruebas para evitar volumen de defectos en etapas cercanas a la producción.

CAPÍTULO 5: CONCLUSIONES Y TRABAJOS FUTUROS

En Chile, la industria bancaria es altamente competitiva, las instituciones financieras luchan agresivamente contra sus competidores para aumentar sus colocaciones y mejorar la posición de la institución en el mercado, bajo este escenario es vital poder anticipar a la competencia y entregar a los clientes soluciones que garanticen su permanencia y atraer a nuevos clientes.

5.1 Trabajo realizado

El trabajo realizado en este proyecto incorporó prácticas y principios de DevOps, que, junto a la metodología ágil, permiten a Banco SCL lanzar sus productos y soluciones digitales en menor tiempo, con la finalidad de anticipar a los otros banco e instituciones financieras del mercado chileno, lo que permite el cumplimiento de la estrategia planteada de llegar antes con las soluciones digitales a los clientes y no clientes.

Para conseguir el cumplimiento de este objetivo fue necesario, en primera instancia, descubrir el problema de la organización, esta etapa fue fundamental para detectar los dolores de los distintos equipos que participan en el proceso de construcción y entrega de software. Luego se construyó una solución de rediseño de procesos respecto a la construcción y entrega de software, en esta etapa fue necesario realizar un proceso de investigación lo que permitió conocer el modelo de operación de DevOps en conjunto con todo el catálogo de herramientas que el mercado tecnológico provee para este propósito. En otra línea de investigación se exploró SAFe como la arquitectura de referencia para aplicar el rediseño de procesos. Por último, se realizó la selección e implementación de las herramientas más adecuadas para llevar a cabo este proyecto.

5.2 Lecciones aprendidas

Con respecto a las lecciones aprendidas se quiere destacar la cultura empresarial como un concepto muy relevante a la hora de llevar a cabo proyectos que cambian la forma de trabajar de las personas. Durante la implementación de este proyecto, fue considerado un proceso de gestión del cambio para acompañar la transformación de la mejor manera posible, sin embargo, la cultura es mucho más fuerte que cualquier proceso de cambio y puede llegar incluso a derribar un proyecto con tantas bondades como lo es el modelo DevOps. En este caso la implementación consideraba un proceso de capacitación para cada proceso rediseñado y para cada herramienta implementada, además de una medición de la adopción de los cambios culturales. Pese a ello, la cultura de trabajo en silos por las áreas que durante muchos años han trabajado de manera antagónica,

presentaron una fuerte resistencia a los cambios planteado por DevOps, que propone un modelo trabajo colaborativo entre todas las áreas para logra un mismo objetivo.

5.3 Impacto del rediseño

El impacto del rediseño realizado implicó un cambio profundo en el proceso de construcción y entrega de soluciones tecnológicas, este rediseño puede ser presentado en dos dimensiones:

La primera tiene relación con el modelo de trabajo colaborativo entre todas las áreas que participan en el proceso de construcción de software, generando mayor sinergia entra el negocio y las áreas de construcción y entrega de software, y de esta manera, se logra satisfacer la estrategia del banco.

La otra dimensión tiene relación a la utilización de la tecnología y las herramientas, logrando un modelo de DevOps con un orquestador que organiza las etapas del proceso, además de incorporar un alto grado de automatización que permiten aumentar la calidad del software minimizando fallas, disminuir defecto y eliminar procesos burocráticos.

5.4 Trabajo futuro

El presente trabajo de tesis incorpora un gran avance en la evolución del modelo de construir y entregar software en menor tiempo y con un alto grado de calidad. Como trabajo futuro se propone la generación de un proceso de analítica predictiva, que permita anticipar los incidentes productivos generados por pases de software a producción. Para la implementación de este modelo predictivo se deben realizar los siguientes pasos:

Lo primero es la implementación de un lago de datos que permita alojar todos los datos que son generados por el proceso de despliegue.

Como segundo paso se deberá crear un proceso en Jenkins que permita copiar todos los datos generados en los resultados de los procesos automatizados del modelo DevOps que se han implementado en este proyecto de tesis.

Como tercer y último punto se debe construir un proceso de analítica que permita procesar toda la información inyectada en el Data Lake para luego procesar dicha información y realizar un análisis predictivo de incidentes en la producción.

CAPÍTULO 6: BIBLIOGRAFÍA

Agile Manifesto. (2001). Obtenido de <http://agilemanifesto.org/>

Agile Alliance. (s.f.). Obtenido de <https://www.agilealliance.org/glossary/kanban/>

Barros, O. (2009). *Ingeniería de Negocios: Diseño Integrado de Negocios, Procesos y Aplicaciones TI*.

Barros, O. (2015). *Ingeniería de Negocios: Diseño Integrado de Servicios, sus Procesos y Apoyo TI*.

Barros, O. (2017). *MODELACION Y DISEÑO DE UNA ARQUITECTURA EMPRESARIAL (AE) MULTINIVEL COMPLEJA*.

Best Practice DevOps. (s.f.). Obtenido de <https://netmind.net/es/devops/>

Fair, J. (2012). *Agile versus Waterfall*. *PMI global congress*.

SAFe Scaled Agile. (s.f.). Obtenido de <https://www.scaledagileframework.com/>

SAFe, C. (2020). *Case studies written by SAFe customers*. Obtenido de <https://www.scaledagile.com/customer-stories/>

Schwaber, K., & Sutherland, J. (2017). *Scrum guide*. Obtenido de <https://www.scrumguides.org/scrum-guide.html>

Twelve Principles of Agile. (2001). Obtenido de <http://agilemanifesto.org/principles.html>