UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DEL PROCESO DE TRANSFORMACIÓN DIGITAL EN UNA MUTUARIA

TESIS PARA OPTAR AL GRADO DE MAGISTER EN TECNOLOGÍAS DE LA INFORMACIÓN

CAMILO ANDRÉS SALAZAR RIQUELME

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:

NELSON BALOIAN TATARYAN

PEDRO ROSSEL CID

IVÁN SIPIRÁN MENDOZA

SANTIAGO DE CHILE
2022

IMPLEMENTACIÓN DEL PROCESO DE TRANSFORMACIÓN DIGITAL EN UNA MUTUARIA

Este documento corresponde a la tesis para optar al grado de Magister en Tecnologías de la Información de la Universidad de Chile. El trabajo consiste en una intervención realizada en una empresa mutuo-hipotecaria desde la perspectiva de sus procesos de negocio, desarrollo de software y metodología de trabajo. La problemática que se aborda está relacionada con la baja adaptabilidad que presenta la organización y que debido a los cambios en el entorno tuvo que ser mejorada para permitir que el negocio pudiera seguir operando. El alcance académico está limitado al equipo de desarrollo de software, su forma de organizarse y su forma de interactuar con las demás áreas de la empresa. De esta manera, se abordan temas técnicos como tecnologías de desarrollo, herramientas de integración continua y metodologías ágiles.

Primero se abordan aspectos metodológicos presentados como antecedentes que motivan el marco de trabajo implementado, correspondiente a LeSS. También se presentan las herramientas tecnológicas utilizadas para desarrollar el producto digital y para apoyar el proceso de desarrollo, incluyendo conceptos de computación en la nube, automatización de pruebas e integración continua.

Posteriormente, se describe el levantamiento y diagnóstico realizado en la organización previo a la intervención, identificando las oportunidades de mejora inmediata que pueden traer valor más rápidamente a la empresa. También se presentan los acuerdos de trabajo propuestos y adoptados por el equipo de desarrollo junto con las principales dificultades identificadas al momento de realizar cambios dentro de la organización, involucrando a las distintas partes interesadas. Además, se dedica un capítulo a describir el producto digital desde una perspectiva técnica, abordando temas arquitectónicos, desarrollo dirigido por pruebas y la descripción del flujo de trabajo del desarrollo e implementación del producto.

En los últimos capítulos del documento se realiza un análisis de resultados en torno a impacto en el negocio y también en torno a percepción de las personas respecto al nivel de adopción y la adherencia al marco de trabajo propuesto. Con respecto a las métricas del negocio, después de haber implementado el nuevo producto digital, se observó una sustancial mejora en la tasa de conversión de las simulaciones y un incremento en el porcentaje de clientes morosos. Por otro lado, respecto al nivel de adopción de LeSS se realizó una encuesta de percepción que permite tener una visión sobre el nivel de adherencia que existe con respecto a cada uno de los pilares fundamentales que propone este marco de trabajo.

Finalmente se deja propuesto abordar la problemática de implantar el negocio en distintos países manteniendo un solo producto digital, sin comprometer los valores que propone LeSS y evitando la generación de verticales dentro de la organización que puedan comprometer la capacidad de adaptarse de la empresa.

Dedicado a todos aquellos desarrolladores que alguna vez se han frustrado

usando metodologías ágiles…

AGRADECIMIENTOS

Voy a partir agradeciendo a mi buen amigo y antiguo compañero de clases, Dr. Víctor Dantas, con quien compartí los períodos más difíciles de nuestra estadía en Marsella como estudiantes extranjeros y quien recientemente me devolvió la confianza en la academia para poder seguir haciendo exposiciones e investigación a pesar de que existan grupos ideológicos que mantienen secuestrados la mayoría de los espacios universitarios.

Agradezco también a la alta gerencia de la empresa en que se desarrolló este trabajo de magister porque siempre se mostraron muy abiertos tanto a realizar cambios en la organización como a ejecutar experimentos para explorar hipótesis que no siempre tenían garantías de obtener los resultados esperados.

Continúo agradeciendo a la profesora guía de este trabajo de magister, Dra. Jocelyn Simmonds, por haber acogido mi propuesta en condiciones difíciles como lo son un calendario universitario afectado por la pandemia y un resumen de propuesta de tesis víctima de la natural falta de rigurosidad académica de la industria financiera, además de haber compartido sus conocimientos en aseguramiento de calidad de software durante el curso que hice con ella.

Tampoco puedo dejar de agradecer a mi madre, Nubia Riquelme, quien me apoyó en las difíciles circunstancias que se crearon durante el período de pandemia en que la universidad no logró realizar sus cobros a tiempo, acumulando aranceles por dos años seguidos.

TABLA DE CONTENIDO

INDICE DE TABLAS

INDICE DE FIGURAS

1 INTRODUCCIÓN

El concepto de transformación digital hace referencia a la práctica de adoptar tecnologías digitales dentro de una empresa[1] remplazando aquellos procesos que se realizan de forma manual o que utilicen tecnologías más antiguas [1]. El sentido de implementar soluciones digitales es poder mejorar la eficiencia del negocio o automatizar tareas, además de generar oportunidades de innovación o mejora.

Sin embargo, para poder aprovechar las oportunidades de mejora que la tecnología puede ofrecer no basta sólo con incorporar medios digitales en los procesos de negocio, sino que también es necesario realizar cambios estructurales, culturales y de diseño de los procesos del negocio. De hecho, como se expone en este trabajo, existen casos en que la incorporación de tecnología sin cambios en la filosofía de trabajo resulta en situaciones menos eficientes o incluso perjudiciales para la organización.

Dentro de la industria financiera se encuentra el nicho de los mutuos y créditos hipotecarios. Debido a las limitantes que impone la legislación[2] existen elementos que no pueden ser libremente digitalizados ni propuestos por los actores que realizan negocios de este tipo, ya que dependen de parámetros dictados por organismos reguladores o de verificaciones por parte actores externos a las partes interesadas. Por ejemplo, la tasa convencional máxima es un número otorgado mensualmente por el organismo regulador. Además, el fuerte carácter burocrático de este tipo de negocios[3] provoca que existan elementos que no pueden ser automatizados o cuya automatización depende de variables externas. Estos factores explican por qué las barreras enfrentadas para realizar un proceso de transformación digital en una empresa dedicada a los mutuos y créditos hipotecarios son más grandes que en otros nichos financieros o en otras industrias.

Este documento presenta un conjunto de lineamientos obtenidos a partir de un caso exitoso de implementación del proceso de transformación digital en una organización dedicada al negocio de los mutuos y créditos hipotecarios. Se abarcan técnicas de desarrollo de software, diseño de arquitectura, incorporación de tecnologías digitales para automatizar y estandarizar procesos, además de propuestas de cambios estructurales y de cultura de trabajo que favorecen la innovación en la operación del negocio y la capacidad de respuesta de la organización ante cambios en el entorno, por medio de identificación de oportunidades de mejora y detección temprana de amenazas.

---

[1] Existen autores que han usado el término «transformación digital» en un sentido más extenso, para referirse también a cambios en distintos aspectos de la sociedad.

[2] Regulación de la tasa de interés, restricciones de tipos de fondos para financiamiento, limitaciones en el porcentaje permitido financiar, entre otros.

[3] Documentación necesaria para evaluación financiera, firma de promesas de compraventa y escrituras en notaría, inscripción en Conservador de Bienes Raíces, entre otros.

Este trabajo se enmarca en el contexto de una mutuaria o empresa mutuo-hipotecaria, esto es, una institución financiera y por lo tanto regulada por la Comisión para el Mercado Financiero (CMF)[4]. El negocio se sostiene en la oferta de créditos hipotecarios y la asociación con una empresa aseguradora que permite la venta de mutuos de inversión a quienes busquen obtener retornos según rentabilidad.

El financiamiento de los créditos hipotecarios se origina en las rentas vitalicias acumuladas por la compañía de seguros, de tal forma que la recaudación de los dividendos es retornada a quien haya adquirido el mutuo de inversión correspondiente. De esta forma, el negocio responde a un modelo de doble cartera que tiene por un lado a los clientes de la mutuaria que adquieren créditos hipotecarios y por el otro a los inversionistas (también llamados clientes institucionales) que compran los mutuos de inversión para obtener rentabilidades.

1.1 Motivación y problemática

Tanto la operación del negocio como la interacción con los clientes ocurre siempre de manera presencial, ya sea en las salas de ventas de proyectos inmobiliarios a través de ejecutivos en terreno o en las oficinas de atención al cliente. Cuando un cliente adquiere un crédito hipotecario debe presentar documentación de antecedentes financieros y firmar compromisos de forma física. Los ejecutivos de la organización en la que se realizó este trabajo acompañan de cerca y asisten al cliente durante el proceso de otorgamiento del crédito y firmas en notaría, además de orientarlo hasta el pago del primer dividendo.

Los inversionistas también son atendidos presencialmente por los ejecutivos, que les ofrecen distintos mutuos de inversión según volumen, rentabilidad y riesgos. Un inversionista puede comprar varios mutuos asociados a distintos créditos hipotecarios y percibir los intereses generados a medida que los deudores van pagando los dividendos de los correspondientes créditos hipotecarios.

1.1.1 Estructura de la organización

Existen ocho gerencias que están a cargo de distintas áreas de la compañía en la cual se realizó este trabajo. Cada una de estas áreas tiene una responsabilidad relacionada a la operación y la continuidad del negocio. Existen dos áreas en las que sólo trabajan sus correspondientes gerente y subgerente porque han externalizado las actividades por medio de servicios entregados por proveedores. El área más grande de la empresa se llama *backoffice* y es el departamento en que trabajan los ejecutivos que asisten y orientan a los clientes. La Tabla 1-1 muestra el tamaño de cada área en términos de número de personas junto con su responsabilidad específica dentro del negocio.

[4] Servicio público descentralizado que reemplazó a la Superintendencia de Valores y Seguros, cumpliendo funciones de supervisión, de regulación, de sanción y de desarrollo de los mercados financieros.

Área	Nº de personas	Responsabilidad
Backoffice	35	Captación y registro de nuevos clientes. Recolección de documentación para el área de riesgo. Soporte al cliente durante generación de la oferta, venta del crédito hipotecario y escrituración de la propiedad.
Marketing	12	Medición de la conversión (*funnel*) de clientes. Generación de campañas de captación de nuevos clientes. Presencia en salas de ventas de proyectos inmobiliarios.
Tesorería	8	Remesa de mutuos. Desembolso de dineros a inversionistas.
Cobranza	2	Externalizado. Gestión legal de créditos siniestrados o dividendos atrasados.
Finanzas	8	Definir las políticas de riesgo, las tasas de interés a nivel corporativo y preferenciales según asociaciones con inmobiliarias. Negociar tasas de rentabilidad con organizaciones inversionistas que compran mutuos.
Riesgo	8	Aplicar las políticas definidas por el área de finanzas durante la evaluación de riesgo de cada cliente. Verificación y depuración de documentación recolectada por el *backoffice*. Elaboración de la oferta de crédito hipotecario para el cliente.
Administración primaria	2	Externalizado. Recaudación de dividendos. Administración de fondos recaudados asociados a mutuos comprados por inversionistas. Cálculo de intereses por mora.
TI	8	Asegurar disponibilidad y mantención del software ERP sobre el cual se apoya la operación del negocio.

Tabla 1-1: Responsabilidades de las áreas de la organización.

La organización cuenta con un software de planificación de recursos empresariales (ERP) operado desde una plataforma web sobre la cual se apoya la completa gestión del proceso de evaluación de riesgo del cliente, oferta del crédito hipotecario, venta del mutuo, cobranza de dividendos y remesa de dineros a inversionistas. El software es usado por distintas áreas de la compañía que intervienen en la operación.

El área de TI cuenta con un equipo de desarrollo que construyó el ERP y está a cargo de asegurar su disponibilidad. El equipo de desarrollo funciona como servicio, atendiendo requerimientos de los demás departamentos de la compañía que pueden ser de distinta índole, por ejemplo: reporte de incidencias, solicitudes de enrolar y/o dar accesos para nuevos usuarios, actualización de políticas de evaluación de riesgo dentro de la plataforma, reporte de errores en el software, solicitud de implementación de nuevas características específicas para algún área en particular.

1.1.2 Descripción del problema

Aunque el área de TI está encargada del desarrollo y mantención del software sobre el cual se apoya la operación, no se puede considerar que sea su dueña. El equipo de desarrollo se limita a atender solicitudes de las distintas partes interesadas de la compañía y a satisfacer requerimientos de las demás áreas, pero en ningún caso puede proponer incrementos ni replantear procesos del negocio.

Es común que un cambio en el sistema solicitado por un área en particular genere un problema relacionado a otra área u otra etapa del proceso de negocio. Dentro del equipo de desarrollo se forman optimizaciones locales, esto es, dependencias internas debido a especialización por capas o por componentes. Si un desarrollador estuvo a cargo de satisfacer un requerimiento puntual, es de esperarse que la misma persona también atienda futuros requerimientos que tengan características o funcionalidades comunes. Luego de pocos requerimientos relacionados a un mismo interés y completados por el mismo desarrollador (que ahora es un especialista), el equipo comienza a depender de esa persona y, por lo tanto, aparece la optimización local.

Por otro lado, dado que la operación depende de interacciones presenciales con los clientes, cualquier cambio en el ecosistema en que se desenvuelve la empresa trae fuertes consecuencias. El proceso se detiene cuando el cliente no puede reunirse con los ejecutivos comerciales. Luego de los acontecimientos que tuvieron lugar en Chile a partir del 18 de octubre 2019 y la posterior pandemia, la organización tuvo otorgamientos de créditos detenidos por hasta 9 meses y entró en un período de 87 días hábiles sin ventas de mutuos de inversión.

Ambos fenómenos provocan que tanto el sistema como el proceso de otorgamiento de créditos hipotecarios y ventas de mutuos sean extremadamente rígidos y dependientes de acciones manuales. De hecho, la combinación de estos dos escenarios provocó que el área de TI paralizara sus actividades completamente debido a la ausencia de solicitudes que atender.

El problema que pretende abordar este trabajo es que el equipo de desarrollo no cuenta con herramientas y/o instancias para poder adaptarse y responder ante cambios en el entorno en el que se desenvuelve. Para ahondar en esta idea, en las siguientes tres secciones se presentarán distintas aristas que permiten estudiar el problema desde una perspectiva específica.

1.1.3 Pérdida de propósito

El equipo de desarrollo debe satisfacer requerimientos de distintas partes interesadas, provocando que en un mismo período de tiempo se estén atendiendo solicitudes que no están necesariamente relacionadas entre ellas. Por ejemplo, en un momento puede llegar una solicitud de parte del *backoffice* junto con otra de parte del departamento de finanzas.

Cuando se comienza a trabajar en modificaciones de elementos del software que no están relacionadas ni tampoco apuntan a un mismo fin, se crean distintos objetivos en el flujo de trabajo de un mismo equipo. Este fenómeno provoca que sea más difícil mantener el foco dentro del equipo de desarrollo y, por consecuencia, dificulta perseguir metas dentro del área de TI.

Satisfacer un requerimiento puntual dentro de un período comprometido no garantiza que otra solicitud sea más fácil de atender ni tampoco que el equipo de

desarrollo se acerque a cumplir sus metas. La presencia de distintos intereses y la ausencia de objetivos comunes dentro del equipo de desarrollo generan pérdida de foco en el flujo de trabajo. Una consecuencia inmediata de la pérdida de foco es que los objetivos propuestos se vuelven irrelevantes. Y finalmente, cuando los objetivos son irrelevantes el equipo de desarrollo pierde su sentido del propósito.

1.1.4 Realización de trabajo sin impacto

Dado que los usuarios del ERP son los encargados de ejecutar la operación del negocio, existe la creencia de que cada solicitud que atiende el equipo de desarrollo tendrá un impacto positivo en el negocio. Sin embargo, la realidad es que un requerimiento particular motivado por los intereses de una de las partes de la organización sólo garantiza que la experiencia particular de ese usuario mejore, pero no así el proceso completo.

Dentro de la organización en que se enmarca este trabajo, es sabido que varios de los requerimientos que llegan al equipo de desarrollo están motivados solamente en deseos puntuales e impresiones subjetivas de los usuarios del ERP. Tanto el equipo de desarrollo como las demás áreas de la compañía tienen experiencias con solicitudes que requieren varios días o incluso semanas de dedicación de parte del equipo de desarrollo, que finalmente cumplen la expectativa del usuario que generó el requerimiento, pero no muestran mejoras en el negocio o en la operación en términos de conversión de ventas o disminución de clientes morosos.

Todas estas condiciones provocan que el equipo de desarrollo tienda a maximizar la cantidad de trabajo realizado, con un impacto mínimo sobre el negocio.

1.1.5 Baja calidad del software

La calidad del software está relacionada a su diseño tanto a nivel de modelo de datos, arquitectura, escalabilidad y calidad del código. Dado que escapa al interés de los usuarios del ERP, cuando llega un requerimiento de alguna de las áreas de la compañía, estos no son planteados en términos de la mantenibilidad del software o de las capacidades técnicas del equipo de desarrollo. Esto significa que, al momento de atender la solicitud, el equipo de desarrollo debe construir nuevas piezas del software sobre el código que ya está escrito.

Las metas y objetivos del equipo de desarrollo se definen según los requerimientos de las demás áreas de la empresa, por esa razón es que el equipo de desarrollo trabaja como un servicio para los demás departamentos de la organización (ver sección 1.1.1). Esta forma de trabajo impide que existan instancias para detenerse a proponer modelos de datos consistentes o escalables, o de implementar piezas modularizadas y reutilizables. Cuando aparece un requerimiento de una parte interesada, el tiempo disponible se dedica completamente a satisfacerlo sin invertir en un software mantenible y escalable. Naturalmente, este esquema no es sustentable para un software que está constantemente recibiendo incrementos y/o

modificaciones, ya que se genera una tendencia a aumentar la posibilidad de errores o adoptar malas prácticas de desarrollo de software.

Las demás áreas de la compañía tienen intereses distintos a la calidad del software. Esto provoca que no haya acuerdos de trabajo relacionados a la calidad ni a la reutilización de código. Un ejemplo de ello es que no existe un estándar de ejecución de pruebas del software, lo que provoca fenómenos tales como que una modificación en el ERP que tiene como objetivo satisfacer a algún usuario, pueda romper características previas o generar inconsistencia en los datos del sistema.

1.2 Objetivo

Teniendo en cuenta las aristas del problema descritas en las secciones 1.1.3, 1.1.4 y 1.1.5 se propone como objetivo general de este trabajo implementar un marco de trabajo para el equipo de desarrollo que le permita determinar y priorizar requerimientos, definir modificaciones en el proceso del negocio y en el software sin depender de los demás departamentos de la organización, para poder aumentar la capacidad de responder frente a cambios en el entorno de la empresa.

Para determinar el cumplimiento del objetivo propuesto se establecen también los siguientes objetivos específicos:

1. Conformar un equipo autoorganizado y multidisciplinario que sea responsable tanto del desarrollo y mantenimiento del software como del aseguramiento de la continuidad del negocio de tal manera que pueda determinar qué trabajo es necesario realizar para incrementar el producto.
2. Establecer y validar un estándar mínimo de calidad de software, para impedir modificaciones que no lo satisfacen y asegurar la integración de los cambios que sí lo cumplan.

En las siguientes tres secciones se van a desarrollar las ideas que motivan los objetivos específicos propuestos.

1.2.1 Generar ownership dentro del equipo de desarrollo

Teniendo en cuenta la problemática relacionada a la pérdida de propósito descrita en la sección 1.1.3, es que se plantea que equipo de desarrollo debe ser dueño del producto que construye; eso significa que los requerimientos y objetivos serán propuestos por el mismo equipo en lugar de las demás áreas de la compañía. De esta forma, el equipo de desarrollo se ve obligado a tomar la completa responsabilidad del diseño, despliegue, mantención y escalado del producto. El equipo deberá proponer incrementos basándose en las necesidades del negocio y para lograrlo es necesario comprender en profundidad el proceso y la operación. Eso significa que tendrá que recoger información tanto de las demás áreas de la organización como de ambas carteras de clientes, para comprender las dificultades y los dolores de cada una de las partes, de tal manera que los ítems propuestos para satisfacer significan una mejora en el negocio.

En coherencia con la idea de que el quipo pueda proponer sus propios requerimientos y plantear sus propios objetivos, la forma de medir el desempeño del equipo de desarrollo ya estará determinada por el cumplimiento de metas de la compañía y del negocio como pueden ser índices de mutuos vendidos, créditos otorgados, alcance de nuevos clientes y todas las demás métricas que afectan a las distintas áreas de la organización.

1.2.2 Priorización de requerimientos

Atendiendo la situación expuesta en la sección 1.1.4, el equipo de desarrollo tendrá que generar sus propios requerimientos técnicos basándose en las necesidades del negocio y para lograrlo es necesario comprender en profundidad el proceso y la operación. También tendrá que recoger información tanto de las demás áreas de la organización como de ambas carteras de clientes, para comprender las dificultades y los dolores de cada una de las partes.

El impacto que tienen las entregas del equipo de desarrollo debe ser medido en función del efecto que se espera que causen en el negocio y entonces determinar si se está logrando priorizar aquellos requerimientos que tienen menor costo de implementación y mayor valor para el negocio.

1.2.3 Integración continua

En un escenario en que el software está constantemente recibiendo modificaciones, es deseable tener un software modularizado, reutilizable y escalable para favorecer el resguardo de la calidad. Si el equipo de desarrollo diseña completamente el producto, es mucho más fácil obtener estas condiciones. Además, es buena idea hacer uso de herramientas tecnológicas que ayudan a asegurar la calidad de las piezas de software que se vayan a desplegar, agilizando la entrega de estas. Así, es posible comparar la cantidad de despliegues de software contra las ejecuciones exitosas de las pruebas de software, de tal manera que no debería existir ningún despliegue sin su correspondiente ejecución de pruebas automáticas y aseguramiento de calidad.

1.3 Solución propuesta

Teniendo en cuenta los dos objetivos específicos presentados en el apartado anterior, este trabajo hace uso de un *framework* llamado LeSS, cuyos lineamientos obligan a que el equipo de desarrollo tome mayores responsabilidades de cara a los procesos de negocio y el diseño del producto digital que use la organización. Por otro lado, y para asegurar la calidad del software, dentro del mismo marco de trabajo que propone LeSS, es que se implementó un sistema de integración continua que logra la automatización de ejecución de pruebas y despliegue de nuevas versiones del sofrware.

1.4 Alcance

Uno de los elementos que la intervención propone es un cambio en la forma en que el equipo de desarrollo se organiza y por lo tanto también cambia la forma en que este interactúa con las demás áreas de la compañía. A pesar de que un cambio en la filosofía de trabajo y en la estructura de la organización afecta directamente la manera de plantear metas dentro de los distintos equipos de trabajo, este trabajo se limitará a las definiciones y herramientas adoptadas por el equipo de desarrollo, sin ahondar en las consecuencias que esto significa para las responsabilidades de las demás áreas.

Se abordarán aspectos relacionados a tres áreas de interés, que en conjunto apuntan a completar la transformación digital dentro de la organización pero que no están necesariamente relacionadas de forma directa. Estos son el levantamiento o diagnóstico del estado inicial del proceso de negocio y la propuesta del nuevo proceso con un producto digital, la definición del proceso de desarrollo del producto y acuerdos de trabajo dentro del equipo de desarrollo enmarcado en un *framework* específico y el diseño e implementación del producto digital, en términos de modelo de datos, tecnologías utilizadas y código fuente del software.

1.5 Estructura del documento

Además de la presente sección introductoria, hay un capítulo dedicado a revisar la bibliografía que respalda los cambios introducidos en el equipo de desarrollo, las filosofías de trabajo implementadas y las herramientas técnicas utilizadas para facilitar el proceso de desarrollo de software.

En el capítulo 3 se describe el trabajo realizado dentro de la organización que, en concordancia con lo descrito en el punto 1.4, se presenta en dos secciones dedicadas los aspectos de diagnóstico, propuesta y oportunidades de mejora e implementación de la nueva forma de administrar el trabajo y al equipo de desarrollo.

El capítulo 4 se refiere al trabajo y los cambios realizados en la organización desde la perspectiva técnica que involucra el uso de tecnologías modernas, el diseño y la implementación del software o el nuevo producto digital de la empresa.

Luego, se dedica un capítulo analizar los resultados obtenidos, tanto desde la perspectiva del impacto sobre el negocio como desde el punto de vista de la percepción del nivel de adopción de las nuevas filosofías de trabajo por parte de los integrantes del equipo de desarrollo.

El último capítulo de esta tesis propone conclusiones obtenidas luego de identificar los obstáculos más relevantes y los elementos mínimos necesarios para poder realizar de forma exitosa la implantación del proceso de transformación digital dentro de la organización.

2 ESTADO DEL ARTE

En este capítulo se realizará una revisión de los conceptos y elementos que justifican o respaldan la solución propuesta dentro de la empresa intervenida durante la confección de esta tesis. Este capítulo está dividido según dos aspectos abordados desde una perspectiva teórica y/o referencial. El primero es el metodológico, que abarca los temas relacionados a la cultura organizacional y filosofías de trabajo. El segundo es el técnico, que envuelve conceptos relacionados a herramientas y prácticas que favorecen la ejecución y aplicación de los aspectos metodológicos.

2.1 Marcos de trabajo

El término *framework* o marco de trabajo hace referencia a una metodología utilizada para controlar, planificar y estructurar el proceso de construcción de un producto [1]. Lo que un marco de trabajo pretende hacer es estandarizar prácticas para tener una referencia general a nivel técnico y conceptual sobre la forma de abordar un tipo genérico de problema. La idea detrás de la utilización o la implantación de un marco de trabajo es poder encontrar patrones comunes en situaciones específicas que puedan ser enfrentadas en función de las sugerencias que brinde el marco de trabajo [1].

