UNIVERSIDAD NACIONAL DE LA PLATA

TESINA DE LICENCIATURA

Programa de apoyo para alumnos con experiencia profesional

TÍTULO: Reingeniería de Sistema Bancario Cobol a Sistema Core Bancario moderno
AUTORES: Juan José Barrera
DIRECTOR ACADÉMICO: Dr. Ariel Pasini
DIRECTOR PROFESIONAL: Walter Radzik
CARRERA: Licenciatura en Sistemas

Resumen

Desde hace unos años algunos bancos emprendieron un proceso de renovación de su sistema central desarrollado en Cobol, con el fin de disponer de la tecnología adecuada para soportar los objetivos estratégicos. El objetivo de esta tesina es realizar una actualización por medio de una reingeniería del sistema Cobol hacia un sistema moderno de aplicaciones core, que permita a un banco particular cubrir las necesidades de negocio y disponer de una plataforma flexible que facilite la satisfacción de requerimientos actuales y futuros.

Palabras Clave

Core, Cobol, Mainframe, sistemas bancarios, reingeniería, migración

Conclusiones

Se desarrollo e implemento un proceso de reingeniería adecuado a las características del banco, que facilito la migración de sistemas monolíticos que se encontraban desarrollados en Cobol, a un modelo de aplicaciones core Bancarias, para hacer frente a las distintas necesidades actuales de los clientes.

Trabajos Realizados

- *Investigación sobre los problemas en la actualidad en los sistemas implementados en Cobol.*
- *Investigación sobre las características que tienen los sistemas implementados en aplicaciones core.*
- *Implementación de los pasos de reingeniería sobre un requerimiento del banco, en el proceso de migración del sistema cobol hacia el sistema de aplicaciones core.*

Trabajos Futuros

- *Continuar avanzando con el proceso de reingeniería en la migración de componentes que aún se encuentran en el sistema cobol.*
- *Migrar la totalidad de los servicios a la nube, para lograr que el banco sea completamente digital.*

Fecha de la presentación: **12/2022**

Agradecimientos

A toda mi familia, en especial a mis padres Luís y Leonor por brindarme el apoyo y contención a lo largo de toda la carrera.

A mi señora Belén por el amor, comprensión, y por acompañarme en todo momento, e impulsarme a seguir adelante.

A mi director Ariel, mi más amplio agradecimiento por su paciencia ante mi inconsistencia, por su valiosa dirección y apoyo para seguir y llegar a la conclusión del mismo.

A mi director profesional Walter por brindarme las herramientas necesarias para poder realizar este trabajo.

A la Facultad de Informática por brindarme la formación y darme las herramientas necesarias para poder desempeñarme como profesional.

Dedicada especialmente a mi hijo Samuel, que lo amo con todo mí ser.

Contenido

Índice de Figuras

Índice de Tablas

Índice de acrónimos

Core: Conjunto de aplicaciones que utilizan el concepto de *Centralized Online Real-Time Exchange*

COBOL: Common Business-Oriented Language

FORTRAN: contracción del inglés The IBM Mathematical Formula Translating System

ATM: Automated Teller Machine

POS: Point of Sale

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

GIT: Herramienta de control de versiones de código

BI: Banca Internet

CICS: Customer Information Control System

DB2: Motor base de datos relacional

System/32: Equipo de IBM para medianas y pequeñas empresas.

AS 400: Equipo de IBM para todo tipo de empresas

RPG: Lenguaje de negocios desarrollado por IBM

Capítulo 1 - Introducción

Objetivo

Describir el proceso de reingeniería de la operatoria de una institución financiera que se encuentra en lenguaje Cobol hacia un sistema digital moderno en donde toda la funcionalidad es implementada en aplicaciones de "Core Bancario".

Motivación

Los sistemas o aplicaciones bancarias han venido evolucionando junto con la tecnología de la época correspondiente, pues deben estar a tono con lo que el negocio busca, que es ofrecer más y mejores productos y servicios a menores costos.