Existen dos grandes corrientes que definen distintos métodos de desarrollo de software. En una se encuentran los llamados métodos rigurosos y en la otra se encuentran las metodologías ágiles [2]. Los métodos rigurosos se caracterizan por favorecer el orden de las etapas del ciclo de vida del software, mientras que las metodologías ágiles favorecen la descomposición de la secuencia del proceso de desarrollo en pequeñas entregas que van incrementando el producto que se está desarrollando [2].

Este trabajo utiliza un marco de trabajo llamado LeSS [3], que se encuentra dentro del conjunto de metodologías ágiles. Para poder justificar esta elección es necesario revisar algunas definiciones que motivan las propuestas de este marco de trabajo. A continuación, se presentará una revisión de distintos marcos de trabajo que pueden explicar por qué, en una situación de implantar un proceso de transformación digital en una empresa, es buena idea usar metodologías ágiles de desarrollo de software.

Para poder entender qué es un “problema complejo”, primero se abordará *cynefin,* que define distintos dominios en que se pueden presentar los problemas. Luego se aborda el modelo en cascada, que es un método riguroso de desarrollo de software que tiene fortalezas y también debilidades que motivaron el surgimiento de las metodologías ágiles. Finalmente se revisa el concepto de metodologías ágiles y se presentan Scrum, que es la metodología ágil más usada en la industria seguido de LeSS, que es un marco de trabajo que permite hacer escalar Scrum.

2.1.1 Cynefin

Es un marco de trabajo que propone a una conceptualización para identificar características presentes en distintos contextos según su nivel de complejidad [4]. Tiene como propósito facilitar escenarios en que el proceso de toma de decisiones sea más evidente y propone cinco dominios para clasificar contextos según sus características. Cada dominio presenta niveles distintos de complejidad en los que se puede encontrar un problema específico y sobre el cual la estrategia sugerida para resolverlo cambia según el dominio en que se encuentre [4]. Los cinco dominios definidos se llaman claro, complicado, complejo, caótico y desordenado [4] (ver Figura 2-1).

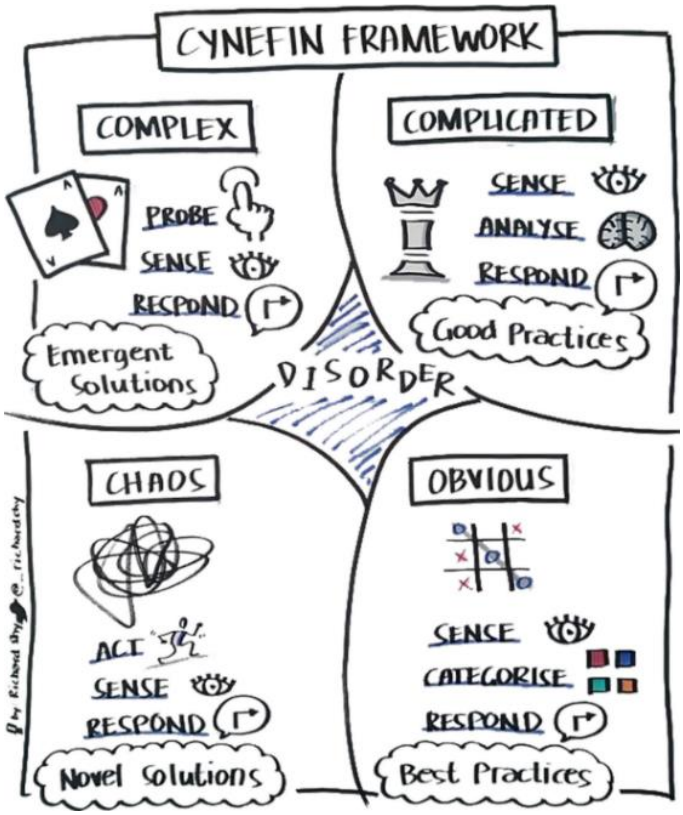


Figura 2-1:Representación gráfica del marco Cynefin. Imagen extraída del artículo en Medium https://medium.com/@rob.hendersonmx/4-el-marco-cynefin-1-de-2-entregas-b272d5c61ecb (15 marzo 2020)

2.1.1.1 Claro

Representa aquellas situaciones en que existen “certezas conocidas”, o sea que hay reglas establecidas que pueden predecir comportamientos y las relaciones de causalidad y efectos son estables. En un escenario que se encuentre clasificado dentro de este dominio, es posible establecer “mejores prácticas” o recetas y la estrategia sugerida para enfrentar problemas que se encuentran en este dominio es la de observar, categorizar y responder aplicando las reglas establecidas.

2.1.1.2 Complicado

Abarca aquellos contextos que presentan características con “certezas desconocidas”. Generalmente esto corresponde a escenarios en que la relación entre la causalidad y el efecto necesita análisis y conocimiento de especialistas para poder ser determinada. En este dominio se encuentran aquellos problemas en que se han podido establecer “buenas prácticas” que ayudan a solucionarlos, pero a la vez existe un rango amplio de soluciones posibles. La estrategia recomendada al enfrentar un problema complicado es la de observar para evaluar los hechos, analizar la situación y responder adecuadamente.

2.1.1.3 Complejo

Se puede caracterizar por la presencia de “incertidumbres desconocidas” y la relación entre causalidad y efecto sólo puede ser determinada después de haber accionado medidas de respuesta. Para este tipo de escenario no existen prácticas correctas predefinidas y las buenas prácticas podrían eventualmente emerger, pero no existe seguridad de que eso siempre ocurra. La estrategia recomendada para enfrentar problemas que se encuentren dentro de este dominio es la de investigar realizando experimentos de bajo riesgo, observar los efectos generados y responder según los posibles patrones que hayan podido ser detectados. El problema enfrentado durante este trabajo (ver sección 1.1) se puede clasificar como un problema complejo porque, en efecto, no se tiene certeza de la mejor forma para implementar el producto digital ni tampoco de la mejor forma de reorganizar la empresa con el fin de obtener más adaptabilidad.

2.1.1.4 Caótico

En un escenario asociado a este dominio, la relación de causalidad y efecto no está clara y los elementos que se pueden observar presentan un comportamiento demasiado confuso para poder tomar decisiones basadas en conocimiento adquirido. Dado que la respuesta ante los problemas no puede ser inexistente, cualquier tipo de acción es una forma válida de enfrentarlos. La recomendación en este caso es tratar de mover el escenario hacia el dominio de lo complejo, de tal manera que al actuar se pueda establecer algún orden, luego observar en qué frentes se puede detectar algún nivel de estabilidad y responder para tratar de trasformar el problema caótico en un problema complejo.

2.1.1.5 Desorden

Describe situaciones donde no existe claridad alguna sobre en cuál de los dominios anteriores se puede categorizar un problema. Dado que está definido por el desconocimiento, es que poder determinar si acaso una situación pertenece a este dominio es extremadamente difícil. La recomendación frente a escenarios indeterminados es tratar de dividir el problema en piezas de alcance limitado para poder categorizarlas en alguno de los otros cuatro dominios y abordarlos según sus correspondientes recomendaciones.

2.1.2 Modelo en cascada

Consiste en desglosar las actividades de un proyecto en etapas o fases secuenciales de tal manera que cada una de ellas depende de la entrega de la etapa anterior [5] y [6]. A la secuencia de etapas del proyecto también se le suele llamar *ciclo de vida del producto*. Debe su denominación a una analogía con los accidentes geográficos en que el agua cae por la fuerza de gravedad hacia otro nivel del cauce [5] (ver Figura 2-2). Tiene un fuerte carácter de rigurosidad en el orden de las etapas, proponiendo que una fase no puede ser ejecutada sin haber terminado de ejecutar la anterior, o sea que es necesario haber generado un entregable de una determinada fase antes de poder continuar con la fase siguiente.

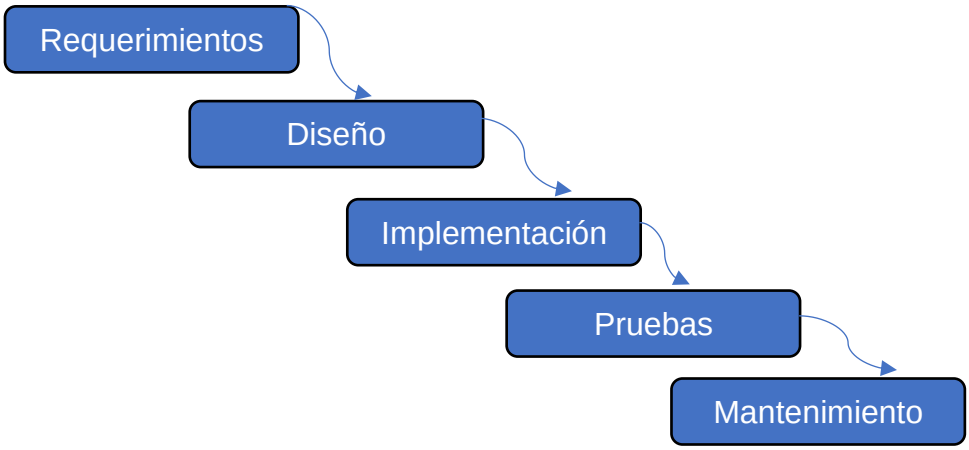


Figura 2-2: Representación gráfica del modelo en cascada.

Existen varios marcos de trabajo que usan la filosofía del modelo en cascada que proponen distintas cantidades de fases o etapas en el ciclo de vida de un software según el contexto definido en cada caso, haciendo referencia al tipo de proyecto que se trabaja o el tipo de problema que se está solucionando. Un ejemplo de propuesta común entre distintos *frameworks* que usan el modelo en cascada consiste en administrar el proyecto en las fases de toma de requerimientos, diseño del producto, implementación, pruebas y mantenimiento [6] y [5].

2.1.2.1 Toma de requerimientos

Recoge y estudia las necesidades de los usuarios del producto, favoreciendo el procedo de proponer y determinar los objetivos que van a definir la completitud del proyecto. El entregable de esta fase suele ser un documento que detalla la especificación completa de lo que el sistema debe realizar, llamado SRS (*software requirements specification*).

2.1.2.2 Diseño del producto

Consiste en el análisis y descomposición de las herramientas y módulos se deben usar y construir para implementar la solución. A partir del documento generado en la fase anterior, se propone una solución que pueda ser implementada para satisfacer los requerimientos especificados. Al final de esta fase se suele obtener

un documento que contiene la descripción de la estructura del producto y la especificación de lo que debe hacer cada uno de sus módulos, llamado SDD (*software design description*).

2.1.2.3 Implementación

Engloba las actividades y tareas relacionadas a escribir el código fuente del software. Contempla la creación de módulos y componentes cuyo nivel de reutilización o calidad dependen directamente de las descripciones disponibles en el entregable de la etapa anterior.

2.1.2.4 Pruebas

Durante esta fase se espera poner en funcionamiento el producto para verificar el comportamiento del software y compararlo con la expectativa. El objetivo de esta etapa del proyecto es poder buscar y corregir errores antes de que el producto sea entregado al usuario final. Se comprueba el correcto funcionamiento de los módulos desarrollados.

2.1.2.5 Mantenimiento

Es posible que al usar el software se revele que este no cumple con todas las expectativas del cliente o del usuario final. En esta fase se utiliza la mayoría de los recursos del proyecto para ajustar, mantener y corregir el producto. Muchos de los elementos que son necesarios de corregir pueden ser detectados en la fase de pruebas, pero es una práctica común encontrarse con comportamientos no esperados que deban ser atendidos posteriormente para poder garantizar el éxito del proyecto.

2.1.3 Metodologías ágiles

Existen razones para afirmar que, aunque el modelo en cascada se sostiene bien al enfrentar problemas claros (ver sección 2.1.1.1) o complicados (ver sección 2.1.1.2), en realidad no es la mejor estrategia para el desarrollo de un producto concebido para solucionar un problema complejo (ver sección 2.1.1.3) [7]. La mayor limitante del modelo en cascada es que no tolera modificaciones de la entrega de una fase ya terminada [7]. Esto puede significar un problema con respecto a la expectativa del producto ya que es común ver a usuarios del software o a clientes sin total claridad para describir sus necesidades [7]. Existen muchas situaciones en las que el usuario o el cliente sólo puede describir claramente sus expectativas después de haber comenzado a utilizar efectivamente el producto, lo que incluso podría hacerle cambiar su perspectiva respecto a la necesidad [5] y [7].

De esta forma, las metodologías ágiles aparecen como una respuesta a las debilidades presentes en el modelo en cascada (ver sección 2.1.2) y propone un proceso de desarrollo basado en la repetición de ciclos que resultan en entregas pequeñas al final de cada iteración [8]. Esta idea permite que el equipo de desarrollo genere aprendizajes durante cada ciclo, que pueden ser utilizados en los ciclos posteriores [8]. El aprendizaje viene tanto de la experiencia de haber desarrollado

una pequeña entrega como del uso del software o el producto a medida que este se va incrementando [9]. Existen varias propuestas de metodologías ágiles que tratan de hacerse cargo del problema referente a aumentar la adaptabilidad de un equipo de desarrollo [10]. Uno de los métodos ágiles más utilizados actualmente en la industria del desarrollo software se llama Scrum [8].

2.1.3.1 Scrum

Es un marco de trabajo concebido para ayudar a los equipos de desarrollo software a generar valor usando los principios de las metodologías ágiles [8], aumentando la adaptabilidad de las soluciones entregadas [2]. Scrum propone una serie de pautas sobre las cuales ejecutar el trabajo de desarrollo software e indica que independientemente de los artefactos utilizados y de los acuerdos de trabajo establecidos dentro del equipo, siempre se puede declarar el uso de este marco de trabajo mientras se cumplan las pautas definidas. La Tabla 2-1 contiene los roles definidos por Scrum [10] junto con su correspondiente descripción.

Rol	Descripción
Scrum master	Su tarea principal es velar por que las pautas propuestas por Scrum se cumplan, eso significa que dentro de sus responsabilidades está ayudar tanto al equipo como al resto de la organización a entender el marco de trabajo. También es un facilitador, tomando la responsabilidad de fomentar un entorno de trabajo en que los demás roles se puedan desenvolver sin obstáculos para agregar valor en conjunto.
Product owner	Debe priorizar y ordenar los ítems que representan el trabajo que el equipo debe realizar para ir incrementando el producto durante cada iteración. Su responsabilidad principal es maximizar el valor del producto, dándole más prioridad a aquellos ítems que aportan más valor y tienen menor costo de implementación.
Desarrolladores	Se comprometen a crear cualquier elemento necesario para poder entregar un incremento utilizable. Los integrantes del equipo de desarrollo deben crear un plan para ser ejecutado durante el transcurso del ciclo iterativo. También es su responsabilidad adaptar el plan en caso de ser necesario para lograr alcanzar los objetivos propuestos al comenzar la iteración.

Tabla 2-1: Roles definidos en scrum.

Scrum se basa en la aplicación de tres conceptos fundamentales llamados pilares empíricos [2]. Estos tres conceptos son transparencia, inspección y adaptación. Los pilares empíricos se relacionan entre sí de manera que la trasparencia permite la inspección basándose en la idea que la inspección sin trasparencia resulta ser engañosa. De la misma forma, Scrum plantea que la inspección permite la adaptación, considerando que la inspección sin adaptación es inútil. La Tabla 2-2 describe los tres pilares empíricos de Scrum [10].

Pilar empírico	Descripción
Transparencia	Establece que todo el trabajo debe ser visible tanto para quienes lo realizan como para quienes lo reciban o tengan algún interés en el resultado. La idea detrás de establecer a la transparencia como un pilar del marco de trabajo se sostiene en que un proceso con poca o nula transparencia puede resultar en toma de decisiones que incrementen el riesgo o disminuyan el valor aportado. El objetivo de la transparencia es permitir la inspección.
Inspección	Establece que el avance del trabajo y el progreso hacia los objetivos propuestos deben ser revisados e inspeccionados con frecuencia y diligencia. Pretende minimizar las desviaciones y los elementos indeseables. Scrum favorece la inspección proponiendo cinco eventos (ver Tabla 2-3) que ocurren durante el ciclo iterativo. El objetivo de la inspección es permitir la adaptación.
Adaptación	Establece que, al identificar algún aspecto del proceso desviado fuera de los límites aceptables, entonces el producto y el proceso de desarrollo deben ajustarse. El ajuste debe realizarse lo antes posible para evitar una desviación mayor. La ausencia de elementos como la autoorganización del equipo podrían dificultar la adaptación, mientras que Scrum sugiere que un equipo debe adaptarse en el mismo momento en que genera cualquier tipo de aprendizaje durante la inspección.

Tabla 2-2: Pilares empíricos de scrum.

A un ciclo iterativo en Scrum se le llama *sprint.* Un *sprint* es uno de los cinco eventos definidos en Scrum y corresponde a un período determinado en que ocurrirán los otros cuatro eventos [10]. A continuación, la Tabla 2-3 describe los cinco eventos que propone Scrum para trabajar [10].

Evento	
Sprint	Es el evento principal de Scrum y se define como la instancia en que las ideas son transformadas en valor para el negocio. Corresponde a un ciclo iterativo de la metodología ágil. Tiene una duración determinada que no puede ser alterada. Un *sprint* comienza inmediatamente después de haber terminado el *sprint* anterior. Todo el trabajo necesario para alcanzar el objetivo del producto (ver Tabla 2-4) junto con los otros cuatro eventos de Scrum ocurren durante los *sprints*.
Sprint planning	Inicia el *sprint*. Produce un plan para ejecutar durante el transcurso de la iteración junto con un objetivo propuesto que debe ser cumplido al final del *sprint.* Aborda temas relacionados al valor que se desea agregar durante el *sprint*, cuánto trabajo se puede realizar en el tiempo acotado que dura el *sprint* y cómo se realizará ese trabajo.
Sprint Scrum	Ocurre todos los días hábiles del *sprint*. Pretende inspeccionar el progreso hacia el objetivo del *sprint* establecido en el *sprint planning.* Cada desarrollador debe indicar qué avances ha realizado, qué avances pretende realizar y si acaso tiene algún impedimento para acercarse al objetivo del *sprint* [2].
Sprint review	Pretende inspeccionar el incremento entregado como resultado del *sprint* y declarar si se logró alcanzar el objetivo propuesto en el *sprint planning*. Se inspecciona también el impacto de la entrega en el producto. Pueden surgir ajustes para prevenir desviaciones o para aprovechar oportunidades emergentes.
Sprint retrospective	Tiene como propósito planificar formas de aumentar la calidad y la efectividad del proceso de desarrollo y el trabajo que generan los incrementos del producto. Se inspecciona el *sprint* realizado con respecto a las personas, las interacciones, las estrategias elegidas y las herramientas utilizadas. Se debe identificar supuestos que expliquen las desviaciones para analizar sus causas. Concluye el *sprint*.

Tabla 2-3: Eventos de scrum.

Además, Scrum define tres artefactos para representar valor o trabajo. Están concebidos para maximizar la transparencia de la información importante [2] y [11]. Cada artefacto define un compromiso para poder asegurar que se entrega información estimulando la transparencia y manteniendo el foco en medir el progreso [10]. Los artefactos y sus respectivos compromisos están descritos en la Tabla 2-4 [10].

Artefacto	Descripción	Compromiso	Descripción
Product backlog	Lista ordenada por el *product owner* que contiene los ítems que representan el trabajo necesario realizar para incrementar el producto.	**Objetivo del producto**	Describe un estado futuro del producto. Se alcanza a largo plazo, luego de completar varios *sprints*.
Sprint backlog	Subconjunto de elementos del *product backlog* seleccionados para ser realizados durante el *sprint*. Se actualiza durante el *sprint* a medida que se adquiere aprendizaje y se adapta el plan.	**Objetivo del sprint**	Establecido durante el *sprint planning.* Define el único propósito del *sprint*.
Incremento	Avance concreto hacia el objetivo del producto que contiene valor. Cada incremento se suma a todos los incrementos anteriormente entregados.	**Definition of Done**	Descripción formal del estado del incremento al alcanzar las expectativas de calidad requeridas por el producto.

Tabla 2-4: Artefactos de scrum.

Dado que este trabajo enfrenta un problema complejo (ver sección 2.1.1.3) y que Scrum es la metodología ágil más usada en la industria del desarrollo de software, es que se eligió para la implantación del proceso de transformación digital presentada en esta tesis, ya que la probabilidad de poder encontrar profesionales con conocimientos de Scrum es mayor, además de que los casos de uso exitosos y la literatura disponible es de más fácil acceso y con más casos de distintos tipos de negocios aplicando este *framework*.

Un desafío común que deben enfrentar las organizaciones que implementan metodologías ágiles es resolver una manera de coordinar distintos equipos o distintas células trabajando de forma colaborativa. A pesar de que Scrum propone una forma de trabajar para un equipo, no resuelve el problema de tener que escalar o coordinar varios equipos que usan el mismo marco de trabajo. A pesar de que en la industria del desarrollo de software existe un consenso general respecto a los escenarios en que Scrum funciona bien o trae resultados deseables [7], no existe un acuerdo equivalente en referencia a cuál es la mejor forma o la forma más recomendable de hacer escalar Scrum. Existen distintas propuestas para hacer escalar scrum dentro de una organización como pueden ser *Nexus* o *SaFe* [7]. El marco de trabajo que utiliza este trabajo para realizar la transformación digital en la organización corresponde a un modelo a escala de Scrum y se llama *LeSS*.

2.1.3.2 Scrum a gran escala (LeSS)

Es la versión a gran escala de Scrum y pretende manejar varios equipos Scrum trabajando sobre un mismo producto. Este marco de trabajo extiende los principios propuestos por Scrum considerando pautas adicionales relacionadas al escalamiento, buscando que este ocurra sin sacrificar el propósito original de Scrum de aumentar la adaptabilidad [3].

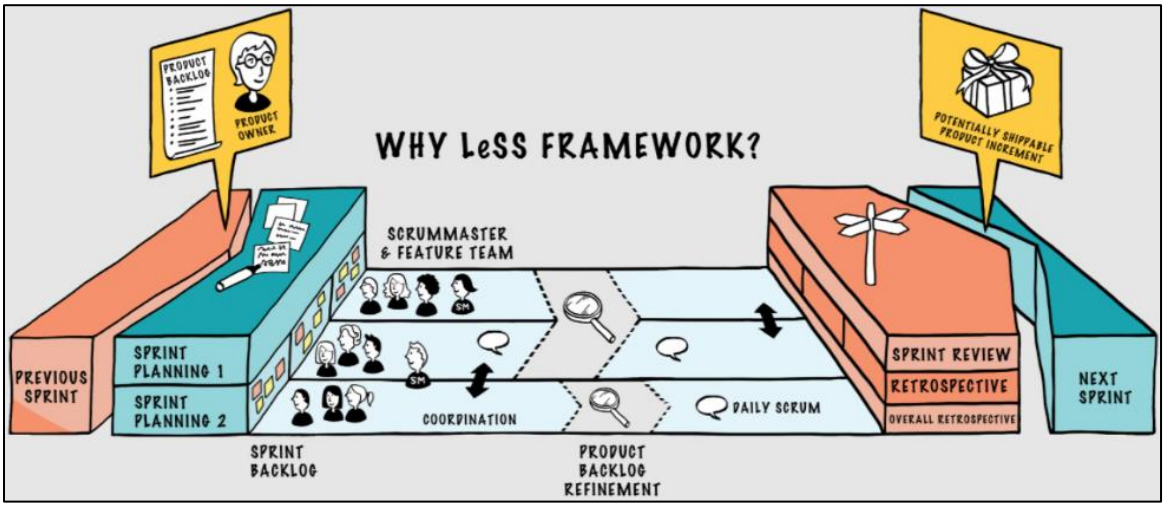


Figura 2-3: Propuesta de LeSS para hacer escalar scrum. Imagen extraída de https://less.works/ (21 mayo 2021)

LeSS pretende que los participantes adopten una filosofía minimalista, bajo los principios de reducir la complejidad de una organización. Se plantea la eliminación de soluciones complejas que no son necesarias y su transformación en alternativas más simples. LeSS impulsa la idea de mantener menos roles, menos administración y menos estructura a nivel empresarial [3]. Esto significa que los equipos necesitan ser autoorganizados y disponer de todas las capacidades necesarias para diseñar, implementar y desplegar el producto. La Figura 2-3 muestra las principales definiciones de LeSS, estas establecen que hay:

- Varios equipos Scrum, cada uno completo en capacidades y de carácter multifuncional.
- Un solo *product backlog,* de donde cada equipo Scrum obtendrá ítems para llevarlo a su *sprint backlog*.
- Un solo *sprint backlog* por cada equipo Scrum, trabajando de forma coordinada.
- Un solo *Definition of Done* para todos los equipos Scrum.
- Un incremento potencialmente entregable al final de cada *sprint*.
- Un solo *product owner*.
- Un solo *sprint* en curso para todos los equipos Scrum.

LeSS propone que en cada *sprint* todos los equipos Scrum trabajen para obtener un incremento entregable en común. Las principales diferencias entre Scrum sin escalar y LeSS se encuentran en la definición de sus eventos. En la Tabla 2-5 se presentan los eventos definidos por LeSS con su correspondiente descripción [3].

Evento	Descripción
Sprint planning 1	Ocurre con participación de todos los equipos Scrum y el *product owner.* Permite que, de forma autoorganizada, los equipos se dividan el trabajo que van a elegir desde el *product backlog*. Es la instancia en que los distintos equipos Scrum encuentran oportunidades para colaborar y compartir trabajo, especialmente en módulos comunes del software.
Sprint planning 2	Ocurre de forma separada en cada equipo Scrum. Es más parecido al *sprint planning* propuesto originalmente por Scrum, cuya salida es el plan que será ejecutado durante el *sprint*.
Daily scrum	Se mantiene la propuesta original de Scrum sin escalar. Ocurre de forma independiente en cada equipo Scrum, aunque permite que miembros de otros equipos estén presente para efectos de coordinación.
Product backlog refinement general	Es opcional. Ocurre con el *product owner* y con miembros de todos los equipos Scrum, aunque no es necesario que estén todos presentes. Pretende identificar qué equipos tienen más posibilidades de implementar qué trabajo seleccionado desde el *product backlog,* para que esos ítems sean refinados en profundidad a posteriori.
Product backlog refinement	Instancia en que un equipo Scrum refina los requerimientos de un ítem del *product backlog* con la intención de dejarlo listo para ser seleccionado en una próxima sesión de planificación.
Sprint review	Ocurre con el *product owner,* con los integrantes de todos los equipos Scrum y con cualquier cliente, usuario o parte interesada en el producto. Al igual que el *sprint review* que propone Scrum sin escalar, tiene como propósito inspeccionar el incremento entregado.
Retrospective general	Requiere la participación de miembros de todos los equipos Scrum, el *product owner* y todos los *scrum masters*. Tiene como propósito mejorar todo el proceso de desarrollo en general, en lugar de concentrarse en sólo un equipo.

Tabla 2-5: Eventos de LeSS.

Las reglas o pautas propuestas por LeSS son mínimas y no necesariamente van a indicar cuál es la mejor forma de aplicar el marco de trabajo. Para poder sugerir estrategias o tener una pauta de toma de decisiones, LeSS presenta diez principios que facilitan la implementación de Scrum a gran escala. La Tabla 2-6 describe los diez principios que facilitan la implementación de LeSS en una organización [12].

Principio	Descripción
LeSS es Scrum	LeSS no se trata de un “Scrum mejorado”, sino de aplicar los principios, elementos y el propósito que propone Scrum en un contexto de varios equipos o escalado a nivel empresarial.
Control empírico del proceso	La inspección y la adaptación del producto, del proceso y de las prácticas adoptadas sirven para crear una situación favorable para la organización en lugar de tener que estar siguiendo estrictamente una serie de reglas.

Principio	Descripción
Transparencia	Reforzando el pilar empírico originalmente propuesto por Scrum, la transparencia debe estar basada en elementos tangibles, verificables y comprobables contra el artefacto llamado *Definition of Done*.
Más con menos	LeSS favorece la generación de aprendizaje por medio de la experiencia por sobre la definición de procesos. LeSS invita a maximizar el valor entregado por los incrementos y a minimizar los desperdicios y gastos. LeSS propone generar más empoderamiento de parte de los equipos y generar más sentido de propósito por medio de trabajar con menos roles, menos artefactos y menos administradores.
Enfoque en la totalidad del producto	Se debe mantener un solo *product backlog,* un solo *product owner,* un solo potencial incremento entregable y un solo *sprint* independientemente de la cantidad de equipos que trabajen sobre el mismo producto.
Centrado en el cliente	Es fundamental poder identificar qué elementos representan valor y desperdicio para el cliente. Recibir retroalimentación de parte del usuario final y del cliente es importante para la concepción de los ítems del *backlog* y el desarrollo del producto. Todos los integrantes de los equipos Scrum deben entender que el trabajo realizado afecta directamente al cliente y al usuario del producto
Mejora continua	Se debe estar constantemente creando y entregando incrementos que no presenten defectos, que mejoren la experiencia de los usuarios. LeSS sugiere la realización de experimentos de bajo riesgo que apunten a encontrar oportunidades de mejora.
Reflexión sistémica	Se debe comprender exhaustivamente el sistema completo para poder optimizarlo. LeSS plantea que se deben evitar las optimizaciones locales (ver sección 1.1.2) y se debe favorecer el foco en la eficiencia y la productividad de las personas.
Reflexión magra	LeSS sugiere que la estructura de la organización favorezca un rol educativo de los gerentes y administradores para fomentar la reflexión sistemática y la práctica de la inspección empírica, sumando los pilares del respeto por las personas y la mejora continua como filosofía empresarial.
Teoría de colas	Aplicar la teoría de colas dentro del dominio de la investigación y desarrollo, para poder administrar el tamaño de las colas de trabajo por realizar, trabajo en curso y variabilidad del trabajo.

Tabla 2-6: Principios para facilitar la implementación de LeSS.

2.2 Herramientas de trabajo

Tanto Scrum como LeSS caben dentro de la categoría de metodologías ágiles de desarrollo de software y su principal objetivo dentro de una organización es lograr mayor adaptabilidad [9] y [10]. Particularmente LeSS, que es el marco de trabajo en que se basa esta tesis, plantea que la práctica de integración continua es esencial para poder escalar cualquier metodología ágil de desarrollo de software. A continuación, se van a revisar los conceptos de integración continua con sus correspondientes extensiones a las nociones de entrega continua y despliegue continuo. Luego, se presentan los fundamentos de computación en la nube y tecnologías específicas utilizadas durante la realización de este trabajo.

2.2.1 Integración continua

Conocida como CI, por sus siglas en inglés (*continuous integration*). Consiste en la práctica de automatizar la integración de cambios en el código fuente del software para que ocurran con la mayor frecuencia posible [13]. El concepto de integración continua hace referencia a la ejecución automática de rutinas que permitan reducir la probabilidad de integrar código que no adhiera a los lineamientos establecidos. Una práctica común es utilizar las herramientas de integración continua para ejecutar las acciones de [3]:

- Inspección del estilo de código.
- Análisis estático de código.
- Inspección de vulnerabilidades de bibliotecas que se usan como dependencias.
- Ejecución de pruebas unitarias de código.
- Construir paquete ejecutable.
- Ejecución de pruebas funcionales de módulos de software.
- Ejecución de pruebas *end-to-end* del producto.

El sentido de ejecutar de forma automática todos estos pasos es asegurar que cualquier modificación en el código fuente del software no tenga desviaciones en términos de calidad y de lineamientos técnicos al momento de ser integrado en los repositorios. Al proceso que ejecuta estos pasos de forma secuencial y que entrega *feedback* en caso de que alguno de ellos no cumpla con los estándares mínimos de aceptación se le llama "*pipeline* de integración continua".

2.2.2 Entrega continua

También llamada distribución continua o CD por sus siglas en inglés (*continuous delivery).* Es una práctica que extiende el concepto de CI (ver sección 2.2.1) cuyo fin es automatizar, además de la integración de cambios en el código fuente, la generación de versiones desplegables o implementables del software que pueden quedar en un repositorio para posteriormente ser usadas por algún responsable de despliegues o por el equipo de operaciones [13].

Si se cuenta con un repositorio de versiones desplegables del software, la práctica de entrega continua permite de automatizar la generación y publicación de versiones que pueden ser implementadas en un ambiente productivo en cualquier momento y según la necesidad del negocio. La acción de implementar o de desplegar el software, sin embargo, es ejecutada con intervención humana y no está automatizada. Generalmente la responsabilidad de implementar una versión desplegable le pertenece a un especialista de infraestructura o a un equipo de operaciones. Para tener CD es necesario tener CI, ya que se trata de una extensión de esta última.

2.2.3 Despliegue continuo

El paradigma de entrega continua automatiza el proceso de integrar cambios en el software y generar versiones implementables, pero el proceso de desplegar el software todavía requiere intervención humana. La práctica que cubre ese último eslabón es el despliegue continuo, también llamado implementación continua o en inglés *continuous deployment.* Es la extensión de la práctica de entrega continua (ver sección 2.2.2) y se encarga de que desplegar una versión entregable funcionando en un ambiente productivo no requiera intervención humana. La Figura 2-4 muestra un esquema con la representación gráfica de los elementos abarcados por los tres modelos mencionados en los últimos apartados.

El objetivo del despliegue continuo es minimizar el tiempo que pasa desde que un cambio en el código fuente del software es empujado a su repositorio y la versión entregable del software que lo integra esté funcionando en un entorno de producción [13]. Un sistema de despliegue continuo favorece el uso de metodologías ágiles (ver sección 2.1.3) porque en lugar de preparar una versión grande del producto, se facilitan las implementaciones de los pequeños incrementos que lo hacen crecer.

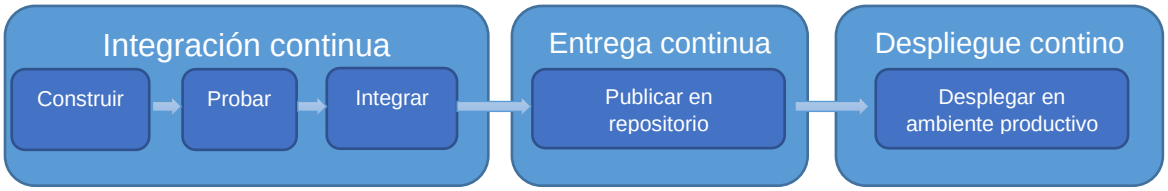


Figura 2-4: Extensión de integración continua hasta despliegue continuo.

Existen distintas herramientas tecnológicas concebidas para implementar sistemas de integración continua y/o de entrega continua como pueden ser Jenkins[5], Deploybot[6] o Buddy[7]. Particularmente en el contexto de este trabajo de magister, la herramienta utilizada fue Drone.io[8]. Esta plataforma provee una herramienta de integración continua que permite la automatización del flujo de entrega y despliegue de software a través del uso de contenedores Docker. El anexo E muestra un ejemplo de una definición de un *pipeline* de integración continua usando Drone.io.

2.2.4 Computación en la nube

Se refiere al acceso a un conjunto de recursos computacionales como lo pueden ser poder de cómputo, redes o espacio de almacenamiento disponibles bajo demanda y entregados por un proveedor a través de Internet sin que sea necesario administrar tales recursos por parte del usuario [14].

La idea detrás de utilizar un servicio de computación en la nube es bajar las barreras de implementación y administración del modelo tradicional de computación que

[5] https://www.jenkins.io/
[6] https://deploybot.com/
[7] https://buddy.works/
[8] https://www.drone.io/

requiere espacio físico, mantención de hardware, licencias de software y administración de sistemas. Usando computación en la nube se logra entonces reducir costos relacionados a mantención de infraestructura porque no es necesario contemplar la compra de hardware y licencias de uso de sistemas operativos [14]. Otro beneficio es la reducción de uso de espacio físico ya que todas las máquinas son mantenidas y administradas por el proveedor en sus propios centros de procesamiento de datos sin requerir ninguna dependencia física del usuario. La computación en la nube también tiene la ventaja de lograr centralizar la información, haciéndola accesible desde cualquier lugar con acceso a Internet y sin tener que depender de computadores específicos o de software autorizados para una cantidad limitada de máquinas. El escalado automático también es un punto atractivo para querer utilizar computación en la nube, ya que los proveedores generalmente cuentan con sistemas elásticos que permiten habilitar mayor o menor capacidad de cómputo y de recursos disponibles según la necesidad específica de la aplicación que se esté implementando [14]. La computación en la nube logró favorecer el trabajo remoto, ya que permite que los colaboradores de una organización puedan acceder a los sistemas sin necesariamente tener que estar físicamente presentes en las instalaciones de la empresa. La Figura 2-5 muestra un esquema para graficar el concepto de tener distintos servicios disponibles “en la nube”.

Figura 2-5: Computación en la nube. Imagen extraída de la cuenta del usuario Tumisu en la plataforma pixabay https://cdn.pixabay.com/photo/2018/04/21/02/11/iot-3337536_960_720.png (22 mayo 2021)

Existen varios proveedores de servicios de computación en la nube como lo pueden ser Google Cloud Platform, Microsoft Azure o Digital Ocean. Particularmente en este trabajo, se utilizó la plataforma de Amazon Web Services (AWS) como proveedor de computación en la nube. AWS ofrece distintos servicios sobre los cuales se pueden desplegar funcionalidades, aplicaciones y otros entregables. A continuación, se van a mencionar los principales servicios ofrecidos por AWS que fueron utilizados en este trabajo.

2.2.4.1 Elastic Compute Cloud (EC2)[9]

Permite crear instancias de máquinas virtuales sobre las cuales ejecutar software. Cada instancia puede ser configurada en términos de capacidades de memoria, almacenamiento y potencia de cómputo. EC2 permite al usuario utilizar instancias más pequeñas o poderosas según necesidad y cambiar su configuración sin necesidad de eliminarlas. Durante este trabajo se utilizó una máquina virtual de EC2 para mantener corriendo la herramienta de integración continua Drone.io (ver apartado 2.2.3).

2.2.4.2 Elastic Beanstalk[10]

Servicio que permite desplegar aplicaciones usando una serie de otros servicios dentro de AWS de tal manera de poder configurar un ambiente escalable, observable, con carga balanceada y privado o expuesto a Internet. Al configurar una aplicación a través de *Elastic Beanstalk* la plataforma a su vez configura otros recursos como EC2 para instancias que ejecutan el software, balanceadores de carga para repartir peticiones entre las instancias en ejecución, monitoreo de uso de memoria, entre otros. Todos los servicios REST desarrollados en este trabajo fueron desplegados usando *Elastic Beanstalk*.

2.2.4.3 Lambda[11]

Es una plataforma que permite desplegar código ejecutable bajo el paradigma de programación dirigida por eventos, de tal manera de poder tener una función escrita en algún lenguaje de programación soportado que sea ejecutada a partir de los eventos que se pueden definir y configurar en la misma plataforma. Las rutinas complementarias que se ejecutan a partir de determinados eventos en el flujo del proceso de negocio dentro del producto digital se desplegaron en *Lambda*.

2.2.4.4 S3 (Simple Storage Service)[12]

Servicio que ofrece almacenamiento por medio de espacios dedicados configurables. En el contexto de un producto digital, es de interés poder contar con almacenamiento para desplegar archivos de una aplicación web (*frontend*) o archivos que los usuarios puedan guardar o descargar a través del producto. Otro caso de uso puede ser la creación de respaldos de datos o *data lakes*. Todos los casos de uso que requieren recopilación de documentos y almacenamiento de archivos identificados durante este trabajo fueron implementados usando *S3*.

2.2.4.5 Route 53[13]

Es un sistema de nombres de dominio (DNS) de alta disponibilidad sobre el cual AWS ofrece la posibilidad de registrar subdominios de zonas hospedadas para poder redirigir a otros recursos utilizados dentro de la misma plataforma, como

[9] https://aws.amazon.com/es/ec2/
[10] https://aws.amazon.com/es/elasticbeanstalk/
[11] https://aws.amazon.com/es/lambda/
[12] https://aws.amazon.com/es/s3/
[13] https://aws.amazon.com/es/route53/

pueden ser aplicaciones desplegadas en *Elastic Beanstalk* o servidores creados en EC2. Route 53 también permite solicitar la creación de certificados digitales asociados a dominios electrónicos que tengan la firma de AWS como autoridad certificadora válida, este servicio es importante al exponer aplicaciones y servicios web usando protocolo HTTPS. En este trabajo se usó el servicio que provee *route 53* para administrar los dominios electrónicos de Internet.

2.2.4.6 API Gateway[14]

Permite configurar una interfaz intermedia entre el cliente que consume un servicio específico y el servicio expuesto. A través del *API Gateway* se pueden configurar las rutas autorizadas para distintos clientes y exponer un único punto de entrada para los clientes que consumen los servicios correspondientes a cada ruta. Al tener distintos servicios expuestos en distintas rutas, el API Gateway puede concentrar la responsabilidad de autorizar o rechazar llamadas según los recursos específicos a lo que está tratando de acceder el cliente HTTP. Usar un *API Gateway* favorece el escalado de una arquitectura para sostenerse en distintos servicios web según responsabilidades específicas como lo puede ser, por ejemplo, el modelo basado en microservicios. La Figura 2-6 muestra un esquema que grafica las interacciones entre el cliente HTTP y los servicios web pasando por el *API Gateway*, que maneja el enrutamiento y la autorización de la petición.

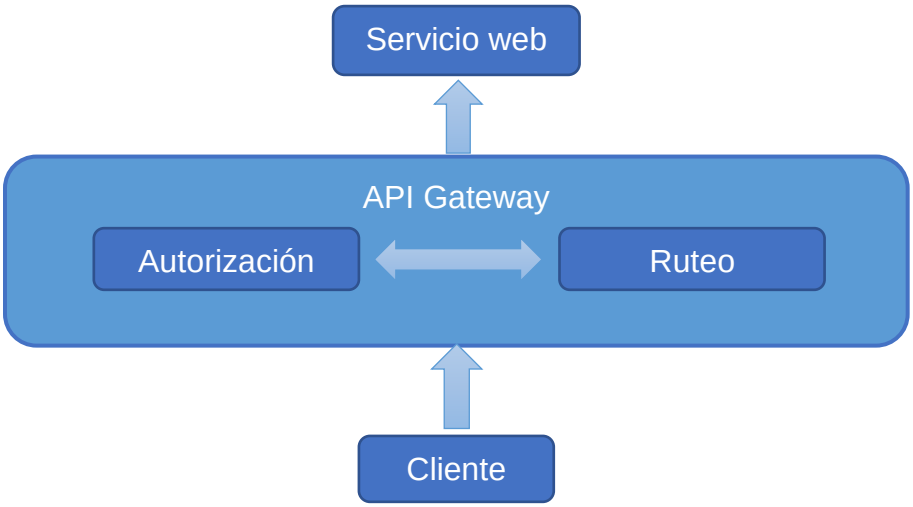


Figura 2-6: API Gateway.

Durante este trabajo, se utilizó el servicio que ofrece *AWS API Gateway* para administrar los distintos *endpoints* de los microservicios del producto digital, además de poder centralizar la autorización y validación de las peticiones recibidas desde Internet.

[14] https://aws.amazon.com/es/api-gateway/

2.2.4.7 MongoDB Atlas[15]

Es una plataforma que usa los principios de computación en la nube y ofrece el uso de una base de datos como servicio. El sentido de usar bases de datos en la nube es abstraerse de la complejidad de desplegar, administrar y mantener hardware. *MongoDB* es una base de datos NoSQL y orientada a documentos. El paradigma consiste en tener colecciones de documentos que pueden ser representados como objetos JSON. Cada documento cuenta con un identificador único que permite hacer relaciones con documentos de otras colecciones. *MongoDB Atlas* permite contar con una implementación de *MongoDB* como servicio.

La base de datos utilizada en el producto digital desarrollado durante este trabajo se encuentra alojada en AWS a través de los servicios que ofrece *MongoDB Atlas*. Esto permite que se pueda crear un acceso por la red interna de AWS entre los servicios del producto digital y la base de datos, sin tener que comunicarse a través de Internet.

2.2.5 Node.js[16]

Es un entorno de ejecución compatible con distintas plataformas basado en JavaScript concebido bajo el enfoque de escribir programas que se ejecuten fuera del explorador de Internet, como lo pueden ser los servicios web (ver anexo A). Node.js tiene una arquitectura dirigida por eventos capaz de manejar instrucciones asíncronas [15]. Este tipo de características favorecen los aspectos de escalabilidad y tasas de transferencia efectiva o volumen de datos que circulan en el sistema. Tanto *Elastic Beanstalk* (ver sección 2.2.4.2) como *Lambda* (ver sección 2.2.4.3) ofrecen entornos de ejecución compatibles con Node.js.

Una razón poderosa para elegir Node.js como tecnología de desarrollo software es que JavaScript se ha convertido en el lenguaje de programación dominante dentro de la industria de aplicaciones web, eso significa que es más fácil encontrar desarrolladores con experiencia en la industria que usen este lenguaje. Node.js permite usar el mismo lenguaje tanto del lado del cliente web como del lado del servidor.

Además, existen tecnologías desarrolladas usando Node.js como base que exponen *frameworks* de desarrollo software para facilitar el desarrollo de tipos específicos de aplicaciones. A continuación, se van a revisar dos marcos de trabajo de desarrollo de software concebidos para construir aplicaciones web (*frontend*) y servicios web (*backend*) correspondientemente.

[15] https://www.mongodb.com/atlas/database
[16] https://nodejs.org/en/about/

2.2.6 VueJS[17]

Es un *framework* de desarrollo software concebido para escribir y construir interfaces de usuario de aplicaciones de una sola página (SPA). Tiene un fuerte carácter adaptativo favoreciendo el renderizado declarativo. Eso significa que los componentes definidos en tiempo de desarrollo (que son renderizadas por el explorador de Internet en tiempo de ejecución), reaccionan según los valores que toman las variables declaradas. VueJS soporta características avanzadas como ruteo y administración de estados por medio de bibliotecas adicionales que funcionan como *plugins* del mismo *framework*.

VueJS tiene una característica curva de aprendizaje más amigable o suave que otros *frameworks* SPA como Angular o React, además de tener una mejor documentación. Con VueJS no es necesario usar TypeScript ya que soporta código escrito en JavaScript [16].

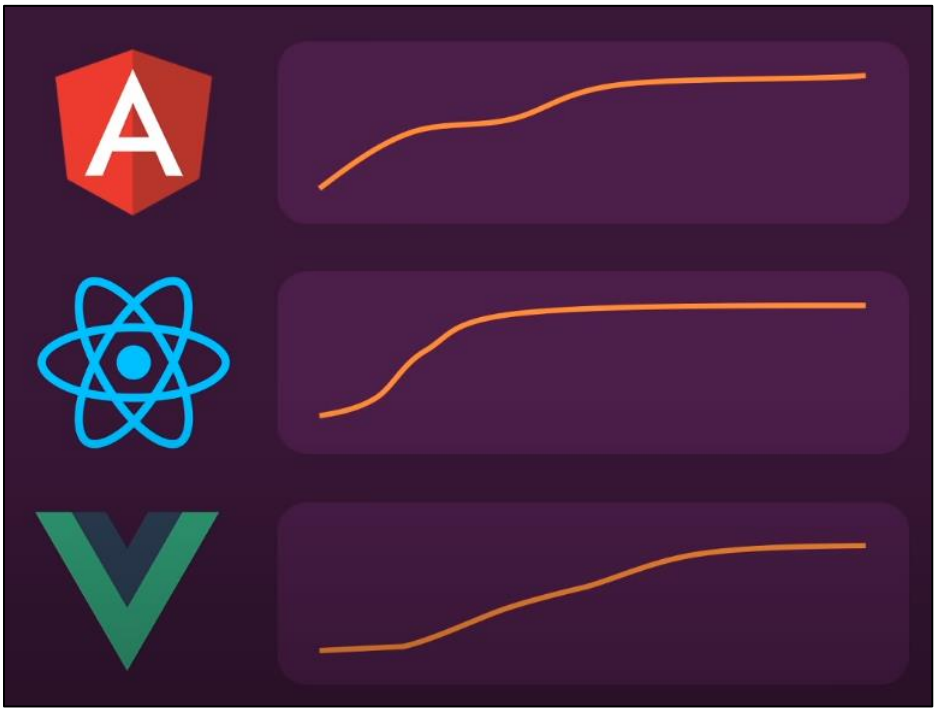


Figura 2-7: Formas de las curvas de aprendizaje de Angular, React y VueJS. Imagen extraída de la cuenta de Academind en YouTube https://www.youtube.com/watch?v=lYWYWyX04JI (22 mayo 2021).

2.2.7 NestJS

Es un *framework* de desarrollo software basado en Node.js concebido para construir aplicaciones del lado del servidor. Usa TypeScript como lenguaje de programación y se puede pensar como un *wrapper* que a su vez envuelve otro *framework* para exponer servicios por medio de protocolo HTTP llamado Express.

Es común que distintos desarrolladores, según el patrón de diseño que elijan para desarrollar servicios web sobre Node.js tengan distintos entendimientos sobre las

[17] https://vuejs.org/

responsabilidades de los conceptos de *middleware*, servicio, o controlador, entre otros componentes. La ventaja que aporta NestJS es fijar un patrón de diseño específico motivado por su documentación, que termina con la discusión sobre las responsabilidades que deben tener los distintos componentes, además de propiciar la definición del uso de clases y objetos tipados. Estas características disminuyen o limitan las posibilidades de desviaciones de buenas prácticas al momento de escribir código. La propuesta de NestJS con respecto a las componentes modeladas en un repositorio de código están descritas en la Tabla 2-7.

Componente	Descripción
Module	Define un recurso específico administrado por el servicio web. Cada recurso está relacionado a la ruta que se expone en cada *endpoint* que recibe peticiones HTTP.
Controller	Contiene los métodos dedicados a capturar las peticiones del cliente HTTP e identificar sus variables, que pueden ser la búsqueda del URL, los parámetros del *endpoint* expuesto o los valores del cuerpo de la petición.
Service	Contiene los métodos definidos para aplicar la lógica de negocio necesaria de implementar. Un método definido dentro de un *service* puede ser invocado desde un método definido en un *controller* o desde otro método definido en un *service*.
Repository	Contiene los métodos definidos para manejar la lógica que manipulan recursos externos al servicio web, como lo pueden ser bases de datos o gestión de archivos en la nube. Un método definido en un *repository* puede ser invocado desde un método definido en un *service*.

Tabla 2-7: Propuesta de NestJS sobre responsabilidades de las componentes.

2.3 Resumen

En este capítulo se revisaron los conceptos que motivan las propuestas de metodologías ágiles, como lo es la definición de un proyecto complejo. También se presentaron los lineamientos propuestos por Scrum, que es el *framework* más usado en la industria del software y además se tocaron los principios que establece LeSS, para escalar la filosofía de trabajo propuesta por Scrum. Además, se explicaron los conceptos de integración continua, computación en la nube, y las herramientas modernas actualmente utilizadas en la industria para implementar productos digitales. Estos conceptos y herramientas fueron utilizados durante este trabajo y serán referenciados durante los próximos capítulos.