Actualmente muchos bancos funcionan con software heredado, son sistemas bancarios centrales que se crearon entre los años 70 – 80, en donde la tecnología Cobol es inflexible para las sucursales (sistemas internos de las sucursales), la aplicación móvil y el sitio web.

Se necesita pasar por una transformación digital, reemplazando toda esa tecnología antigua con un ecosistema digital moderno que brinda servicios rápidos y sin problemas.

Cada día más Bancos eligen migrar toda su operatoria hacia nuevas tecnologías que le permiten competir en un mundo cada vez más digital, cambiar la forma de proveer los servicios financieros y satisfacer las necesidades de los consumidores de una manera diferente.

Además, permite generar eficiencias en los procesos internos y mejorar los tiempos de respuestas.

Desarrollo propuesto

Para cumplir el objetivo de la tesis se desarrollará un marco teórico con los lineamientos para implementar una solución que permita:

- Describir el modelo del Core Bancario y su aplicación.
- Analizar los distintos Cores Bancarios existentes en el mercado, junto con las diferencias que hay entre sistemas bancarios tradicionales y sistemas implementados por aplicaciones core Bancarios.
- Describir las necesidades de la institución financiera en el marco de un sistema digital moderno de mejora en la satisfacción del cliente.
- Analizar el proceso de reingeniería aplicado en la migración del banco.
- Relevar y analizar la integración, entrega y deploys continuos que BitBucket, Bamboo y Harness nos permite aplicar.
- Analizar las ventajas obtenidas por la institución financiera en el marco del proceso de mejora con la utilización de las aplicaciones core.

Resultado esperado

Los bancos mediante la utilización aplicaciones de "Core Bancario" podrán gestionar mejor sus operaciones (normalmente las operaciones de Core bancario incluyen: cuentas de depósito, préstamos, hipotecas y pagos), accediendo en línea y tiempo real, a través de cualquier canal (sucursales, agencias, cajeros automáticos ATM, puntos de venta POS, Banca por internet, dispositivos móviles) a una fuente de datos centralizada, de tal forma que cualquier transacción es reflejada inmediatamente en los sistemas y bases de datos de la Entidad Financiera.

Por otro lado, la utilización del Core Bancario permitirá realizar un mejor análisis de las distintas funcionalidades del sistema por parte de los analistas funcionales y programadores, encargados de mantener y agregar funcionalidad al sistema mediante la reingeniería.

Esto traerá como consecuencia un mejor uso de los recursos del sistema, con lo cual se obtendría una gestión operativa más eficiente y eficaz, lo que obtendrá mejores tiempos de respuestas de cara al cliente, y de manera paralela un sistema con una interface más amigable para el usuario.

Contexto.

La presente tesina se presenta en el contexto del Programa de Apoyo para Alumnos con Experiencia Profesional (PAEP).

En el año 2008 ingresé a trabajar en el Banco Galicia, integrando el equipo de Moras, con el objetivo de realizar tareas de desarrollo y diseño en el sistema del banco. Las tareas de desarrollo y diseño fueron realizadas en Mainframe (COBOL, CICS, DB2).

En el año 2011 integre el equipo de Desarrollo de ISBAN, el cual realizaba el mantenimiento del Banco Santander Rio. Se realizaron tareas de mantenimiento, análisis técnico y funcional de los requerimientos.
Las tareas fueron realizadas en Mainframe (COBOL, DB2) y JAVA.
Desde el año 2014 integro el equipo de Desarrollo de un Banco Nacional, realizando la migración del Core Bancario que se encuentra en Mainframe COBOL a un Sistema de aplicaciones core.

Estructura del documento

Esta tesina está organizada en seis capítulos, siendo este primero el correspondiente con la introducción y presentación del tema.

En el Capítulo 2 se definen los conceptos generales sobre sistemas bancarios, se detalla la evolución de los distintos sistemas bancarios a lo largo de la historia. Se analizan los sistemas implementados en el lenguaje Cobol, características, ventajas que han tenido a lo largo del tiempo. Luego se analizan los sistemas implementados en Mainframe, características, descripciones y tipos de trabajos que pueden ser realizados sobre ellos. En el final del capítulo se enumeran los distintos problemas que tiene en la actualidad los sistemas Cobol Mainframe.