3 INTERVENCIÓN REALIZADA

En este capítulo se abordará el trabajo realizado dentro de la organización intervenida, distinguiendo dos principales secciones, la primera dedicada a los tópicos de diagnóstico y entendimiento del proceso de negocios, y la segunda dedicada a revisar los cambios en la forma de trabajar del equipo de desarrollo y su forma de interactuar con las demás partes interesadas. El diseño del producto digital en términos de tecnologías, arquitectura e infraestructura es revisado en el capítulo 4.

3.1 Diagnóstico y levantamiento inicial

Antes de proponer un nuevo proceso de negocios es necesario realizar un levantamiento para poder entender el proceso actual o el flujo de trabajo antes de la intervención. El propósito de este ejercicio es identificar claramente las oportunidades de mejora inmediatas y aquellos elementos que no son necesarios o que no entregan valor para poder eliminarlos o al menos comenzar a prescindir de ellos. Al saber qué pasos del flujo de trabajo son prescindibles o eliminables, se favorece el planteamiento de una propuesta más simple lo cual es compatible con el principio de LeSS que plantea que “más es menos” (ver Tabla 2-6).

El proceso de otorgamiento de un crédito hipotecario se divide en cuatro etapas secuenciales en la que cada una tiene una entrada y genera una salida sobre la cual trabaja la etapa siguiente. A continuación, se explicarán las cuatro etapas del proceso de negocios cuyo diagrama completo se puede ver en el anexo B.

3.1.1 Simulación

A la persona que entra en esta primera etapa se le llama *lead* y se refiere a una persona que se acercó a la organización para realizar una simulación de evaluación financiera para otorgamiento de crédito hipotecario. Aquellos *leads* que completaron una simulación son posteriormente administrados por ejecutivos del *backoffice* y se les llama prospectos. La simulación es un cálculo matemático que asume variables de riesgo comunes tomando en cuenta entradas como el valor de la propiedad, el porcentaje del pie y los ingresos mensuales del *lead.*

La página web de la empresa está desplegada sobre una plantilla WordPress alojada en una máquina virtual EC2 de AWS (ver sección 2.2.4.1). Los datos para realizar la simulación son recogidos a través de un formulario simple alojado por una plataforma web que se llama *JotForm*[18]. *JotForm* es entonces una herramienta externa que permite crear formularios a través de una interfaz de usuario amigable y conectar los datos recopilados a algún otro sistema compatible, por medio de plantillas utilizables e integrables en distintas tecnologías, incluyendo WordPress. Los datos recogidos por el formulario del simulador son enviados a un servicio de

[18] https://www.jotform.com/

otro proveedor llamado *Zapier*[19], que permite integrar distintas aplicaciones para recibir datos a partir de alguna aplicación, procesarlos, y enviarlos a otra aplicación. *Zappier* recibe las variables enviadas por *JotForm* y realiza el cálculo matemático correspondiente a la simulación para obtener un valor tentativo de una oferta. Todas las simulaciones originadas en *JotForm*, que luego son calculadas en *Zappier* son finalmente enviadas a otra plataforma web llamada *Pipedrive*[20]. *Pipedrive* es una aplicación pensada para gestión de relación con el cliente (CRM) que almacena los datos de las simulaciones y permite a los ejecutivos del *backoffice* realizar seguimientos o comenzar a trabajar con los *leads* para concretar una evaluación financiera.

La Figura 3-1 grafica el proceso de simulación, cuyos porcentajes de conversión, como se expuso en la sección 1.1.1 (particularmente en la Tabla 1-1), son obtenidos y declarados por el equipo de marketing, ya que maneja la responsabilidad de medición de *funnels* y generación de campañas hasta antes de la intervención realizada en este trabajo.

Las simulaciones pueden ocurrir directamente en la página web alojada en WordPress (círculo verde en el lado superior izquierdo de la Figura 3-1) o también por medio de ejecutivos inmobiliarios que atienden presencialmente a clientes en las salas de ventas de los proyectos inmobiliarios (círculo violeta en lado inferior izquierdo de la Figura 3-1). Los *leads* están distribuidos 84% a través de la página web y 16% en el canal inmobiliario. El simulador en los canales inmobiliarios está expuesto a través de un tótem con pantalla táctil que expone el mismo formulario de JotForm. Un 83% de los *leads* del canal web se explican debido a campañas de marketing, mientras que el 7% restante llega directamente luego de haber visitado el sitio de la organización.

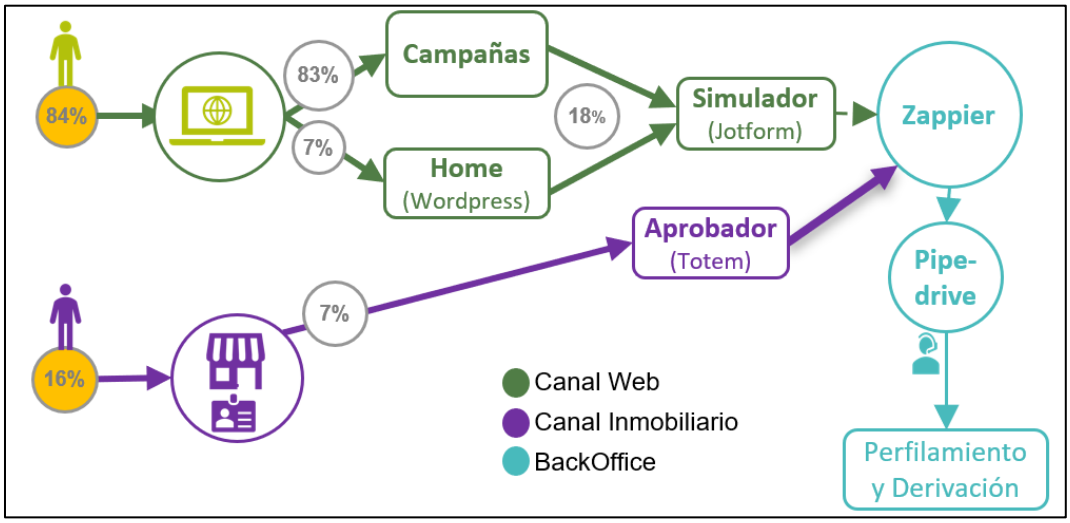


Figura 3-1: Etapa de simulación antes de la intervención.

[19] https://zapier.com/
[20] https://www.pipedrive.com/es

El 18% de los *leads* del canal web realizan una simulación que termina con datos administrados en Pipedrive, mientras que sólo un 7% de los *leads* del canal inmobiliario realizan una simulación que genere prospectos. De esta manera, sólo el 16% de los *leads* son transformados en prospectos. Este número es relevante porque es el punto de referencia para poder comparar cómo cambia la conversión del simulador luego de realizar la intervención realizada en este trabajo.

3.1.2 Evaluación

La Figura 3-2 grafica el proceso de evaluación. Los *leads* que fueron generados por las simulaciones son administrados por los ejecutivos del *backoffice* en Pipedrive de tal manera de tener un seguimiento de la documentación recopilada y completar el perfilamiento de cada *lead* (perfilamiento y derivación en Figura 3-2). El proceso de recopilación de documentación tarda un promedio de cuatro días dependiendo de las interacciones que ocurran entre la persona que está interesada en el crédito hipotecario y el ejecutivo que lleva su caso. Existe también la posibilidad de que una persona sea un *lead* sin haber pasado por la etapa de simulación, siendo este el caso de los clientes de las salas de proyectos inmobiliarios que entran directamente en contacto con un ejecutivo comercial sin usar ni el tótem, ni la página web (Trazado violeta en la Figura 3-2).

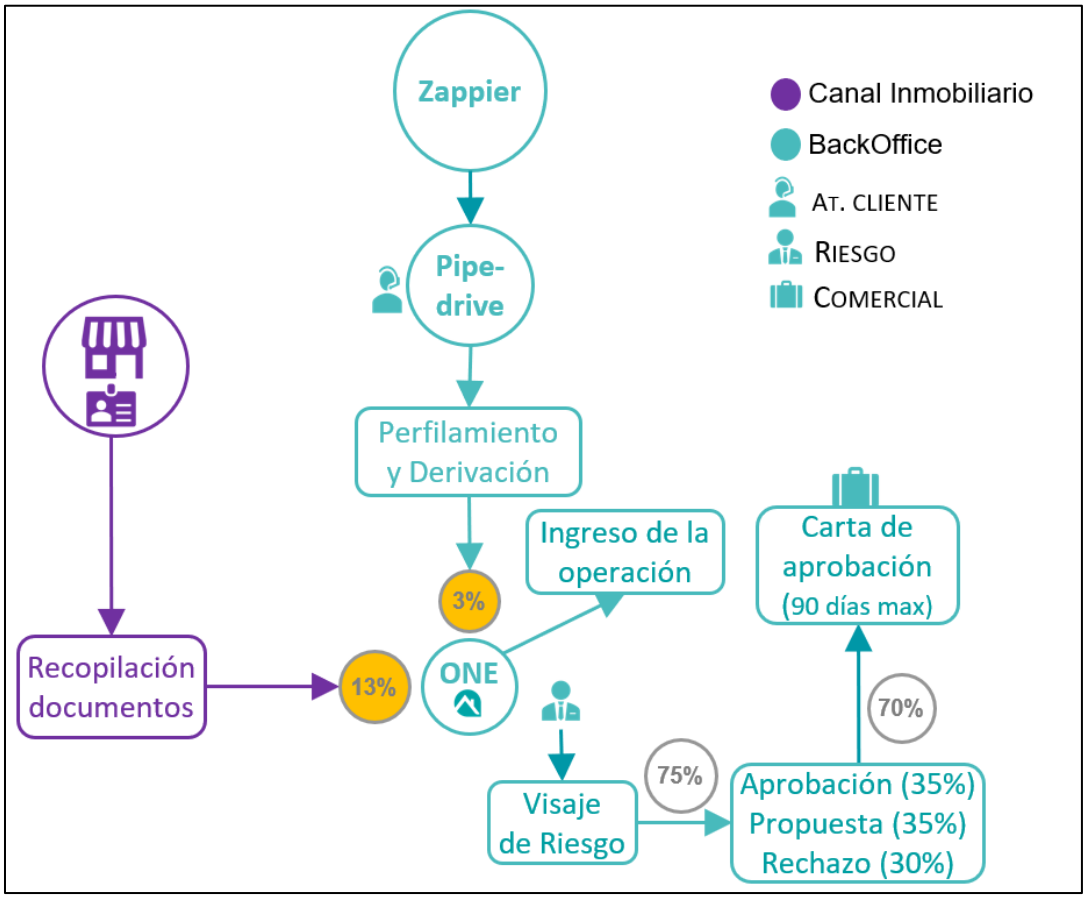


Figura 3-2: Etapa de evaluación antes de la intervención.

Una vez que se ha terminado de completar la documentación del *lead*, sus datos son ingresados al ERP (ver sección 1.1.1) llamado ONE (circulo central inferior en la Figura 3-2). En ONE se genera una potencial operación. Una operación es la entidad administrada por la mutuaria, que por el lado del deudor es representado por el crédito hipotecario otorgado y por el lado del inversionista es representada por el mutuo de inversión que retorna utilidades a medida que se van cobrando los dividendos. Luego de haber generado la potencial operación y de tener toda la documentación necesaria, los datos son enviados al departamento de riesgo (ver sección 1.1.1) para realizar verificaciones de la validez de los documentos y datos entregados. El proceso de verificación se llama visaje de riesgo (cuadro inferior en la Figura 3-2) y termina la etapa de evaluación, trasformando al *lead* en un prospecto o rechazando al *lead*. El visaje de riesgo puede terminar en tres escenarios distintos. Aprobación es el caso en que el lead se trasforma en un prospecto con una oferta igual a la que este estaba solicitando. Propuesta es el caso en que el prospecto recibe una oferta parcial o por una cantidad menor a la que originalmente estaba postulando. Rechazo es el caso en que el *lead* no logra transformarse en un prospecto. El prospecto comienza a existir en el momento en que recibe la carta de aprobación de parte de la organización; esta carta contiene la oferta de crédito hipotecario que le será otorgado en caso de aceptarla. La carta de aprobación tiene una validez de 90 días y el prospecto debe devolverla con su firma para considerarse aceptada. Como indica la Figura 3-2, sólo un 30% de los casos que llegan al visaje de riesgo son rechazados, mientras el 70% de los casos que reciben una carta de aprobación están distribuidos en igual proporción entre aquellos que se les aprueba la totalidad de la solicitud y aquellos que reciben una propuesta alternativa. De esta forma, la conversión de *leads* que lograron convertirse en prospectos y luego se transformaron en aprobados (en totalidad o con propuesta alternativa) es del 11%.

3.1.3 Firma de escritura

Una persona que terminó el proceso de evaluación de forma exitosa pasa de ser un prospecto a ser un aprobado. Un aprobado se convierte en un cliente en el momento de firmar la escritura. Debido a las restricciones impuestas por la normativa vigente, es necesario que la firma ocurra de forma presencial y con el respaldo de un notario. Esto introduce inmediatamente una serie de barreras y discontinuidades en el proceso. Una vez que la persona ha aceptado la carta de aprobación, se realiza el trámite de escrituración por medio de una notaría de tal manera que el aprobado pueda acercarse a firmar. Un problema común es que luego de que el cliente haya firmado, no existe un mecanismo para que la notaría notifique a la mutuaria oportunamente para continuar el proceso. Existen casos en que se ha tenido que realizar el proceso de escrituración desde el principio, debido a que no se recibió notificación de parte de la notaría o las partes involucradas y la tasa acordada deja de estar vigente o pasa a ser reemplazada en un anuncio nuevo de la CMF (ver capítulo 1).

3.1.4 Perfeccionamiento

La etapa de perfeccionamiento es la última del *costumer journey* (ver anexo B) y hace referencia a la administración de la operación existente a partir del momento en que se realiza el desembolso. La administración de la operación incluye trámites como registro en el conservador de bienes raíces, cálculo y cobro de gastos operacionales, endoso del mutuo al inversionista correspondiente, cálculo de intereses, cobro de dividendos y remesa a inversionistas.

3.1.5 Oportunidades inmediatas de mejora

Luego de haber realizado el primer levantamiento, se identificaron los puntos más fáciles de intervenir para realizar mejoras inmediatas a través de la incorporación de tecnologías que permitan automatizar los procesos del negocio. Las etapas que tienen menos burocracia y que no están directamente afectadas por las imposiciones de la normativa vigente están al principio y al final del *customer journey*, en las etapas de simulación y perfeccionamiento (ver anexo B). Las oportunidades inmediatas de mejora son también candidatos para representar requerimientos de alta prioridad para comenzar a desarrollar el producto digital de la organización.

3.1.5.1 Simulación

El proceso de simulación es un cálculo matemático que depende de variables que son entregadas por el *lead* (ver sección 3.1.1) y cuyas restricciones no implican trámites burocráticos o firmas físicas. Eso significa que el proceso de simulación puede ser mejorado a través de la unificación en una sola plataforma en lugar de tener involucrados distintos proveedores como WordPress, JotForm, Zappier y Pipedrive.

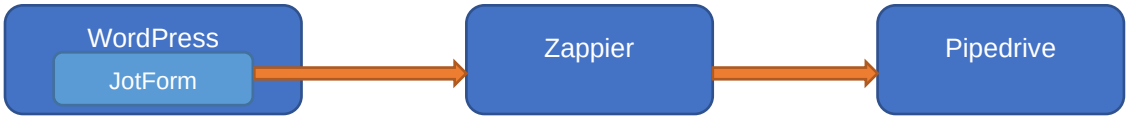


Figura 3-3: Simulador antes de la intervención.

La Figura 3-3 ilustra las interacciones entre distintas plataformas involucradas en el simulador antes de la intervención. Dentro del sitio web que está alojado en WordPress, se utiliza un *plug-in* para integrar un formulario construido en JotForm, que al ser completado gatilla un proceso en Zappier que aplica la fórmula matemática, alerta a los ejecutivos comerciales y también tiene activada la integración contra Pipedrive para crear la potencial operación a la que se le realiza el seguimiento. Un esquema como este no es deseable porque involucra más esfuerzos de mantención, mayores costos de facturación y más posibles puntos de falla.

Durante este trabajo se implementó una mejora en el simulador que lo hace pasar a ser parte del producto digital de la organización, sin depender de proveedores externos y sin tener que implementar integraciones tecnológicas innecesarias. En

el capítulo 4 se abordan los detalles técnicos para incluir el simulador dentro del producto digital.

3.1.5.2 Recaudación de dividendos

Uno de los procesos que ocurren durante la etapa de perfeccionamiento es el cobro de los dividendos a los clientes. Antes de la intervención, la recaudación se encontraba a cargo de la administración primaria (ver Tabla 1-1) y externalizada a través de un proveedor que ofrece servicios de pago por caja al público, a pesar de que la operación funciona correctamente en términos de cobros de intereses y remesas a la organización, representa uno de los costos más altos que tiene la empresa. En un contexto en que se pretende desarrollar un producto digital, la opción de pago por Internet es un requerimiento inmediato o evidente, que además puede representar una baja importante en los costos de operación al momento de prescindir del proveedor externo.

3.2 Proceso de desarrollo

Los cambios más grandes y que representan más resistencia al cambio dentro de la organización son los lineamientos introducidos por el marco de trabajo adoptado. Antes de la intervención el equipo de desarrollo funcionaba como servicio (ver sección 1.1.1), atendiendo pedidos de las distintas áreas de la organización, respondiendo a un modelo vertical, como el que muestra el lado izquierdo de la Figura 3-4. Ese paradigma cambia completamente de tal manera que, en lugar de estar sirviendo a otros departamentos, el equipo de desarrollo más bien se va a servir de las demás áreas, respondiendo a un modelo horizontal de organización, como el que se muestra en el lado derecho de la Figura 3-4.

Para favorecer la capacidad de adaptarse de la empresa, LeSS plantea que el equipo de desarrollo debe proponer los requerimientos y que estos sean motivados en los procesos de negocio y los dolores que tengan las partes involucradas [3]. Para poder determinar qué es necesario implementar y además definir la prioridad que tienen los requerimientos, el equipo de desarrollo debe ser capaz de traducir los dolores, inquietudes y necesidades de los clientes y de las distintas áreas de la organización en requerimientos que efectivamente favorezcan los procesos del negocio, faciliten el trabajo de los operadores y agreguen valor a la organización. Para poder lograrlo, es necesario que el equipo de desarrollo esté en constante contacto con las demás áreas de la organización y con los clientes, entendiendo sus procesos, recogiendo necesidades e identificando oportunidades de mejora. De la misma forma, las demás áreas deben estar disponibles para entregar datos relevantes al equipo de desarrollo y servirle como fuente de información durante el proceso de identificación de mejoras y proposición de requerimientos.

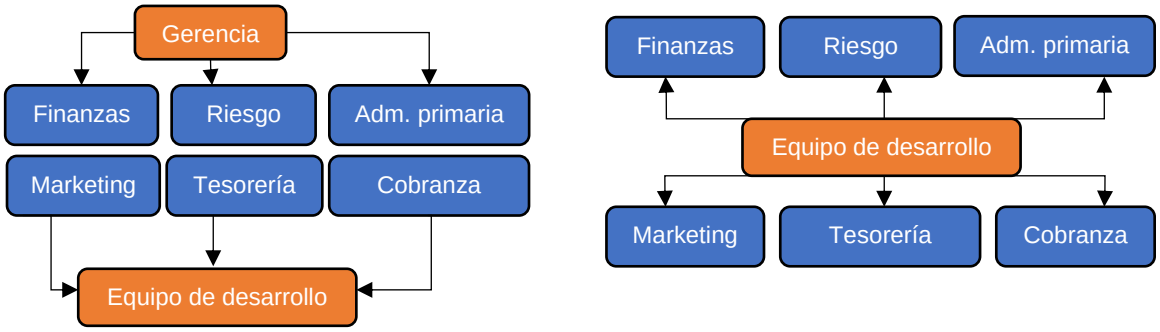


Figura 3-4: Modelo vertical (izquierda) y modelo horizontal (derecha).

Este cambio es complejo porque involucra dos aristas de las cuales hay que hacerse cargo, una de carácter técnico, y otra de naturaleza cultural [8]. La primera tiene que ver con el perfil de los profesionales que conforman el equipo de desarrollo y con sus habilidades técnicas o capacidades de moverse en distintas áreas del conocimiento, mientras que la segunda está relacionada a la disposición que tengan tanto los integrantes del equipo de desarrollo como el resto de los colaboradores de la empresa en efectivamente adoptar las nuevas filosofías de trabajo.

A continuación, se presenta un análisis respecto a las habilidades que debe tener el equipo de desarrollo. Luego se abordan elementos relacionados a la adopción del marco de trabajo. Después se menciona un punto importante referente a la forma en que el equipo de desarrollo debe balancear sus esfuerzos entre propuestas nuevas y la operación del negocio. Finalmente, se presentan los elementos del proceso de desarrollo que fueron automatizados por la implementación de un *pipeline* de despliegue continuo.

3.2.1 Reclutamiento y completitud del equipo de desarrollo

Desde la perspectiva técnica, el equipo de desarrollo pasa a tener muchas más responsabilidades que antes porque no sólo debe hacerse cargo de mantener el producto digital, sino que también debe asegurar la continuidad del negocio y facilitar los procesos internos sobre los cuales se sostiene. Esto significa que el equipo de desarrollo necesita abarcar distintas disciplinas que complementen el desarrollo de software, como pueden ser la estrategia e inteligencia de negocio, la experiencia de usuario, el diseño gráfico, la administración de sistemas, la seguridad computacional, el aseguramiento de la calidad. Puesto de otra forma, el equipo de desarrollo necesita estar conformado por distintos profesionales que puedan disponer de sus conocimientos y habilidades en distintas áreas del conocimiento para poder mantener el producto digital, involucrando identificación de necesidades, propuesta de requerimientos, diseño de soluciones, codificación e implementación del software, despliegue del producto, canalización y resolución de incidencias, y todos los elementos necesarios para poder responder rápidamente ante cambios en el entorno de la empresa. La estrategia adoptada para dotar al equipo de desarrollo la completitud necesaria fue la de ejecutar un plan de reclutamiento durante seis semanas antes de comenzar el desarrollo del producto digital.

El objetivo de completar el equipo de desarrollo con especialistas de distintas áreas del conocimiento es incorporar profesionales que brinden habilidades relacionadas a distintas disciplinas involucradas en el mantenimiento del software y de esa forma seguir el principio de LeSS de enfoque en la totalidad del producto (ver Tabla 2-6). El plan de reclutamiento tuvo como objetivo formar un equipo de 12 personas, de manera tal que se puedan tener dos células de seis personas, cada una a cargo de una de las etapas del *customer journey* de simulación y de perfeccionamiento (ver anexo B) y entonces poder aprovechar las oportunidades inmediatas de mejora (ver sección 3.1.5).

Cada célula quedará entonces, en primera instancia, constituida por un profesional de la experiencia de usuario (puede ser diseñador gráfico u otro), un ingeniero comercial, un ingeniero de software y tres desarrolladores especialistas de *frontend* o *backend*. Si bien uno de los problemas a los que apunta solucionar el presente trabajo es la aparición de optimizaciones locales (ver sección 1.1.2) a través del principio del principio de reflexión sistémica propuesto por LeSS (ver Tabla 2-6), de todas maneras es necesario proveer al equipo de desarrollo con distintos profesionales que dejen a disposición sus conocimientos de manera transversal para cumplir con las responsabilidades del equipo de desarrollo. La idea general entonces es que, a pesar de que exista un especialista en un área específica (lo cual es incluso algo natural), hay que facilitar constantemente que tal conocimiento no se monopolice, por ejemplo, haciendo que los especialistas de estrategia de negocio participen de las sesiones de *pair programming* y vayan adquiriendo habilidades técnicas transversalmente a medida que se van cumpliendo más iteraciones.

3.2.1.1 Medición del desempeño del equipo de desarrollo

Antes de la intervención, el desempeño de los desarrolladores se medía según el nivel de satisfacción de los departamentos que realizaban las solicitudes al equipo de desarrollo. Dado que el equipo de desarrollo debe hacerse cargo del completo ciclo de vida del producto digital y por lo tanto asumir más responsabilidades en términos de gobierno de negocio y planteamiento de requerimientos, será necesario también cambiar la forma de medir el desempeño de los integrantes del equipo de desarrollo.

La nueva forma de medir el desempeño del equipo de desarrollo responderá directamente a los indicadores globales de la empresa y por lo tanto al impacto que están teniendo las entregas del equipo de desarrollo sobre el desempeño del negocio. De esta forma, la célula que está a cargo de la etapa de simulación será medida por la cantidad de simulaciones que se realizan y la conversión que estas presentan con respecto a la cantidad de *leads* que efectivamente se transformar en prospectos (ver sección 3.1.1). El desempeño de la célula responsable de la etapa de perfeccionamiento será medido de igual manera, según la tasa de mora o el porcentaje de clientes morosos y la conversión de la recaudación de dividendos.

3.2.2 Adopción de LeSS

Desde el punto de vista de la cultura de trabajo, el mayor impacto provocado por la intervención realizada es que la toma de decisiones respecto a las características del producto digital ahora depende directamente del equipo de desarrollo (que se puede apoyar en las demás áreas de la organización). En lugar de que las partes interesadas estén realizando solicitudes según sus intereses particulares, es el equipo de desarrollo el que debe estudiar la operación de cada área y determinar qué es lo mejor para el sistema y por lo tanto qué elementos serán reflejados en requerimientos. Esta forma de trabajo implica un cambio cultural importante, tanto para el equipo de desarrollo como para las demás áreas de la organización y para que el cambio pueda ser gestionado de forma correcta es necesario que se proteja el principio de transparencia que propone LeSS (ver Tabla 2-6), o sea que la constante comunicación es fundamental, tanto para sincerar dolores de parte de las demás áreas como comunicar prioridades y explicar características propuesta e implementadas por parte del equipo de producto.

El buen desempeño del negocio va a depender tanto de las definiciones que nacen dentro del equipo de desarrollo como del correcto uso del producto digital y correcta ejecución de los procesos de parte de las demás áreas de la compañía. LeSS no puede funcionar si no hay un compromiso transversal de toda la organización.

El ejercicio que debe hacer el equipo de desarrollo al definir un proceso de negocio o al proponer un requerimiento para el producto digital involucra una comprensión profunda de todas las aristas involucradas como lo son las interacciones de los operadores, las interacciones de los clientes, o las dependencias con proveedores externos. Eso significa que el equipo de desarrollo no puede limitarse solamente a desarrollar y desplegar software, sino que debe asegurar la correcta continuidad del negocio. En este trabajo, el producto digital parte en primera instancia con características asociadas a los procesos de simulación y de recaudación de dividendos, dejando las etapas intermedias del *customer journey* fuera de las primeras iteraciones o d ellos primeros incrementos del software. A pesar de que el software se encuentre en sus primeras versiones y todavía no pueda atender todos los procesos del negocio o no pueda gestionar completamente el *customer journey*, de todas maneras, el equipo de desarrollo debe facilitar los medios necesarios para que las distintas áreas puedan seguir operando, ya sea asumiendo carga operativa o bien habilitando herramientas provisorias que en el futuro sean reemplazadas por las características del producto digital que se vayan implementando.

3.2.3 Carga operativa

Dado que el equipo de desarrollo debe ser dueño del producto digital, y el producto digital es, en efecto, el sistema sobre el cual ocurre la operación del negocio, el equipo de desarrollo es entonces dueño de los procesos de negocio y responsable por asegurar la correcta ejecución de sus procesos. Eso significa que, aunque existan características que el producto digital todavía no tiene implementadas, el

equipo de desarrollo sigue siendo el responsable de que aquellos procesos que todavía no pueden descansar en la automatización ocurran de forma correcta.

Por ejemplo, en la sección 3.1.5.1 se describió que el producto digital se haría cargo de realizar las simulaciones, pero también se explicó que las simulaciones terminan apareciendo en Pipedrive para que los ejecutivos puedan continuar atendiendo el caso correspondiente. Eso significa que, para asegurar la continuidad del negocio, Pipedrive todavía no puede desaparecer como dependencia en el flujo de otorgamiento del crédito hipotecario. A pesar de que el equipo de desarrollo todavía no haga un entregable que permita a los operadores prescindir de Pipedrive, dentro de las responsabilidades del equipo de desarrollo sí se encuentra asegurar que la plataforma externa funcione correctamente o que al menos esté bien integrada con el producto digital, de tal manera que las simulaciones que ocurren a través del producto digital tengan como salida un caso al que se le puede hacer seguimiento en Pipedrive.

Se le llama carga operativa entonces, a todas aquellas actividades que debe asumir el equipo de desarrollo que no están relacionadas al desarrollo del producto digital, a resolución de incidencias o corrección de errores pero que sí están vinculadas a mantener la operación del negocio. Una situación común en que existe carga operativa se da en el escenario en que el producto digital todavía no cuenta con características de reportería para otras áreas de la organización, por lo tanto, el equipo de desarrollo debe asumir la responsabilidad de entregar oportunamente la información con los números que miden la recaudación de dividendos al departamento de finanzas, la cantidad clientes morosos al departamento de cobranza, entre otros. Resolver una tarea de carga operativa puede significar una operación manual como extraer series de datos del sistema y formatearlos para alguna otra área de la empresa, garantizar accesos para que los operadores puedan leerlos directamente, o habilitar alguna herramienta externa que extraiga esos datos y grafique paneles que puedan usar los operadores.

3.2.4 Integración continua

Uno de los acuerdos de trabajo adoptados por el equipo de desarrollo hace referencia a resguardar la calidad del software adoptando el paradigma de despliegue continuo (ver sección 2.2.3) para prevenir cambios en el código que no cumplan con los estándares de calidad que también fueron acordados por el mismo equipo.

Un ejemplo de un archivo que contiene la configuración del *pipeline* de despliegue continuo de la aplicación web (FrontEnd) se puede ver en el anexo E. En el caso particular de este trabajo, el *pipeline* de despliegue continuo se ejecuta usando la herramienta Drone.io (ver sección 2.2.3). Además, en el anexo F se puede ver un ejemplo de ejecución de un pipeline de despliegue continuo, usando Drone.io.

A continuación, se hará una descripción de las configuraciones que se adoptaron haciendo uso de herramientas de trabajo que sirven para implementar un sistema de entrega continua (ver sección 2.2.3).

1. **Análisis estático de código**
   Proceso de inspección del código fuente sin ejecutar el software. El objetivo es detectar posibles errores comunes, anti-patrones o vulnerabilidades. En el caso particular de este trabajo, en que el software está escrito usando Javascript y Typescript, el análisis estático de código se realiza usando el entorno de ejecución de Python3 y su herramienta *nodejsscan*.
2. **Instalación de dependencias**
   Es necesario que durante la ejecución del pipeline de despliegue continuo se instalen las dependencias del software para poder ejecutar las etapas siguientes del mismo pipeline. En el caso particular de este trabajo, en que el software está desarrollado usando los marcos de trabajo que proponen VueJS y NestJS (ver secciones 2.2.6 y 2.2.7), la instalación de dependencias se ejecuta con la instrucción *npm install*.
3. **Revisión de estilo de código**
   El estilo de código hace referencia al conjunto de acuerdos usados por el equipo de desarrollo para escribir código fuente, que abarca aspectos como tamaño de la tabulación, uso de saltos de línea, consistencia en definición de funciones y declaración de variables, entre otros. En el caso particular de este trabajo, el estilo de escritura de código está resguardado por una herramienta que se llama ESLint cuyas definiciones se encuentran en el anexo C.
4. **Revisión de vulnerabilidades**
   El software que se desarrolló en este trabajo está enmarcado en NodeJS (ver sección 2.2.5), que usa un sistema de distribución de bibliotecas o paquetes de software que sirven como dependencias. En el entorno de ejecución de NodeJS, se utiliza la instrucción *npm audit* que realiza una inspección simple por la lista de dependencias del proyecto correspondiente y levanta alertas en caso de detectar alguna dependencia que la comunidad de usuarios haya denunciado en el sistema de distribución de paquetes de NodeJS.
5. **Ejecución de pruebas unitarias**
   En este paso del pipeline de despliegue continuo se ejecutan las pruebas automatizadas que tienen como objetivo verificar que distintos segmentos específicos del código fuente funcionen correctamente. Al ejecutar las pruebas unitarias, se define un umbral de tasa de cobertura de código fuente, esto es, un porcentaje de líneas de código mínimo que sea necesario tener cubiertas para poder integrar un cambio en el software. En este trabajo se usó una herramienta llamada Jest para ejecutar pruebas unitarias.
6. **Preparación de pruebas funcionales**
   Dado que una pieza de software (por ejemplo, un servicio web) tiene dependencias en un contexto de ejecución como pueden ser bases de datos u otras piezas de software, es necesario preparar un ambiente ficticio que entregue respuestas que imiten una situación real. Esta etapa del pipeline pretende preparar el entorno ficticio para que, al momento de ejecutar las pruebas funcionales, la pieza de software que está siendo probada reciba las respuestas adecuadas.

7. **Ejecución de pruebas funcionales**
   A diferencia de las pruebas unitarias, las pruebas funcionales pretenden levantar y ejecutar piezas independientes de software y hacerlas funcionar para verificar su correcto comportamiento antes de ser integradas al resto del producto digital.
8. **Preparar pruebas end-to-end**
   Similar al paso de preparación de pruebas funcionales, el caso de las pruebas end-to-end también necesita un paso de preparación de entorno temporal no productivo que levante todas las piezas de software involucradas en el producto digital, como lo pueden ser distintos servicios web y bases de datos involucradas.
9. **Ejecución de pruebas end-to-end**
   Las pruebas end-to-end tienen como objetivo hacer funcionar las características del producto digital involucrando todos sus componentes, desde la interfaz de usuario, los servicios involucrados y hasta la base de datos del sistema. En el caso particular de este trabajo, las pruebas end-to-end se ejecutan usando una herramienta que se llama Cypress.
10. **Transpilar código**
    Hace referencia a “traducir” código fuente que está escrito en un lenguaje de programación, como puede ser TypeScript a otro lenguaje como puede ser JavaScript para contar con archivos que puedan interpretados en un entorno de ejecución. El resultado es una versión del software utilizable en un entorno de ejecución.
11. **Borrar dependencias no productivas**
    Luego de haber generado una versión ejecutable del software, se eliminan las dependencias que no son necesarias en el entorno productivo, esto es, todos aquellos paquetes que fueron utilizados en las etapas anteriores del pipeline de despliegue continuo pero que no son requeridos para el correcto funcionamiento del software.
12. **Crear imagen Docker**
    El sentido de tener disponible una imagen Docker de la pieza de software es contar con un versionado del programa y también proveer un medio para poder levantarlo, ejecutarlo y probarlo independiente del sistema en que se esté trabajando. En el caso particular de este trabajo, las imágenes Docker de las piezas de software son usadas en el contexto de ejecución de pruebas funcionales o de pruebas end-to-end, pero no para el entorno productivo.
13. **Desplegar**
    Es el último paso del pipeline de despliegue continuo y consiste en dejar la versión del software generado funcionando en un ambiente productivo. En el caso particular de este trabajo, el despliegue de servicios web se ejecuta con la instrucción *eb deploy* y en el caso de la aplicación web con la instrucción *aws s3 sync*, ambos desde el entorno de ejecución de Python3.
14. **Notificar despliegue exitoso**
    Dado que se está implementando el paradigma de despliegue continuo (ver sección 2.2.3), es una buena práctica notificar por algún medio de comunicación (por ejemplo, un canal de chat o por e-mail) a las partes interesadas par que tengan la oportunidad de revisar la nueva versión productiva del software desplegado.

4 PRODUCTO DIGITAL

En este capítulo se analizarán las principales características del producto digital desarrollado dentro del marco del alcance académico de este trabajo (ver sección 1.4). Primero se abarcan los principales lineamientos usados como referencia al momento de diseñar el producto digital tanto desde la perspectiva técnica como del punto de vista de la experiencia de usuario, teniendo en cuenta la filosofía organizacional propuesta en este trabajo y el propósito del producto digital. Luego se abarcan las definiciones técnicas utilizadas para implementar el modelo de datos y la arquitectura del software. Finalmente, se presentan las características implementadas en el producto digital durante el desarrollo de este trabajo.

4.1 Paradigma: el sistema es el producto, el producto es el sistema

Teniendo presente el problema descrito en la sección 1.1.3 se propone una estrategia usando los nuevos lineamientos de la organización:

- El producto digital no es un sistema de planificación de recursos empresariales (ERP) o una herramienta de apoyo a la operación. El producto digital es el sistema sobre el cual opera el negocio, por lo tanto, cualquier cambio en el producto digital es un cambio en el sistema de la organización.
- Cualquier paso del proceso de otorgamiento del crédito hipotecario y venta del mutuo de inversión que pueda ser automatizado, será automatizado y ejecutado por el sistema, o sea, por el producto digital.
- El proceso de otorgamiento del crédito hipotecario y venta del mutuo es consecuencia del producto digital, por lo tanto, cualquier cambio en el diseño del producto digital significa también un cambio en el proceso del negocio.
- El diseño del producto digital está basado en el diseño del proceso, por lo tanto, cualquier cambio en el proceso es también un cambio en el producto digital.
- El producto no es un ERP sino una solución concebida de cara a usuario, con el cual interactúan tanto los clientes deudores, los inversionistas que adquieren mutuos, los ejecutivos comerciales y las distintas áreas de la compañía.

En la sección 3.1.5 se presentaron las oportunidades inmediatas de mejora propuestas para ser implementadas durante las primeras iteraciones y estar presentes en las primeras versiones del producto digital. A continuación, se exponen los detalles técnicos relacionados a la implementación de esas características: el modelo de datos utilizado, la arquitectura del sistema y los servicios de computación en la nube usados para alojar el producto digital de la organización.

4.2 Modelo de datos

En esta sección se presenta el modelo de datos que permite el funcionamiento del simulador y del sitio privado. La base de datos del sistema está alojada en *MongoDB Atlas* (ver sección 2.2.4.7) y está modelado en torno a colecciones que representan cada identidad que interactúa en el sistema. De esa manera entonces, cada documento en una colección representa una instancia de la correspondiente entidad. En la Tabla 4-1 se detallan las entidades concebidas para manejar las primeras características que están soportadas por el producto digital. El anexo G muestra un diagrama del modelo de datos de cómo las distintas entidades se relacionan entre ellas.

Colección	Caracterización	
users	*_id*:	Identificador único del documento en la base de datos.
	name:	Nombre del usuario.
	lastName:	Apellido del usuario.
	password:	Digerido de la contraseña del usuario.
	email:	Correo electrónico del usuario.
	lastLogin:	Fecha del último inicio de sesión.
	role:	Valor que describe qué tipo de usuario es asignado, puede ser cliente, inversionista, ejecutivo o administrador.
	clientId:	Identificador único del documento correspondiente a la colección *clients*.
	investorId:	Identificador único del documento correspondiente a la colección *investors*.
	avatar:	Url de la imagen asociada al usuario.
clients	*_id*:	Identificador único del documento en la base de datos.
	userId:	Identificador único del documento correspondiente a la colección *users*.
	name:	Nombre del cliente.
	lastName:	Apellido del cliente.
	idNumber:	Número de identidad del cliente.
	phone:	Número de teléfono del cliente.
	email:	Correo electrónico del cliente.
	walletId:	Identificador único del documento en la colección *wallets*.
operations	*_id*:	Identificador único del documento en la base de datos.
	clientId:	Identificador único del documento correspondiente a la colección *clients*.
	investorId:	Identificador único del documento correspondiente a la colección *investors*.
	stateId:	Número único asignado por la CMF para identificar el crédito hipotecario.
	term:	Plazo del crédito hipotecario en cantidad de meses.
	Insured:	Monto asegurado de la propiedad.
	approved:	Monto aprobado para el crédito hipotecario.
	fireRate:	Tasa del seguro de incendio y sismo de la propiedad.
	lifeRate:	Tasa del seguro de desgravamen del crédito hipotecario.
	signature:	Fecha de firma de la escritura.
	payments:	Arreglo de objetos que representan las cuotas del crédito hipotecario.

Colección	Caracterización	
investors	*_id*:	Identificador único del documento en la base de datos.
	name:	Nombre del cliente institucional.
	idNumber:	Número de identidad de la institución que invierte comprando mutuos hipotecarios.
	account:	Objeto JSON que contiene los datos de la cuenta bancaria en que se le depositan las remesas al cliente institucional.
	contact:	Objeto JSON que contiene los datos de contacto del cliente institucional.
transactions	*_id*:	Identificador único del documento en la base de datos.
	clientId:	Identificador único del documento correspondiente a la colección *clients*.
	selected:	Arreglo de objetos JSON que contienen información sobre las cuotas seleccionadas para pagar. Pueden ser varias cuotas y de distintos créditos.
	amount:	Monto a pagar total de la transacción.
	status:	Valor que indica el estado de la transacción, puede ser pendiente, confirmada, cancelada, fallida o remesada.
	provider:	Valor que identifica el método de pago utilizado para pagar la transacción.
prospects	*_id*:	Identificador único del documento en la base de datos.
	name:	Nombre del prospecto.
	lastName:	Apellido del prospecto.
	email:	Correo electrónico del prospecto.
	idNumber:	Número de identidad del prospecto.
	simulations:	Arrego de identificadores únicos de documentos correspondientes a la colección *simulations*.
consignments	*_id*:	Identificador único del documento en la base de datos.
	investorId:	Identificador único del documento correspondiente a la colección *investors*.
	payments:	Arreglo de objetos JSON que representan las cuotas asociadas a esta remesa.
	amount:	Monto total de la remesa hecha al cliente institucional.
wallets	*_id*:	Identificador único del documento en la base de datos.
	clientId:	Identificador único del documento correspondiente a la colección *clients*.
	amount:	Monto en la billetera del cliente.
simulations	*_id*:	Identificador único del documento en la base de datos.
	prospectId:	Identificador único del documento correspondiente a la colección *prospects*.
	offer:	Objeto JSON que contiene los detalles de la oferta hecha al prospecto.
	params:	Objeto JSON que contiene los valores de los parámetros usados para realizar la operación matemática de la simulación.

Tabla 4-1: Modelo de datos del producto digital.

4.3 Arquitectura del sistema

La interfaz de usuario del producto digital está desarrollada en VueJS (ver sección 2.2.6) y alojada en AWS S3 (ver sección 2.2.4.4), mientras que los servicios web están desarrollados en NestJS (ver sección 2.2.7) y desplegados en AWS Elastic Beanstalk (ver sección 2.2.4.2). Además, los servicios web que no deben estar expuestos a internet, se encuentran protegidos por la herramienta AWS API Gateway (ver sección 2.2.4.6). En la Figura 4-1 se muestra un esquema que grafica a alto nivel, la arquitectura del producto digital contemplando las características implementadas en el alcance de este trabajo. En azul se muestran los elementos de FrontEnd y en anaranjado se muestran los servicios web. En amarillo está representada la base de datos que contiene las distintas colecciones (en gris) de las entidades presentadas en el apartado anterior. Las flechas indican el sentido de la dependencia o de las peticiones que interactúan dentro del producto digital. Por ejemplo, FrontEnd del sitio privado puede realizar peticiones HTTP contra el API Gateway, que a su vez transmite la petición contra el microservicio *clients,* que va contra el microservicio *wallets* y administra la entidad *clients* de la base de datos.

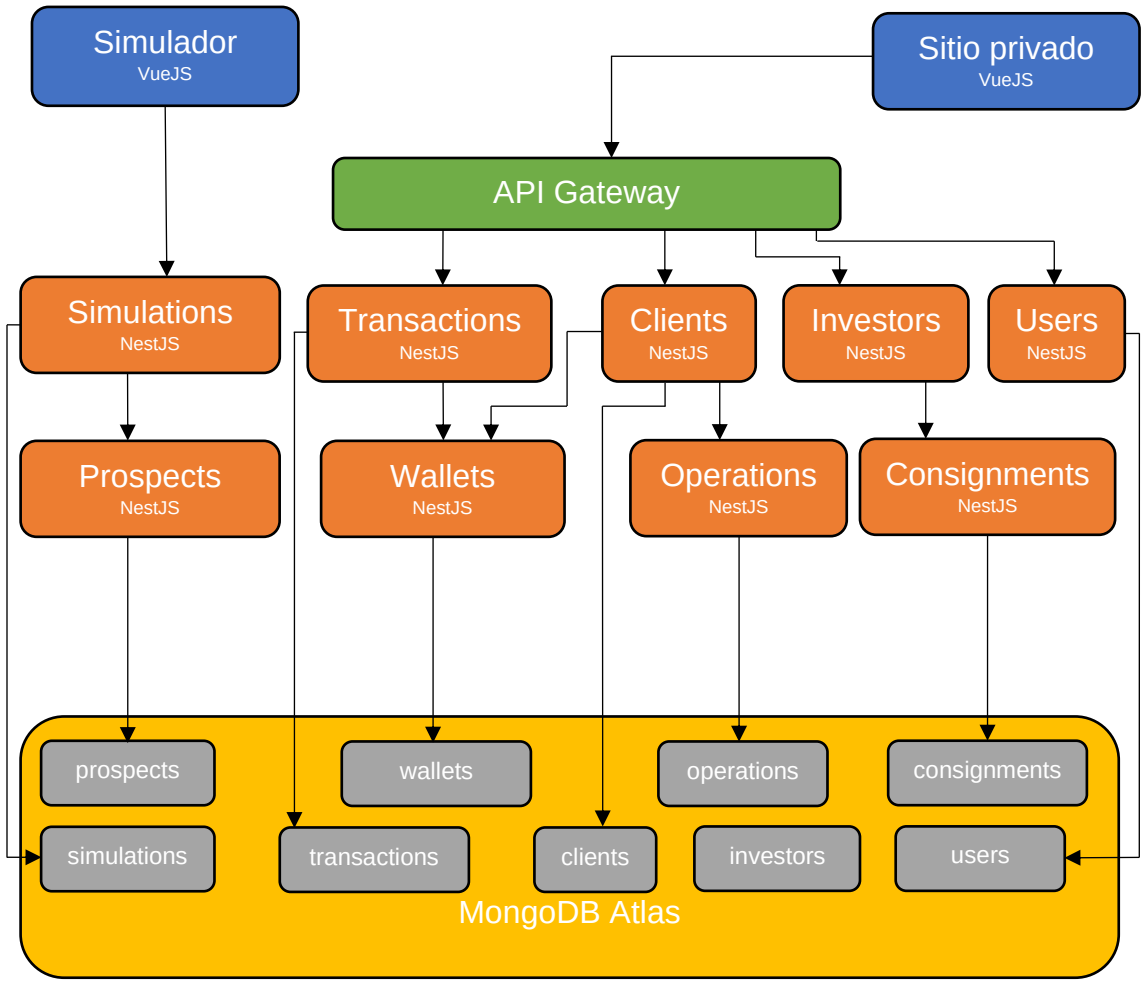


Figura 4-1: Arquitectura del producto digital.

4.4 Simulador

Dado que la simulación se basa en un cálculo matemático que es posible de implementar directamente en el software, esta característica representa una de las mejoras más importantes con respecto al estado anterior a la intervención realizada en la empresa. La Figura 4-2 muestra la interfaz de usuario del simulador usando datos de prueba que no son reales y utilizando el producto digital desplegado en un ambiente pre productivo cuyo propósito es poder usar el software sin contaminar los datos del sistema en ambiente de producción.

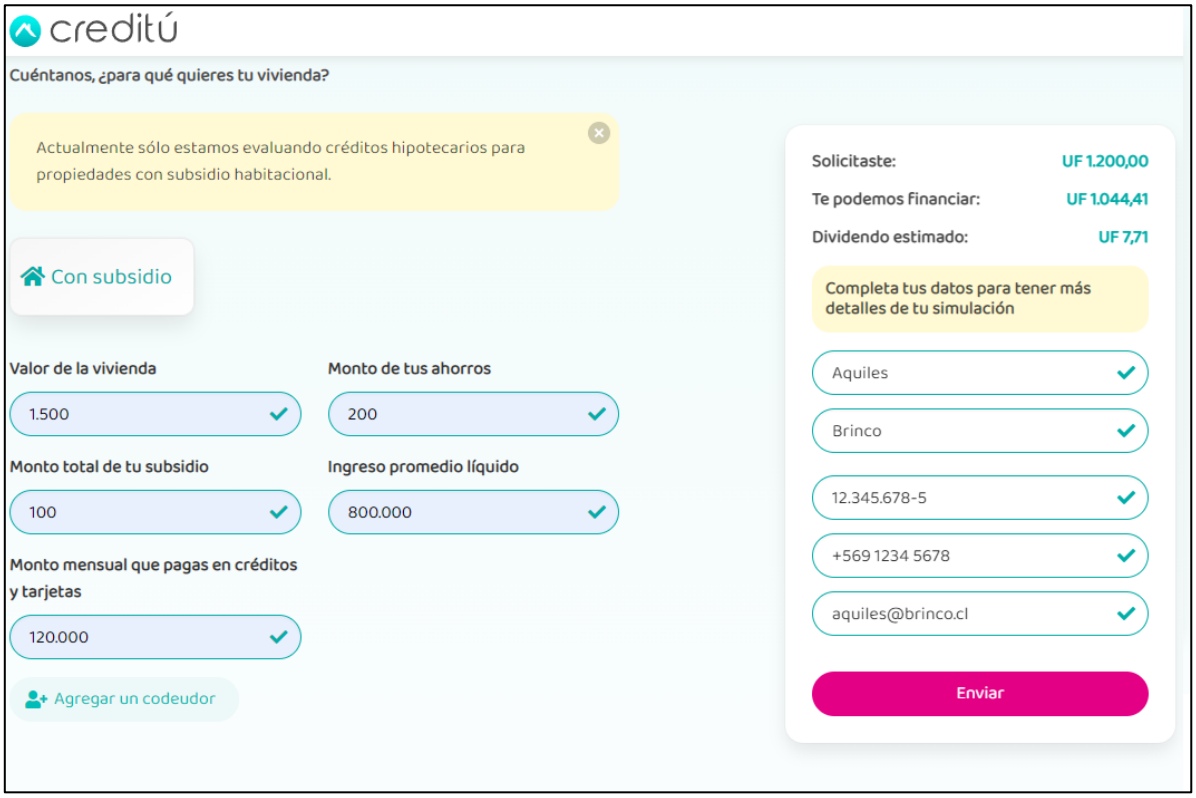


Figura 4-2:Simulador del producto digita.

En lugar de estar recogiendo los datos del *lead* para transmitirlos a distintas plataformas (ver sección 3.1.5.1) y posteriormente obtener la oferta, esta vez el simulador tiene la capacidad de ir reaccionando a la interacción del usuario y mostrar instantáneamente la tentativa de oferta que se le podría ofrecer al prospecto en caso de ser evaluado positivamente. La idea es proveer una experiencia estilo “calculadora” de tal manera que el usuario pueda inmediatamente tomar decisiones con respecto al pie que puede aportar o respecto a declarar un codeudor en la simulación. Otra mejora del nuevo simulador es que el *lead* puede tomar la decisión de enviar sus datos de contacto para continuar con la simulación en caso de que vea una oferta que esté dentro de su expectativa. Si bien este cambio debería disminuir la cantidad de simulaciones, también es de esperar que aumente la tasa de conversión de *leads* que se convierten en prospectos (ver sección 3.1.5.1).

El usuario sólo puede continuar enviando sus datos en caso de que exista una oferta a partir del cálculo matemático realizado. Cuando el usuario envía sus datos, el sistema guarda los datos del *lead* que en el futuro puede convertirse en un cliente que use el resto de las características del producto digital. Además, se deja creado un caso en Pipedrive para que los ejecutivos puedan continuar con el proceso. La Figura 4-3 muestra el resultado de una simulación, incluyendo el valor de la carga anual equivalente (CAE) que no puede ser mostrada de forma instantánea dado que involucra un cálculo con varias iteraciones, por lo que toma unos segundos.

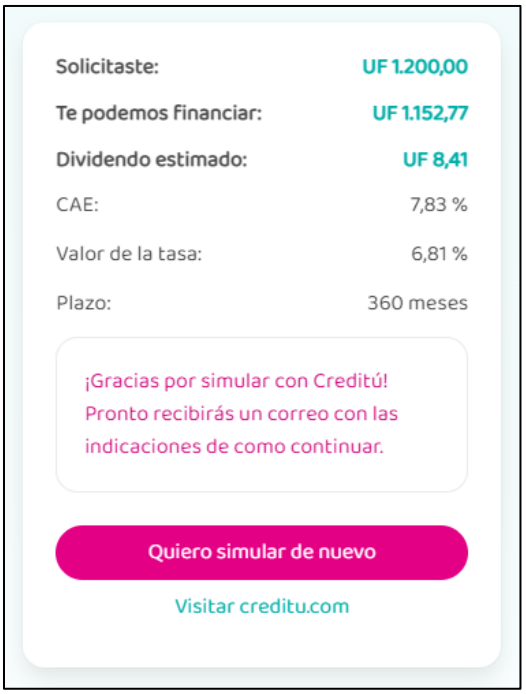


Figura 4-3:Resultado de la simulación.

A pesar de que ya se está prescindiendo de WordPress, Jorform y Zappier, para asegurar la continuidad del negocio todavía no se puede eliminar a Pipedrive del flujo de otorgamiento del crédito. Cuando el producto digital cuente con las características necesarias para hacerse cargo de las siguientes etapas del *customer journey,* se podrá prescindir de todas las plataformas ajenas al producto digital. Esto se escapa del alcance de este trabajo y debiese estar presente en futuros incrementos.

4.5 Sitio privado

Otra característica identificada como oportunidad de mejora inmediata durante el levantamiento inicial de la intervención realizada es entregar a los clientes la posibilidad de pagar sus dividendos (ver sección 3.1.5.2) a través de la página web de la organización, dándoles acceso a un sitio privado. La Figura 4-4 muestra la interfaz implementada para ingresar al sitio privado, solicitando correo electrónico y contraseña.

En futuras iteraciones, el sitio privado debiese hacerse cargo de otras características básicas como solicitud de certificados, centro de ayuda, historial de pagos, etc.

Figura 4-4: Interfaz para ingresar al sitio privado.

La implementación del sitio privado representó un esfuerzo importante que incluyó la recolección de la nómina completa de clientes activos, además de realizar campañas para informar y sensibilizar a los clientes sobre el cambio en el método de pago.

Antes de la intervención, la recaudación estaba externalizada a través de un proveedor que ofrecía el servicio de pago presencial por caja. Eso significa que los clientes no estaban acostumbrados a pagar sus dividendos por Internet y por lo tanto hubo que ejecutar un esfuerzo adicional para informarlos, asistirlos y educarlos. Este es un buen ejemplo de trabajo en conjunto con el área de marketing, que representa carga operativa (ver sección 3.2.3) para el equipo de desarrollo en que el principio de centrarse en el cliente que propone LeSS (ver Tabla 2-6) es importante de aplicar para gestionar el cambio tanto del lado de la organización como desde la perspectiva de la interacción con los clientes.

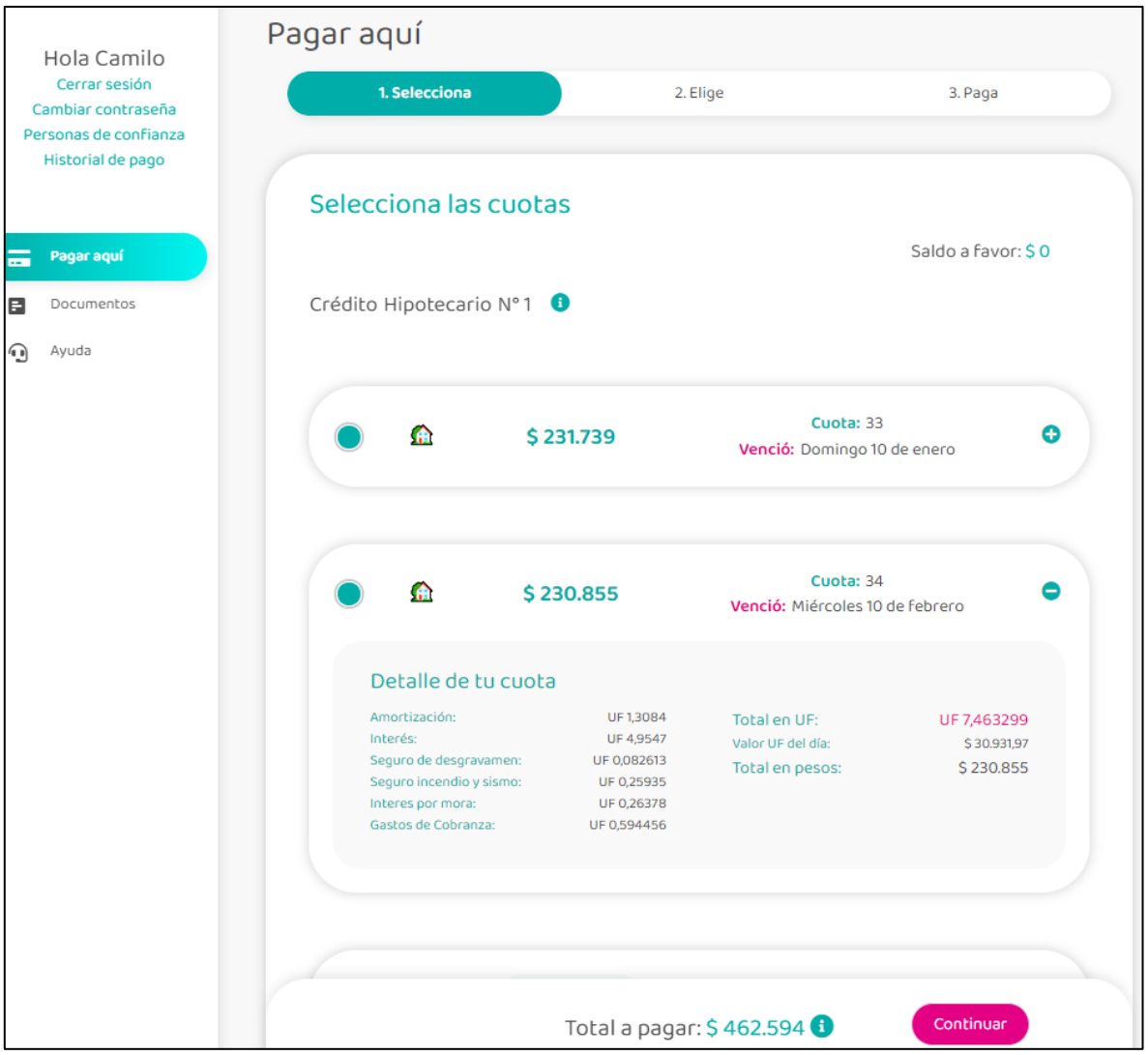


Figura 4-5: Sitio privado.

El sitio privado le permite a un usuario elegir qué cuotas desea pagar de sus créditos vigentes, considerando interés moratorio y gastos de cobranza según la situación particular o la fecha de vencimiento de la cuota respectiva. Dado que los dividendos están calculados en UF, el monto que ve el cliente es calculado al momento de ingresar a la interfaz de usuario. Además, por razones relacionadas a la seguridad de la información y la prevención de fraude, el cálculo de la deuda total ocurre desde el lado del servidor (BackEnd), de tal manera que la información es proveída a la aplicación web (FrontEnd) tras realizar una consulta después de haber ingresado al sitio privado.

5 RESULTADOS

En este capítulo se presentarán las mediciones realizadas después de haber realizado la intervención en la empresa desde el punto de vista del problema expuesto en la sección 1.1.2 y el objetivo propuesto en el párrafo 1.2. Primero se comparan las diferencias en la tasa de conversión del simulador antes y después de la intervención, luego se revisan los números referentes a la recaudación de los dividendos y después se presta una encuesta de percepción realizada en el equipo de desarrollo para realizar un breve análisis con respecto a la adopción de LeSS.

5.1 Conversión del simulador

La implementación del nuevo simulador (ver sección 4.4) preveía disminuir la cantidad de simulaciones debido a que el usuario no envía sus datos para realizar una simulación sin tener una tentativa de oferta calculada previamente desde el lado del FrontEnd. Si el usuario no observa una tentativa de oferta que esté dentro de su expectativa, entonces tiene la opción de no continuar o de tratar de mejorar los parámetros que está ingresando para buscar un resultado mejor. Es de esperarse entonces que la cantidad de simulaciones disminuya y de esa forma, se puede disminuir la carga de trabajo realizada por los operadores al recibir solicitudes, ya que comienzan a administrar solamente casos que sí tienen una oportunidad de recibir una oferta desde el punto de vista del cálculo matemático.

También existe la expectativa que la tasa de conversión de *leads* a prospectos mejore debido a que la primera barrera, que depende el monto de los ahorros y los ingresos, ya viene superada.

Todos estos supuestos funcionan como hipótesis que fueron probadas al implementar las características correspondientes del producto digital, obedeciendo al principio de control empírico del proceso (ver Tabla 2-6) que propone LeSS. La Tabla 5-1 muestra los números de cantidad de simulaciones y tasas de conversión antes y después de la intervención durante un periodo de dos meses separados por un año.

Simulaciones antes	Conversión antes	Simulaciones después	Conversión después
1563	16%	643	63%

Tabla 5-1: Comparación de simulaciones antes y después de la intervención.

El comportamiento observado está dentro de lo esperado, esto es, la cantidad de simulaciones disminuyó mientras la tasa de conversión aumentó. En un período de dos meses, antes de contar con el simulador del producto digital se lograba generar del orden de 250 prospectos a partir de 1563 *leads*, mientras que usando el producto digital y eliminando el canal inmobiliario, se lograron generar del orden de 405 prospectos a partir de 643 *leads.* Estos números se encuentran graficados en la Figura 5-1.

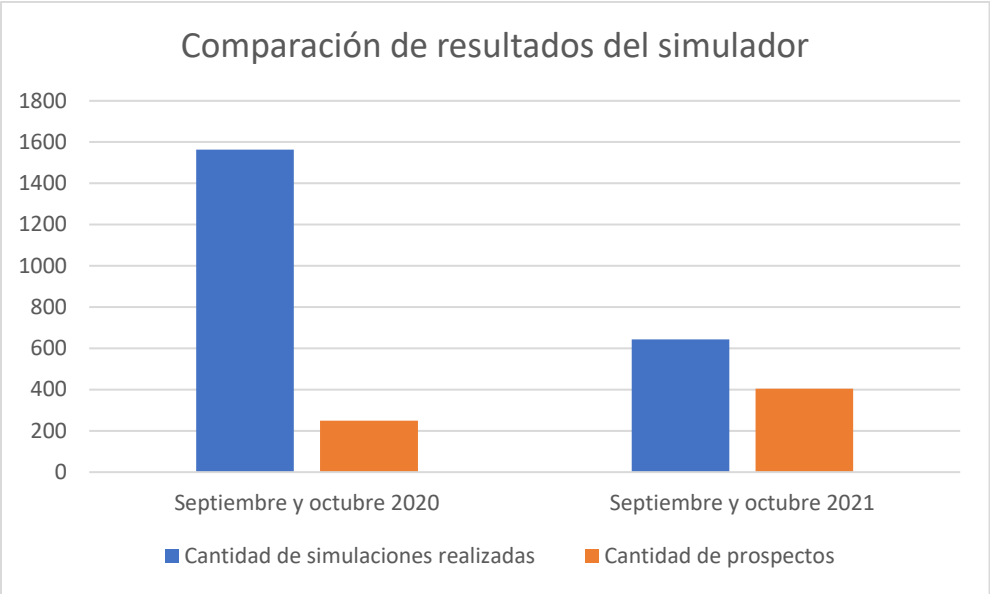


Figura 5-1: Grafico de comparación de resultados del simulador.

5.2 Metas de recaudación y mora

De la misma forma que el simulador del producto digital tuvo un impacto en la cantidad de prospectos generados, se esperaba también que el sitio privado (ver sección 4.5) también tuviese un impacto en los números referentes a la recaudación de dividendos. La expectativa en este caso era que la recaudación disminuyera y la mora aumentara, debido a que los clientes estaban acostumbrados a pagar presencialmente en vez de usar herramientas tecnológicas o estar pagando por Internet.

A diferencia del simulador, la migración de los clientes con créditos hipotecarios activos se realizó gradualmente durante un período de 5 meses, de tal manera que en cada período se incluyó una cantidad de clientes más grande que el período anterior. El primer mes se migraron sólo 10 clientes, que fueron elegido según su perfil de buenos pagadores y su edad para asegurar adherencia a herramientas tecnológicas y no hubo mora. El segundo mes se migró un grupo de 50 clientes más, favoreciendo los mismos criterios y sólo se presentó un 5% de clientes morosos. El tercer mes se hizo la migración de 100 clientes más, haciendo subir el porcentaje de mora a un 42,5%. El cuarto mes se migraron 500 clientes más y gracias a la capacidad de respuesta que había generado hasta entonces el equipo de desarrollo, involucrando asistencia a los clientes, campañas de sensibilización y notificaciones tempranas de vencimiento de cuotas, la mora pudo ser reducida a un 13,3%. El quinto mes se migró toda la cartera de clientes faltantes, logrando que la organización tenga la totalidad de su cartera de créditos hipotecarios activos siendo cobrados a través del producto digital, lo cual resultó en un porcentaje de mora del 28%. La Tabla 5-2 contiene los números de la cantidad de clientes migrados en cada período junto con el detalle del avance de tasa de mora de toda la cartera migrada.

	Abril 2021	Mayo 2021	Junio 2021	Julio 2021	Agosto 2021
Tamaño de la migración	10	50	100	500	720
Total de clientes migrados	10	60	160	660	1380
Dividendos pagados	10	57	92	572	968
Porcentaje de mora	0	5	42,5	13,3	29,8

Tabla 5-2: Migración de clientes y recaudación de dividendos.

Esta información es interesante porque permite estudiar tanto la proporción de clientes migrados en el tiempo contra la cartera completa y cómo se fue completando en el tiempo durante la migración. La Figura 5-2 muestra un gráfico de barras contrastando los clientes ya migrados contra los que fueron agregados en cada período.

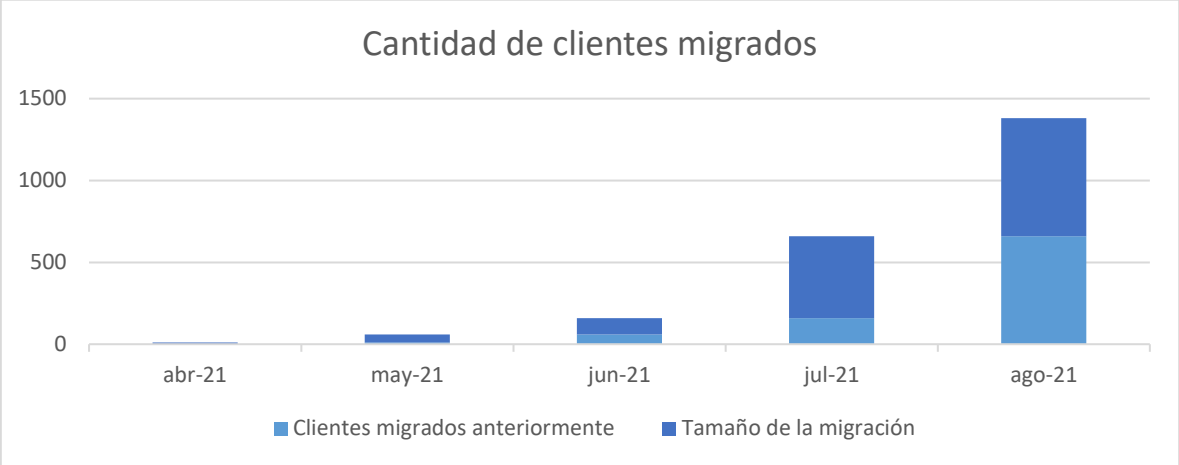


Figura 5-2: Gráfico de avance de la migración de clientes.

Otra observación interesante que se puede hacer es el comportamiento que va teniendo la diferencia entre cantidad de dividendo pagados y cantidad de clientes migrados en cada período. La Figura 5-3 muestra este comportamiento con un gráfico de líneas con marcadores en el tiempo.

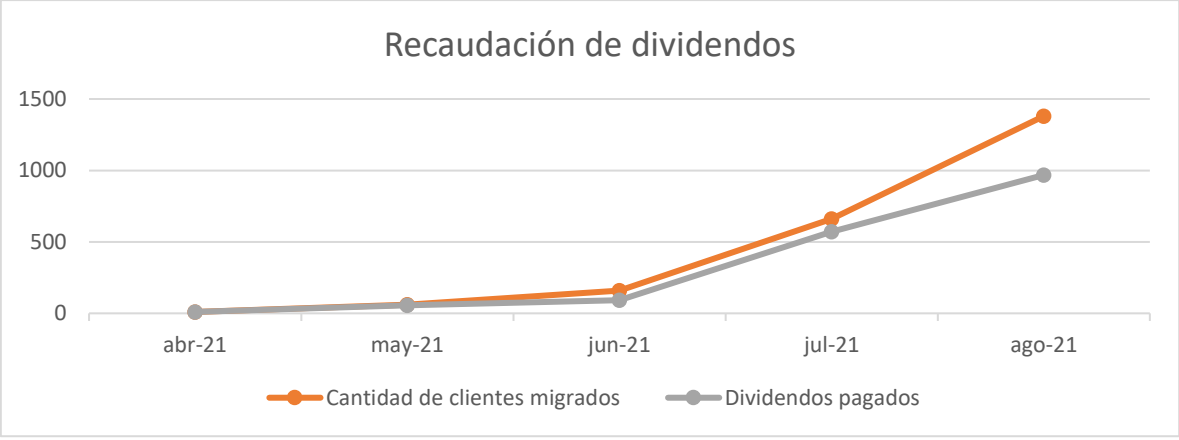


Figura 5-3: Cantidad de dividendos pagados durante la migración.

Un gráfico similar al anterior se puede ver en la Figura 5-4, pero con respecto al porcentaje de mora a través del tiempo.

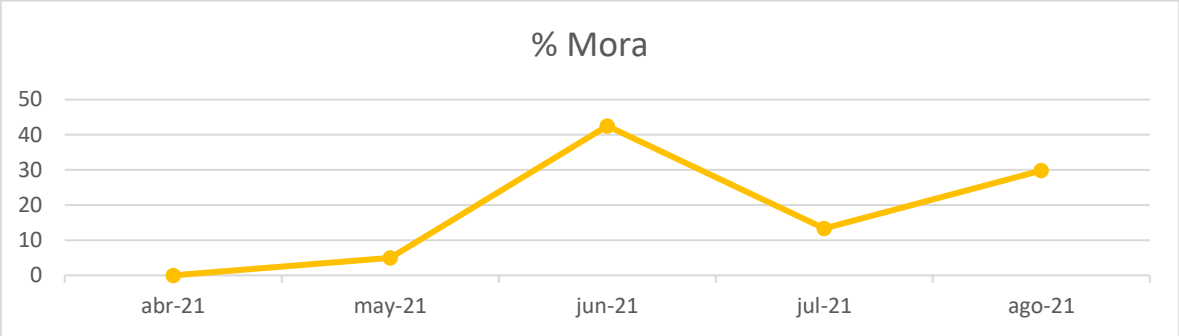


Figura 5-4: Evolución del porcentaje de mora durante la migración.

Finalmente, la Tabla 5-3 muestra una comparación entre un período de un mes antes de la intervención y un período de un mes luego de 90 días de haber implementado el sitio privado y de haber terminado de migrar toda la cartera de clientes para que paguen sus dividendos en el sitio privado. Se debe tener en cuenta que estos números son obtenidos a partir de un universo de 1380 operaciones activas que son administradas por la organización.

Dividendos pagados antes	Mora antes	Dividendos pagados después	Mora después
1163	13%	968	28%

Tabla 5-3: Comparación de recaudación y mora.

Los resultados obtenidos en esta materia también confirmaron la hipótesis planteada. Si bien los nuevos valores son peores que los antiguos, el equipo de trabajo continúa con la tarea de gestionar junto a las demás áreas de la organización la mejora en la recaudación y la disminución de la mora. Una mejora en la mora de la cartera de clientes fue registrada desde el tercer al cuarto período durante la migración, cuya explicación puede encontrarse en la rápida capacidad de respuesta que ha adquirido el equipo de desarrollo al adoptar las nuevas filosofías de trabajo. Estos son números relativamente complejos de analizar, porque un dividendo pagado en un período específico no representa necesariamente una disminución de la mora, ya sea porque se está pagando a tiempo, o bien porque corresponde a un período anterior y el muto sigue en mora después de cobrar ese dividendo específico.

5.3 Despliegue continuo

El segundo objetivo específico planteado en la sección 1.2 hace referencia a implementar un protocolo que impida cambios en el software que no cumplan con los estándares. En la sección 3.2.4 se presentaron los pasos que se ejecutan en el pipeline de despliegue continuo que previenen integraciones que no tengan las características para ejecutar cada paso de forma exitosa. A continuación, se

presentan los acuerdos de trabajo adoptados por el equipo de desarrollo con respecto a integrar cambios en el software.

- Cualquier cambio en el código fuente deber pasar de forma exitosa el pipeline de integración continua.
- Cualquier cambio en el código debe pasar por una revisión de pares y ser aprobado por al menos un integrante del equipo de desarrollo (que no sea quien hizo el cambio) antes de lanzar la ejecución del pipeline de integración continua.
- Sesiones de *pair programming* o *mob programming* sí cuentan como revisión de pares.
- Las pruebas unitarias deben cubrir al menos un 80% del código fuente.
- Cada criterio de aceptación propuesto en cada ítem del *backlog* debe tener una correspondiente prueba funcional o prueba end-to-end.
- Las pruebas deben ser escritas antes del código fuente que las satisface.

Siguiendo los principios de LeSS, todos estos acuerdos de trabajo fueron propuestos por el equipo de desarrollo en vez de haber sido impuestos la gerencia o una figura de liderazgo técnico. A través del uso de las herramientas que ofrecen las plataformas de alojamiento y versionado de código (como GitLab o GitHub) es posible configurar los repositorios del código fuente del software para asegurar que estos acuerdos sean cumplidos. Por ejemplo, es posible configurar que los cambios de una rama específica presentes en un *pull-request* no puedan ser mezclados a menos que otro integrante del equipo de desarrollo lo apruebe (revisión de pares).

De esta forma, y teniendo en cuenta las definiciones del sistema de integración continua implementado (ver sección 3.2.4), es que el segundo objetivo específico propuesto para este trabajo (ver sección 1.2) se considera cumplido.

5.4 Encuesta de percepción

El primer objetivo específico presentado en la sección 1.2 hace referencia a que la organización adopte un paradigma de trabajo horizontal y que el equipo de desarrollo adopte tanto nuevas libertades como responsabilidades (ver sección 3.2.2). Dado que el planteamiento de este objetivo es más bien de alto nivel y sería muy difícil de medir cuantitativamente parámetros que indiquen el “nivel de adopción” del marco de trabajo dentro de la organización.

En lugar de tratar de medir niveles de adherencia, se realizó una encuesta de percepción dentro del equipo de desarrollo, usando el método de evaluaciones sumarias [17] y basándose en 10 enunciados ligados a los principios de LeSS, de tal manera que cada uno de los enunciados presentes en la encuesta se puede relacionar directamente con uno de los principios de LeSS expuestos en la Tabla 2-6. La encuesta se realizó después de un año de haber comenzado a trabajar con LeSS, en un momento en que tanto el simulador nuevo como el sitio privado que permite a los clientes pagar sus dividendos ya estaban en condiciones productivas.

Como se mencionó en la sección 3.2.1, el equipo de desarrollo está conformado por 12 personas. A cada uno de los enunciados, cada integrante del equipo de desarrollo declaró una de las siguientes opciones: “Muy en desacuerdo”, “En desacuerdo”, “Ni en desacuerdo ni de acuerdo”, “De acuerdo” o “Muy de acuerdo”.

La Tabla 5-4 muestra los diez enunciados usados en la encuesta de percepción y el correspondiente principio de LeSS con el que está relacionado. Las relaciones entre los números asignados a los enunciados junto con el principio de LeSS asociado serán usados en esta sección para presentar los resultados de la encuesta de percepción.

N.º	Enunciado	Principio LeSS
1	En el equipo de desarrollo se usan los principios de Scrum aplicados a varias células.	LeSS es Scrum
2	El trabajo del equipo de desarrollo favorece empíricamente la situación de la organización.	Control empírico del proceso
3	El trabajo del equipo de desarrollo cumple con la *Definition of Done* acordada.	Transparencia
4	El flujo de trabajo del equipo de desarrollo no presenta burocracia o necesidad de contar con autorizaciones para realizar cambios. Las decisiones se toman al interior del equipo.	Más con menos
5	Mi trabajo no se concentra solamente en desarrollo de sofware, sino también en la visión de alto nivel del negocio y la totalidad del producto.	Enfoque en la totalidad del producto
6	Los ítems del *backlog* se conciben en torno a las necesidades de los clientes y se recibe *feedback* del usuario final para seguir proponiendo nuevos ítems.	Centrado en el cliente
7	Los incrementos entregados no presentan defectos y mejoran la experiencia de los usuarios del producto digital.	Mejora continua
8	Comprendo el sistema en profundidad y soy capaz de intervenir en distintos frentes relacionados al producto digital.	Reflexión sistémica
9	La gerencia de la organización favorece el empoderamiento del equipo de desarrollo y facilita espacios para que la mejora continua sea una política de toda la empresa.	Reflexión magra
10	Los ítems del *backlog* están bien ordenados con respecto a su valor para el negocio y esfuerzo requerido para implementar.	Teoría de colas

Tabla 5-4: Enunciados usados en la encuesta de percepción.

En el anexo H se encuentran los resultados de cada respuesta obtenida por los integrantes del equipo de desarrollo en la encuesta de percepción. En la Tabla 5-5 se muestra un resumen, de tal manera que las columnas representan los enunciados expuestos y las filas representan la respuesta elegida por los integrantes del equipo de desarrollo. Cada celda contiene entonces un número indicando la cantidad de respuestas en cada combinación.

Respuesta / Enunciado	1	2	3	4	5	6	7	8	9	10
Muy en desacuerdo	0	0	0	0	0	0	0	0	0	0
En desacuerdo	0	0	3	0	0	1	2	2	0	4
Ni en desacuerdo ni de acuerdo	1	5	6	1	0	6	7	5	1	7
De acuerdo	6	4	3	4	6	5	3	3	5	1
Muy de acuerdo	5	3	0	7	6	0	0	2	6	0

Tabla 5-5: Resultados de la encuesta de percepción.

Para hacer más fácil el análisis de esta información, se pueden generar gráficos de que permitan contrastar visualmente los resultados de cada uno de los enunciados. La Figura 5-5 muestra la misma información que la Tabla 5-5, pero presentando los histogramas de cada uno de los diez enunciados de la encuesta de percepción. Una observación interesante es que no existe ningún enunciado al que se le haya asignado la respuesta “muy en desacuerdo”, mientras que la máxima cantidad de respuestas con la opción “en desacuerdo” se encuentra en el enunciado asociado al principio de LeSS que hace referencia a la teoría de colas.

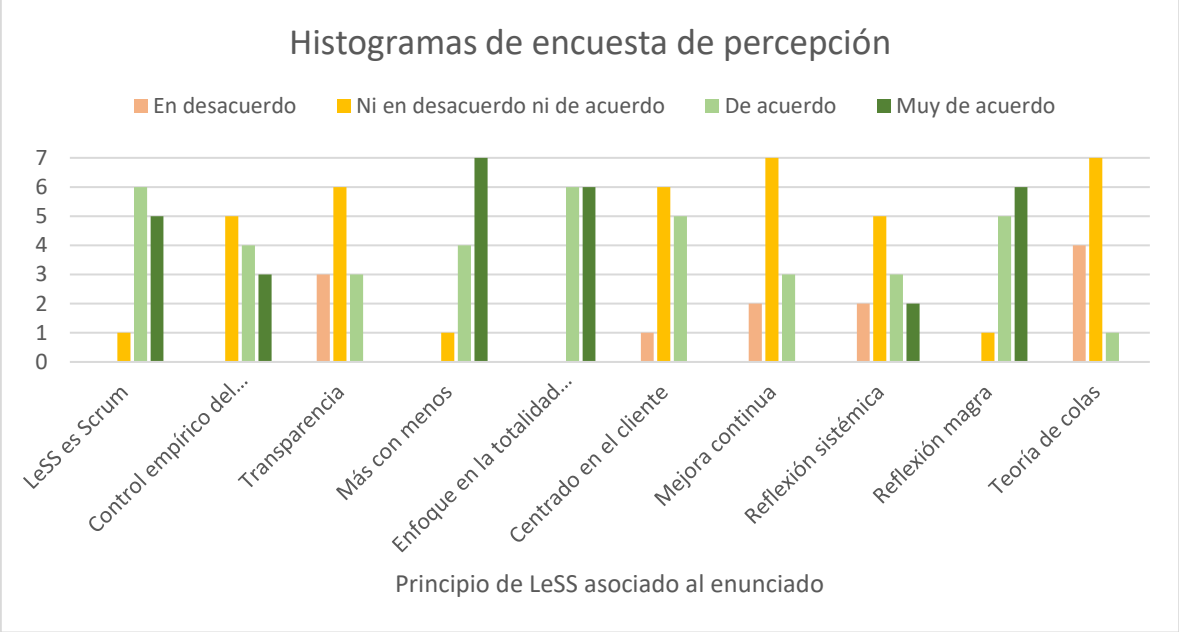


Figura 5-5: Histogramas de la encuesta de percepción.

Además, la Figura 5-6 también muestra la misma información, pero dispuesta en un gráfico de barras apiladas divergentes, lo cual permite observar que el enunciado asociado al principio de LeSS referente al enfoque en la totalidad del producto es el que se observa más desplazado hacia la derecha.

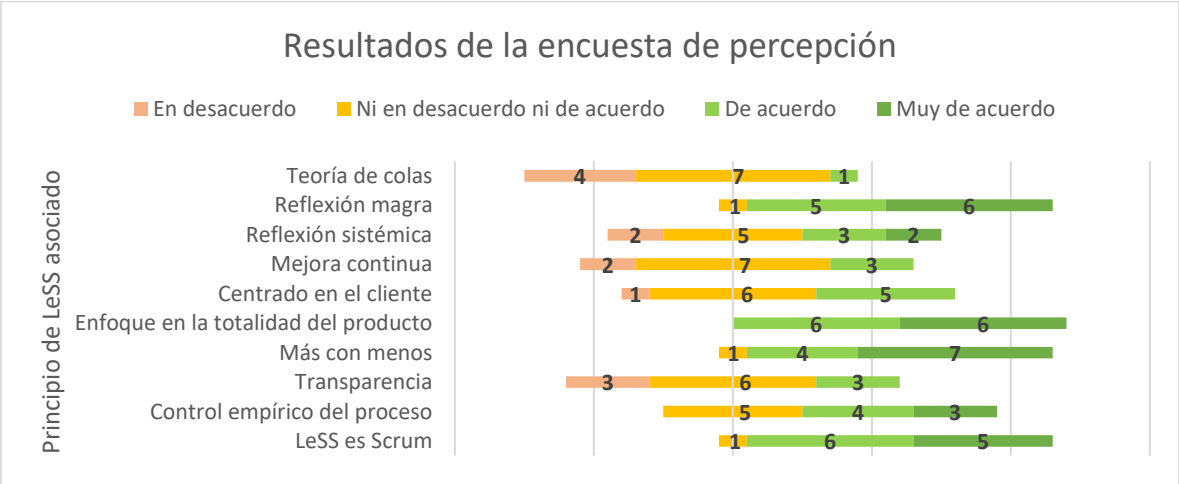


Figura 5-6: Resultados de la encuesta de percepción.

La Figura 5-7 muestra un gráfico radial de las modas obtenidas a partir de cada enunciado asociado a un principio de LeSS, de tal manera que cada eje representa un enunciado de la encuesta y los radios van creciendo desde la opción “muy en desacuerdo” hasta la opción “muy de acuerdo”. Una observación relevante es que ninguno de los enunciados presenta una moda ni con el valor “muy en desacuerdo”, ni con el valor “en desacuerdo”.

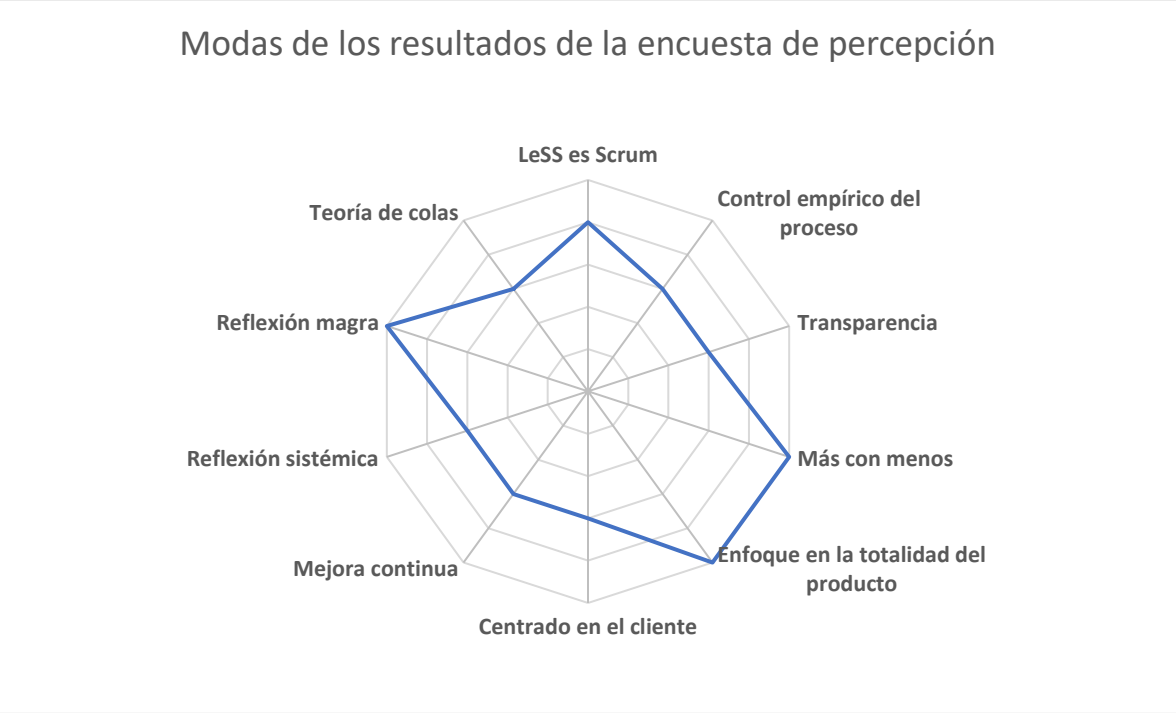


Figura 5-7: Grafico radial de las modas de la encuesta de percepción.

En primera instancia, esto muestra que la implantación del marco de trabajo ha sido exitosa y por lo tanto el objetivo específico se puede considerar logrado. Sin embargo, el enunciado que hace referencia al principio de LeSS relacionado a teoría de colas muestra el peor resultado dentro de la encuesta. Eso significa que a pesar de que se ha podido implantar una filosofía de trabajo centrada en el equipo de desarrollo, todavía se puede mejorar el aspecto de gestión de colas del trabajo por realizar y trabajo en curso.

6 CONCLUSIÓN

En este documento se ha presentado un caso de éxito de implantación del proceso de transformación digital para una empresa mutuo-hipotecaria que le permitirá transformarse en una Fintech o compañía de finanzas que basa su negocio en el uso de la tecnología. A continuación, se presentará un análisis en torno al cumplimiento de los objetivos propuestos al principio del trabajo, los elementos mínimos necesarios para poder tener éxito en un escenario de implementar el proceso de transformación digital y los trabajos propuestos para quienes deseen continuar estudiando en la línea que propuso este trabajo.

6.1 Cumplimiento de objetivos propuestos

En el capítulo 5 de este documento se presentaron los resultados observados desde la perspectiva del impacto para el negocio y también de desde el punto de vista del nivel de adopción del marco de trabajo. Los objetivos presentados en la sección 1.2 hacían referencia a adoptar un marco de trabajo que pueda permitir aumentar la adaptabilidad de la organización y establecer un protocolo con estándares mínimos de calidad.

Los objetivos fueron cumplidos, tanto desde la perspectiva técnica, al lograr implementar un sistema de despliegue continuo que permite prevenir cambios en el software que no cumplan con el estándar mínimo de calidad establecido, como también desde la perspectiva de la adopción del marco de trabajo, reflejado en los resultados de la encuesta de percepción. Sin embargo, se detectó un punto en particular en que el equipo de desarrollo declara no tener tanta adherencia a uno de los pilares fundamentales de LeSS, referente a la forma de priorizar los ítems del *backlog* (ver sección 5.4).

Particularmente, en el caso de estar usando LeSS como marco de trabajo de desarrollo de software, la responsabilidad de ordenar los ítems del *backlog* es parte de las tareas que ejecuta el *product owner* (ver Tabla 2-1). El *product owner* no es parte del equipo de desarrollo (sí es parte del equipo Scrum), lo que podría ser un elemento que necesite mayor esfuerzo para sincronizar las expectativas de lo que percibe el equipo de desarrollo, contra la técnica que usa el *product owner* para priorizar u ordenar los ítems del *backlog*. De la misma forma en que se está evitando la generación de optimizaciones locales o de verticales de especialidades dentro de la organización, este resultado podría indicar que es necesario hacer un esfuerzo por cohesionar o sincronizar el trabajo que hace el *product owner* con respecto a la propuesta de requerimientos y ejecución de tareas que realiza el equipo de desarrollo.

6.1.1 Motivos para elegir Scrum con LeSS

Como se presentó en la sección 2.1.3, las razones para haber elegido Scrum como marco de trabajo basal durante el reordenamiento de la forma de trabajar de la

organización están motivadas en que es justamente Scrum el marco de trabajo más utilizado en la industria del desarrollo de software, esto inmediatamente facilita un proceso de reclutamiento para lograr completitud del equipo de desarrollo menos accidentado, toda vez que en el mercado laboral es más probable encontrar profesionales que sí tengan experiencia usando Scrum y enfrentando problemas complejos.

La decisión de escalar Scrum se respalda en que autores como Jeff Sutherland [2] dan testimonio de que un equipo Scrum funciona de forma óptima cuando está compuesto entre cinco y ocho integrantes. En el apartado 3.2.1 se presentó que el equipo inicial está formado por 12 personas distribuidas en dos células. Dado que cada célula de trabajo corresponde a un equipo Scrum que está trabajando sobre el mismo producto, es necesario formalizar un método para coordinar distintos equipos, lo cual corresponde, en efecto, a hacer escalar Scrum.

La adopción de LeSS está motivada principalmente en su filosofía de “más con menos” y de evitar la burocracia, lo que está en completa concordancia con las ideas propuestas en este trabajo referente a favorecer el *ownership* de la totalidad el producto digital por parte del equipo de desarrollo. En el caso particular de este trabajo, en que la transformación digital “desde cero” y por lo tanto se presenta inmediatamente la oportunidad de comenzar a construir los sistemas manteniendo la premisa de tener menos roles, menos administradores y menos artefactos.

6.2 Requerimientos mínimos

En esta sección se presenta un análisis desde la perspectiva de los elementos mínimos, que son necesarios cumplir para poder tener éxito en el proceso de implementación de transformación digital y, particularmente, en un escenario de pretender adoptar un marco de trabajo que use metodologías ágiles.

6.2.1 Apoyo de la alta gerencia

En este trabajo se ha mencionado varias veces que el equipo de desarrollo necesita tener facultades para definir procesos de negocios y realizar cambios en el producto digital sin tener que depender de otras áreas de la empresa. Una conclusión inmediata entonces es que el proceso de transformación digital sólo puede ser exitoso si el resto de la organización se integra y está dispuesto a trabajar en torno a esta filosofía. Esta afirmación es además respaldada por la literatura revisada en el capítulo 2 [3].

Esto significa que, si no se cuenta con el respaldo de la alta gerencia o al menos de las demás áreas de la organización que están involucradas en la operación y en la continuidad del negocio, el proceso de transformación digital y de implantación de un *framework* agile no podrá ser exitoso. Por ejemplo, en un escenario que se pueda conformar una célula de trabajo *scrum*, por muy capacitados que sean los profesionales que lo conformen o por mucho que tengan una filosofía de trabajo *Agile,* el equipo no podrá causar impacto en el negocio si de todas maneras no tiene

facultades de definir procesos, o debe estar levantando solicitudes para realizar cambios, o simplemente funciona como servicio que atiende a las demás áreas de la organización.

6.2.2 Filosofía de trabajo y perfil de los profesionales

Uno de los elementos necesarios para poder implementar LeSS, es otorgarle pleno poder de diseño, e integración de cambios al equipo de desarrollo [3], lo que obviamente significa que este tendrá más responsabilidades y mayor nivel de exigencia en distintas áreas del conocimiento para asegurar la continuidad del negocio y la mantención del producto digital. Durante el proceso de reclutamiento realizado en la organización, se priorizó la necesidad del equipo de desarrollo de contar con profesionales de distintas especialidades que pudieran complementarse para cumplir el objetivo de empoderar al equipo de desarrollo pero que además hayan declarado abiertamente estar dispuestos a trabajar en un régimen multidisciplinario y abiertos a aprender o adoptar habilidades que no necesariamente estén relacionadas con su área particular de pericia.

Este ejercicio vislumbra que los profesionales necesarios para adoptar este tipo de filosofía de trabajo deben tener un perfil más bien generalista y orientado al sistema completo [3]. Aunque un ingeniero de software tenga muchas habilidades técnicas o una gran maestría al momento de desarrollar software, ese perfil no podría compatibilizar con esta forma de trabajo si no es capaz de tener discusiones orientadas a la continuidad del negocio o en torno a las estrategias de alto nivel.

Por otro lado, a pesar de ser un tema que se escapa del alcance académico de este trabajo, la responsabilidad de evaluar y seleccionar futuros profesionales que puedan integrarse al equipo de desarrollo también será responsabilidad de los mismos integrantes del equipo de desarrollo. En este respecto, existe un acuerdo de trabajo de que en el momento de entrevistar a una persona que postule a trabajar en el equipo de desarrollo, se debe contar con al menos el acuerdo de todos los actuales integrantes de la célula (responsables de la etapa específica del *customer journey*) a la que la persona esté postulando.

6.2.3 Herramientas de trabajo

De la misma forma en que es posible identificar elementos necesarios para realizar la transformación digital dentro de una organización, como lo son el apoyo de las gerencias y el perfil abierto de profesionales que van a formar el equipo de desarrollo, también se hace fundamental contar con herramientas que permitan agilizar el flujo de trabajo y automatizar la mayor cantidad de tareas posibles, de manera que se logre “minimizar el trabajo realizado”. Después de haber tenido un caso exitoso de transformación digital en una mutuaria que apunta a convertirse en una FINTECH, es que se puede asegurar que elementos como automatización de pruebas, versionador de código fuente e integración continua son esenciales para el éxito y adopción de esta nueva forma de trabajar dentro de la organización.

6.3 Trabajos propuestos

Como se describió en la sección 3.2.3, aquellas tareas necesarias para mantener la continuidad del negocio que no han sido implementadas o automatizadas a través del producto digital son responsabilidades que el equipo de desarrollo de be asumir. Esto significa que es necesario que además de gestionar y ordenar los trabajos por realizar referentes a los requerimientos y características que se van a implementar, también se debe tener una “capacidad reservada” para resolver tareas operativas. Un desafío interesante y del cual el marco de trabajo implementado no se hace cargo es buscar una estrategia para minimizar la carga operativa y mantener los procesos de negocio ejecutándose normalmente, sin sacrificar la priorización de los ítems del backlog, o sugerir técnicas para que un incremento entregado no aumente la carga operativa del equipo de desarrollo.

En la industria del software, existe un consenso general con respecto a reconocer las ventajas que da Scrum para enfrentar un problema que caiga en la categoría del “complejo” [7] y también con respecto a la mayor adaptabilidad que puede ofrecer para una organización. Sin embargo, no existe tal consenso con respecto a la forma en que se puede hacer escalar Scrum [7]. En este trabajo se implementó LeSS, pero existen otros marcos de trabajo para hacer escalar Scrum en distintas situaciones o para distintos tamaños de empresas. Una línea interesante de investigación sería entonces poder comparar distintas experiencias con marcos de trabajo de escalamiento de Scrum, como puede ser SaFe o Nexus, para identificar y proponer recomendaciones con respecto a qué estrategia es la mejor para hacer escalar Scrum dependiendo las características de la organización o el escenario en que se está trabajando.

Otra línea de trabajo o experimentación interesante que puede ser explorada en torno a las ideas de darle más protagonismo al equipo de desarrollo que han sido expuestas en este trabajo se encuentra ligada a la forma de medir el desempeño de los integrantes del equipo de desarrollo. En la sección 3.2.1.1 se plantea que la medición del desempeño debe definirse en torno a las metas del negocio. Sería interesante además presentar algún modelo de auto evaluación y evaluación de pares dentro del equipo de desarrollo que sea reflejado en estímulos directos con respecto a las carreras de los profesionales, por ejemplo, bonos de desempeño o requisitos para optar a rangos de compensación superiores. Un factor para tener en cuenta en esta línea de experimentación es poder evitar estímulos a la colusión y también poder presentar formas que aseguren el cumplimiento de acuerdos relacionados del flujo de trabajo (revisión de pares, protección automática del *pipeline* de integración continua, etc.) y correcta ejecución de procesos de selección.

BIBLIOGRAFÍA

[1] K. Patel y M. P. McCarthy, Digital transformation: the essentials of e-business leadership, 2000.

[2] J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, Random House Business, 2014.

[3] C. Larman y B. Vodde, Large-Scale Scrum: More with LeSS, Addison-Wesley Professional, 2016.

[4] G. Brougham, The Cynefin MiniBook, lulu.com, 2015.

[5] S. McConnell, Rapid development: Taming wild software schedules, Microsoft Press, 2000.

[6] W. Royce, «Managing the Development of Large Software Systems,» de *Proceedings*, 1970, pp. 1-9.

[7] M. Cagan, How to Create Tech Products Customers Love, Wiley, 2017.

[8] T. Mayer, The People's Scrum: Agile Ideas for Revolutionary Transformation, Dymaxicon, 2012.

[9] R. Estrada, «Agilidad no es rapidez ni tampoco reducción de costos,» 15 Marzo 2020. [En línea]. Available: https://medium.com/@rodrigo.estrada/agilidad-no-es-rapidez-ni-tampoco-reducci%C3%B3n-de-costos-43ff966f5619. [Último acceso: 20 Mayo 2021].

[10] J. Sutherland y K. Schwaber, «Scrum Guide,» Noviembre 2020. [En línea]. Available: https://scrumguides.org/scrum-guide.html. [Último acceso: 20 Mayo 2021].

[11] V. Grgić, «Descaling Organizations with LeSS,» 8 Mayo 2015. [En línea]. Available: https://less.works/blog/2015/05/08/less-scaling-descaling-organizations-with-less.html. [Último acceso: 21 Mayo 2021].

[12] The LeSS Company, «More with LeSS: Principles Overview,» 2022. [En línea]. Available: https://less.works/less/principles/overview. [Último acceso: 06 Junio 2022].

[13] RedHat, «¿Qué son la integración/distribución continuas (CI/CD)?,» [En línea]. Available: https://www.redhat.com/es/topics/devops/what-is-ci-cd. [Último acceso: 22 Mayo 2021].

[14] V. Dantas, Architecting Google Cloud Solutions: Learn to design robust and future-proof solutions with Google Cloud technologies, Estocolmo: Packt Publishing, 2021.

[15] L. Orsini, «What You Need To Know About Node.js,» 2013 Noviembre 07. [En línea]. Available: https://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/. [Último acceso: 2021 Mayo 23].

[16] A. Hurskiy y S. Merenych, «Front-end JavaScript Frameworks Showdown: Vue vs. React vs. Angular,» 18 Noviembre 2020. [En línea]. Available: https://clockwise.software/blog/angular-vs-react-vs-vue/. [Último acceso: 24 Mayo 2021].

[17] R. Lkert, «A Technique for the Measurement of Attitudes,» de *Archives of Psychology*, New York, 1932, pp. 5-55.

ANEXOS

A. Descripción de Node.js

Documento de presentación que describe las características principales del entorno de ejecución de Node.js presente en el sitio https://nodejs.org

Acerca de Node.js®

Editar en Github

Ideado como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para crear aplicaciones network escalables. En el siguiente ejemplo de "hola mundo", pueden atenderse muchas conexiones simultáneamente. Por cada conexión, se activa la devolución de llamada o *callback*, pero si no hay trabajo que hacer, Node.js se dormirá.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hola Mundo');
});

server.listen(port, hostname, () => {
  console.log(`El servidor se está ejecutando en http://${hostname}:${port}/`);
});
```

Esto contrasta con el modelo de concurrencia más común de hoy en día, en el que se emplean hilos del Sistema Operativo. Las redes basadas en hilos son relativamente ineficientes y muy difíciles de usar. Además, los usuarios de Node.js están libres de preocuparse por el bloqueo del proceso, ya que no existe. Casi ninguna función en Node.js realiza I/O directamente, por lo que el proceso nunca se bloquea. Por ello, es muy propicio desarrollar sistemas escalables en Node.js.

Si alguno de estos términos no le es familiar, hay un artículo completo en Blocking vs Non-Blocking.

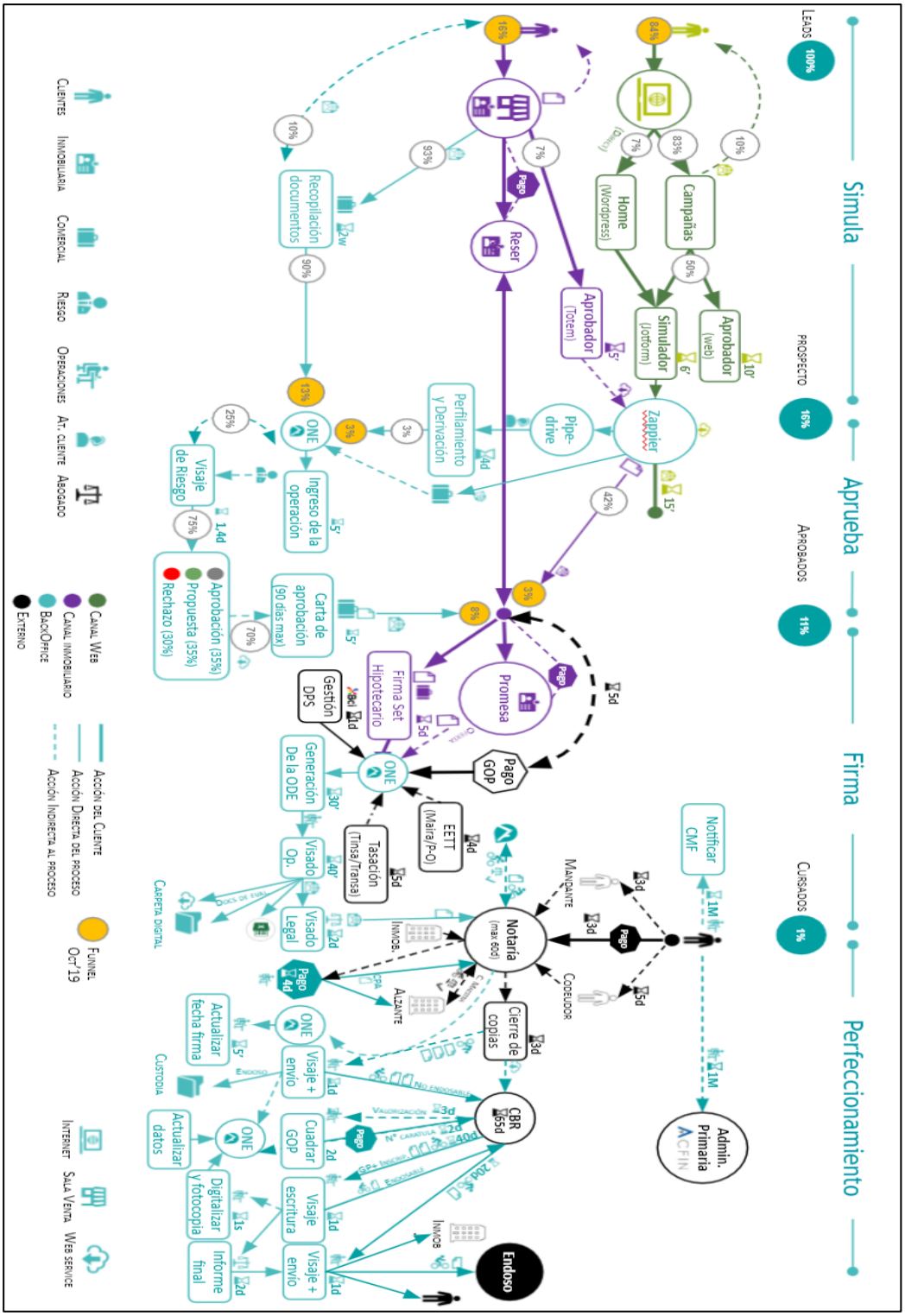
Node.js es similar en diseño y está influenciado por sistemas como Event Machine de Ruby y Twisted de Python. Pero Node.js lleva el modelo de eventos un poco más allá. Incluye un bucle de eventos como runtime de ejecución en lugar de una biblioteca. En otros sistemas siempre existe una llamada de bloqueo para iniciar el bucle de eventos. Por lo general, el comportamiento se define mediante devoluciones *callbacks* de llamada al iniciarse un script y al final se inicia un servidor a través de una llamada de bloqueo como `EventMachine::run()`. En Node.js, no existe como tal la llamada de inicio del evento de bucle o *start-the-event-loop*. Node.js simplemente entra en el bucle de eventos después de ejecutar el script de entrada y sale cuando no hay más devoluciones *callbacks* de llamada para realizar. Se comporta de una forma similar a JavaScript en el navegador - el bucle de eventos está oculto al usuario.

HTTP es un elemento destacado en Node.js, diseñado teniendo en cuenta la transmisión de operaciones con streaming y baja latencia. Esto hace que Node.js sea muy adecuado para la base de una librería o un framework web.

Que Node.js esté diseñado para trabajar sin hilos no significa que no pueda aprovechar múltiples núcleos en su entorno. Se pueden generar subprocesos o procesos hijos utilizando nuestra API `child_process.fork()`, la cual está diseñada para que la comunicación entre ellos sea fácil mediante su proceso principal. Desarrollada sobre esa misma interfaz está el módulo `cluster`, que le permite compartir sockets entre procesos para permitir el balanceo de carga entre sus múltiples núcleos.

B. Diagrama del proceso del negocio antes de la intervención

Esquema presentado a la alta gerencia de la organización en la etapa de diagnostico del trabajo de implantación de proceso de transformación digita, describiendo el *customer journey* o el flujo del negocio antes de la intervención.



C. Reglas de estilo de código

Ejemplo de un archivo ***.eslintrc.js*** que define las reglas de estilos de código de uno usado en uno de los repositorios de software del producto digital.

```
module.exports = {
  parser: '@typescript-eslint/parser',
  parserOptions: {
    project: 'tsconfig.json',
    sourceType: 'module',
  },
  plugins: [
    '@typescript-eslint',
    '@typescript-eslint/eslint-plugin'
  ],
  extends: [
    'plugin:@typescript-eslint/eslint-recommended',
    'plugin:@typescript-eslint/recommended',
    'prettier',
    'prettier/@typescript-eslint',
  ],
  root: true,
  env: {
    node: true,
    jest: true,
  },
  rules: {
    '@typescript-eslint/type-annotation-spacing': ['error', { before: false, after: true }],
    '@typescript-eslint/no-explicit-any': 'off',
    '@typescript-eslint/no-empty-function': 'off',
    '@typescript-eslint/semi': [2, 'always'],
    '@typescript-eslint/indent': ['error', 2],
    'space-infix-ops': ['error'],
    'space-before-blocks': 'error',
    'arrow-parens': ['error', 'as-needed'],
    'object-curly-spacing': ['error', 'always'],
    'array-bracket-spacing': ['error'],
    'no-trailing-spaces': 'error',
    'semi-spacing': 'error',
    'semi': [2, 'always'],
    'indent': ['error', 2],
    'max-len': ['error', { code: 200 }],
    'space-before-function-paren': ['error', { anonymous: "never", named: "never", asyncArrow: "always" }],
    'arrow-spacing': ['error'],
    'key-spacing': 'error',
    'space-in-parens': 'error',
    'no-spaced-func': 'error',
    'keyword-spacing': 'error',
    'comma-spacing': ['error', { before: false, after: true }],
    "comma-dangle": ["error", "never"],
    "no-console": "warn",
    quotes: ["error", "single",  { avoidEscape: true }]
  }
};
```

D. Archivo con definiciones de pruebas unitarias

Ejemplo de pruebas unitarias implementadas usando jest en un repositorio de unos de los servicios web del producto digita.

```
import { Test, TestingModule } from '@nestjs/testing';
import { FeaturesTogglesController } from './features-toggles.controller';

describe('FeaturesToggles Controller', () => {
  let controller: FeaturesTogglesController;
  const featuresTogglesServiceMock: any = {};
  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      controllers: [FeaturesTogglesController],
      providers: [{ provide: 'FeaturesTogglesService', useValue: featuresTogglesServiceMock }]
    }).compile();
    controller = module.get<FeaturesTogglesController>(FeaturesTogglesController);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });
  it('should find feature toggle by name', async () => {
    featuresTogglesServiceMock.findFeatureToggleByName = jest.fn(() => 'feature-toggle');
    const result = await controller.getFeatureToggles({ name: 'name' });
    expect(result).toBe('feature-toggle');
    expect(featuresTogglesServiceMock.findFeatureToggleByName).toHaveBeenCalledWith('name');
  });
  it('shoul get all features toggles', async () => {
    featuresTogglesServiceMock.getAllFeaturesToggles = jest.fn(async () => 'features-toggles');
    const result = await controller.getFeatureToggles({} as any);
    expect(result).toBe('features-toggles');
    expect(featuresTogglesServiceMock.getAllFeaturesToggles).toHaveBeenCalled();
  });
  it('should update feature toggle', async () => {
    featuresTogglesServiceMock.updateFeatureToggle = jest.fn(async () => 'updatedFeatureToggle');
    const options = [{ key: 'key', value: 1 }];
    const isActive = true;
    const result = await controller.updateFeatureToggle({ featureToggleId: 'featureToggleId' }, {
    expect(result).toBe('updatedFeatureToggle');
    expect(featuresTogglesServiceMock.updateFeatureToggle).toHaveBeenCalledWith('featureToggleId',
  });
});
```

E. Pipeline de despliegue continuo

Ejemplo de un archivo con la configuración de un pipeline de despliegue continuo implementado en uno de los repositorios de las piezas de software del producto digital.

```yaml
kind: pipeline
name: creditu-ui

steps:
- name: static-code-analysis
  image: python:3
  commands:
  - pip install nodejsscan
  - nodejsscan -d src/ -o static.scan.json
  when:
    event:
    - pull_request
    - tag
- name: install-dependencies
  image: node:12
  commands:
  - npm install
  environment:
    CYPRESS_INSTALL_BINARY: '0' # No instalamos el binario de cypress para ahorrar tiempo
- name: code-quality
  image: node:12
  commands:
  - npm run quality
  when:
    event:
    - pull_request
    - tag
  depends_on:
  - install-dependencies
```

```yaml
- name: code-style-lint
  image: node:12
  commands:
  - npm run lint
  when:
    event:
    - pull_request
    - tag
  depends_on:
  - install-dependencies
- name: audit-dependencies
  image: node:12
  commands:
  - npm audit --production
  when:
    event:
    - pull_request
    - tag
  depends_on:
  - install-dependencies
- name: unit-test
  image: node:12
  commands:
  - npm run test:unit
  when:
    event:
    - pull_request
    - tag
  depends_on:
  - install-dependencies
- name: install-cypress # Ahora sí instala el binario de cypress
  image: node:12
  commands:
  - npm install cypress@3.8.3
  # Las pruebas end to end corren en otra imagen docker,
  # así que rescatamos archivos para no tener que volver a instalar y poder ahorrar tiempo
  - mv /root/.cache/Cypress ./cypress
  when:
    event:
    - pull_request
    - tag
  depends_on:
  - install-dependencies
```

```yaml
- name: end-to-end-test
  image: cypress/base
  commands:
  - mkdir /root/.cache #Se crea la carpeta en que van los archivos de cypress
  - mv ./cypress /root/.cache/Cypress # Se recuperan los archivos de cypres rescatados en el paso install-dependencies
  - npm run test:e2e -- --headless
  mem_limit: 8589934592
  shm_size: 4294967296
  environment:
    VUE_APP_FUNNEL_URL: http://creditu-internal-backend:3000
    VUE_APP_AMORTIZATION_SCHEDULE_SERVICE_URL: 'https://jhzrh456fe.execute-api.us-east-1.amazonaws.com/amortization-schedule'
    VUE_APP_BACKEND_URL: https://api-dev.cdtu.cl
    VUE_APP_OLD_BACKEND_URL: https://actu.cdtu.cl
    MONGODB_URI: mongodb://mongo:27017/creditu-db #En lugar de localhost, se usa el nombre del service (ver seccion services)
    AWS_ACCESS_KEY_ID:
      from_secret: AWS_ACCESS_KEY_ID
    AWS_SECRET_ACCESS_KEY:
      from_secret: AWS_SECRET_ACCESS_KEY
    VUE_APP_ANALYTICS_BACKEND_URL:
      from_secret: VUE_APP_ANALYTICS_BACKEND_URL
  when:
    event:
    - pull_request
    - tag
  depends_on:
  - static-code-analysis
  - code-quality
  - code-style-lint
  - audit-dependencies
  - unit-test
  - install-cypress
- name: build-qa
  image: node:12
  commands:
  - npm run build:qa
  when:
    event:
    - push
    branch:
    - master
  depends_on:
  - install-dependencies
- name: deploy-qa
  image: python:3
  environment:
    AWS_ACCESS_KEY_ID:
      from_secret: AWS_ACCESS_KEY_ID
    AWS_SECRET_ACCESS_KEY:
      from_secret: AWS_SECRET_ACCESS_KEY
    AWS_DEFAULT_REGION: 'us-east-1'
  commands:
  - pip install awscli
  - aws s3 sync dist s3://internal-qa.cdtu.cl
  - aws cloudfront create-invalidation --distribution-id E3EK1J9RLY7ZDN --paths "/*"
  when:
    event:
    - push
    branch:
    - master
  depends_on:
  - build-qa
```

```yaml
- name: notify-deploy-qa
  image: plugins/slack
  settings:
    webhook:
      from_secret: SLACK_CHAT_WEBHOOK
    channel:
      from_secret: SLACK_CHANNEL
    username: Despliegue qa
    icon_url: https://cdnx.jumpseller.com/the-wiser/image/8333116/resize/635/635?1587586696
    template: >
      Pipeline *{{build.number}}* del repositorio `{{repo.name}}` desplegó exitosamente en ambiente *qa*.
  when:
    status:
    - success
    event:
    - push
    branch:
    - master
  depends_on:
  - deploy-qa
- name: build-prod
  image: node:12
  commands:
  - npm run build:prod
  when:
    event:
    - tag
  depends_on:
  - static-code-analysis
  - code-quality
  - code-style-lint
  - audit-dependencies
  - unit-test
  - end-to-end-test
- name: deploy-prod
  image: python:3
  environment:
    AWS_ACCESS_KEY_ID:
      from_secret: AWS_ACCESS_KEY_ID_PROD
    AWS_SECRET_ACCESS_KEY:
      from_secret: AWS_SECRET_ACCESS_KEY_PROD
    AWS_DEFAULT_REGION: 'us-east-1'
  commands:
  - pip install awscli
  - aws s3 sync dist s3://internal.creditu.com
  - aws cloudfront create-invalidation --distribution-id E3UNU0GGXOE091 --paths "/*"
  when:
    event:
    - tag
  depends_on:
  - build-prod
```

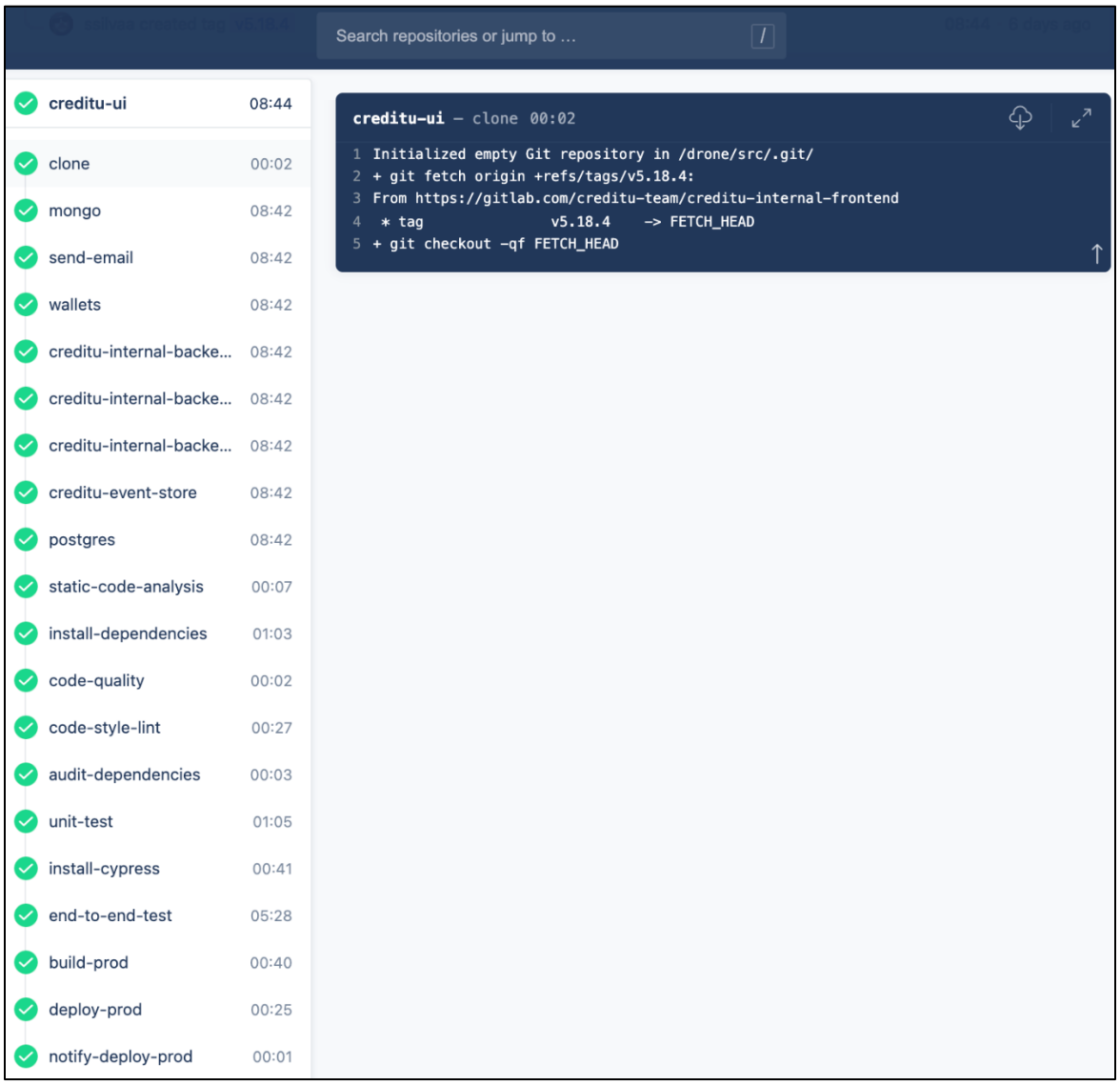
```yaml
  - name: notify-deploy-prod
    image: plugins/slack
    settings:
      webhook:
        from_secret: SLACK_CHAT_WEBHOOK
      channel:
        from_secret: SLACK_CHANNEL
      username: Despliegue prod
      icon_url: https://cdnx.jumpseller.com/the-wiser/image/8333116/resize/635/635?1587586696
      template: >
        Pipeline *{{build.number}}* del repositorio `{{repo.name}}` desplegó exitosamente en ambiente productivo con el tag `{{build.tag}}`.
    when:
      status:
      - success
      event:
      - tag
    depends_on:
    - deploy-prod
  services:
    - name: mongo
      image: mongo
      pull: always
      when:
        event:
        - pull_request
        - tag
    - name: send-email
      image: registry.gitlab.com/creditu-team/send-email
      pull: always
      environment:
        MONGODB_URI: mongodb://mongo:27017/creditu-db
        AWS_DEFAULT_REGION: us-east-1
        AWS_ACCESS_KEY_ID:
          from_secret: AWS_ACCESS_KEY_ID
        AWS_SECRET_ACCESS_KEY:
          from_secret: AWS_SECRET_ACCESS_KEY
      when:
        event:
        - pull_request
        - tag
```

```yaml
- name: wallets
  image: registry.gitlab.com/creditu-team/wallets
  pull: always
  environment:
    MONGODB_URI: mongodb://mongo:27017/creditu-db
  when:
    event:
    - pull_request
    - tag
- name: creditu-internal-backend
  image: registry.gitlab.com/creditu-team/creditu-internal-backend
  pull: always
  environment:
    EVENTS_APP_BACKEND_URL: 'http://creditu-event-store:3000'
    SEND_EMAIL_API_URL: 'http://send-email:3000'
    WALLETS_API_URL: 'http://wallets:3000'
    AWS_CLIENTS_DOCUMENTS_BUCKET_NAME: clients-documents-dev
    AWS_AUDIT_REPORT_BUCKET_NAME: audit-report-dev
    AWS_AUDIT_REPORT_MUTUAL_ADMIN_FOLDER: '/cmf/administrados'
    AWS_LAMBDA_NAME_GENERATE_REPORTS: GenerateExcelTest
    AWS_LAMBDA_REGION: us-east-1
    AWS_S3_BUCKET_REPORTS: generate-excel-test
    AWS_SES_EMAIL_CONFIGURATION_PAYMENTS: credituEmailConfiguration
    AWS_SES_EMAIL_SOURCE_PAYMENTS: rcerda@creditu.com
    AWS_SES_EMAIL_TO_ADDRESS_TEST: rcerda@creditu.com
    CC_PAYMENT_MAILS: broa@creditu.com
    AWS_SES_REGION: us-east-1
    BASE_URL: http://funnel-dev.cdtu.cl
    JWT_SECRET_KEY: '8xshjZF1AsZysS6XJJ68oGCMgsVObIze4TXcn5nXPhEtf5XmDCHvDQ6QNnDuid3FhFb2uURwzORg7xuhPWE
    MONGODB_URI: mongodb://mongo:27017/creditu-db
    NODE_ENV: local
    TEMPLATE_PAYMENT_CONFIRMATION: confirmacion_pago_transaccion_v1
    SERVIPAG_INTEGRATION_PRIVATE_KEY: '-----BEGIN RSA PRIVATE KEY-----{endline}MIIBOgIBAAJBAKDipK6ujFFsC
    SERVIPAG_PUBLIC_KEY: '-----BEGIN PUBLIC KEY-----{endline}MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKDipK6ujFF
    SERVIPAG_CHANNEL_CODE: 604
    AWS_SES_EMAIL_SOURCE_AUTH: hola@creditu.com
    AWS_SES_PORTABILITY_BACKOFFICE_EMAIL: portabilidad@creditu.com
    AWS_ACCESS_KEY_ID:
      from_secret: AWS_ACCESS_KEY_ID
    AWS_SECRET_ACCESS_KEY:
      from_secret: AWS_SECRET_ACCESS_KEY
  when:
    event:
    - pull_request
    - tag
- name: creditu-event-store
  image: registry.gitlab.com/creditu-team/creditu-event-store
  pull: always
  environment:
    PORT: '3000'
    POSTGRESDB_HOSTNAME: 'postgres'
    POSTGRESDB_USERNAME: 'postgres'
    POSTGRESDB_PASSWORD: 'password'
    POSTGRESDB_DB_NAME: 'postgres'
  when:
    event:
    - pull_request
    - tag
```

```yaml
- name: postgres
  image: postgres
  pull: always
  environment:
    POSTGRES_PASSWORD: password
  when:
    event:
    - pull_request
    - tag
image_pull_secrets:
- dockerconfig
```

F. Ejecución de pipeline de despliegue continuo

Captura de pantalla de la interfaz de usuario del sistema de despliegue continuo implementado para integrar cambios en el software del producto digital.



G. Modelo de datos

H. Resultados de la encuesta de percepción

	LeSS es Scrum	Control empírico del proceso	Transparencia	Más con menos	Enfoque en la totalidad del producto	Centrado en el cliente	Mejora continua	Reflexión sistémica	Reflexión magra	Teoría de colas
1	Muy de acuerd	De acuerdo	En desacuerdo	Muy de acuerdo	Muy de acuerdo	Ni en desacuerdo ni de	En desacuerdo	En desacuerdo	Muy de acuerdo	Ni en desacuerd
2	De acuerdo	Muy de acuerdo	Ni en desacuerd	De acuerdo	De acuerdo	En desacuerdo	Ni en desacuerdo	Ni en desacuerdo ni	De acuerdo	En desacuerdo
3	De acuerdo	Ni en desacuerdo ni de acuerd	De acuerdo	Muy de acuerdo	De acuerdo	De acuerdo	En desacuerdo	En desacuerdo	Muy de acuerdo	Ni en desacuerd
4	Muy de acuerd	Ni en desacuerdo ni de acuerd	Ni en desacuerd	De acuerdo	Muy de acuerdo	De acuerdo	Ni en desacuerdo	Ni en desacuerdo ni	De acuerdo	En desacuerdo
5	De acuerdo	De acuerdo	En desacuerdo	De acuerdo	Muy de acuerdo	Ni en desacuerdo ni de	Ni en desacuerdo	Muy de acuerdo	Ni en desacuerdo	Ni en desacuerd
6	De acuerdo	Ni en desacuerdo ni de acuerd	Ni en desacuerd	De acuerdo	De acuerdo	Ni en desacuerdo ni de	De acuerdo	De acuerdo	De acuerdo	De acuerdo
7	Muy de acuerd	Ni en desacuerdo ni de acuerd	Ni en desacuerd	Muy de acuerdo	De acuerdo	De acuerdo	De acuerdo	Muy de acuerdo	Ni en desacuerd	
8	Ni en desacuer	Muy de acuerdo	En desacuerdo	Muy de acuerdo	Muy de acuerdo	Ni en desacuerdo ni de	Ni en desacuerdo	Ni en desacuerdo ni	Muy de acuerdo	Ni en desacuerd
9	De acuerdo	De acuerdo	Ni en desacuerd	Muy de acuerdo	Muy de acuerdo	De acuerdo	De acuerdo	Ni en desacuerdo ni	Muy de acuerdo	En desacuerdo
10	Muy de acuerd	Muy de acuerdo	Ni en desacuerd	Ni en desacuerdo	De acuerdo	Ni en desacuerdo ni de	Ni en desacuerdo	De acuerdo	De acuerdo	Ni en desacuerd
11	De acuerdo	Ni en desacuerdo ni de acuerd	De acuerdo	Muy de acuerdo	De acuerdo	De acuerdo	Ni en desacuerdo	De acuerdo	Muy de acuerdo	Ni en desacuerd
12	Muy de acuerd	De acuerdo	De acuerdo	Muy de acuerdo	Muy de acuerdo	Ni en desacuerdo ni de	Ni en desacuerdo	Ni en desacuerdo ni	De acuerdo	En desacuerdo