En el Capítulo 3 se describen los Cores Bancarios, se da su definición, ventajas y características. Luego se detallan los componentes de un Core Bancario, describiendo en forma teóricas las distintas arquitecturas para luego pasar a detallar la arquitectura del banco. En el siguiente paso

se enumeran los motivos por el cual elegir los sistemas de aplicaciones core. Al final del capítulo se presenta una comparación de las distintas aplicaciones core existentes en el mercado.

En el Capítulo 4 se realiza una explicación teórica de los distintos tipos de rejuvenecimiento del software. Luego se describen los beneficios de aplicar reingeniería en un sistema de software. Sobre el final del capítulo se detalla el proceso de reingeniería utilizado y el contexto en el cual fue aplicado por el banco.

En el Capítulo 5 se mencionan los servicios en los cuales se apoya el banco, se realiza una breve descripción de los ambientes de trabajo, las distintas metodologías y herramientas utilizadas en las entregas por parte de los equipos del banco. Luego se describe en forma detallada el proceso de reingeniería para el requerimiento de apertura de cuenta. Sobre el final del capítulo se enumeran operaciones nativas y customizaciones realizadas sobre los módulos de Cuenta y Transferencias.

En el Capítulo 6 se presentan las conclusiones a las cuales se ha arribado y los trabajos futuros.

Capítulo 2 – Sistemas Bancarios

Introducción

En este capítulo se realiza un repaso por la evolución de los sistemas bancarios, desde la década de los 60 hasta la actualidad. Luego se describe el lenguaje Cobol, lenguaje más utilizado a lo largo de la historia por los sistemas bancarios. Se hace hincapié en los Mainframes, equipos muy potentes que unidos al lenguaje Cobol fueron la base de la mayoría de los sistemas bancarios. Sobre el final del capítulo se comentan los problemas en la actualidad que tienen los sistemas desarrollados en Cobol.

Historia de los sistemas bancarios

Los primeros sistemas bancarios surgen con la introducción de la informática en la industria Bancaria entre las décadas de los 60 y 70, con el fin de compartir y aumentar la velocidad del intercambio de información entre las sucursales del banco.
Durante estos años cada sucursal solía tener un servidor local que contenía datos del cliente, así como datos de cuentas.
Los servidores estaban fuera de línea y por lo tanto se encontraban desconectados entre sí, de manera tal que se intercambia la información al final de cada día hábil en forma de lotes o procesamiento Batch(programas sin supervisión directa del usuario), los cuales eran procesados por cada servidor, quienes sincronizaban sus propios datos con la información de otras sucursales bancarias.
La tecnología se basa en grandes computadoras o Mainframes los cuales tienen un costo alto de mantener.
Sólo las grandes organizaciones con enormes volúmenes diarios de trabajo administrativo pueden permitirse dichos costes. La tecnología es compleja y requiere personal especializado por lo general ingenieros o matemáticos.

En esta década, se comienzan a utilizar transistores en lugar de tubos de vacío para procesar la información, ya que eran más pequeños y confiables. La información se comienza a guardar en anillos magnéticos.

Durante este periodo surge el lenguaje ensamblador (se asemeja un poco más al lenguaje natural), el cual dio lugar a los primeros lenguajes de programación: COBOL Y FORTRAN.

La interacción con la máquina se seguía realizando mediante tarjetas perforadas. Durante esta década la empresa IBM tuvo una gran relevancia ya que lanzaría dos de los modelos más conocidos de esta generación: el IBM 1401 y el IBM 1620 (Héctor Sánchez San Blas, 2021).

Algo a destacar en la década es que COBOL permite la entrada a no-matemáticos y no-ingenieros al desarrollo del software, ya que su simpleza y su similitud con el lenguaje ingles permitía programar a aquellas personas no especializadas (por lo general hasta ese momento eran matemáticos o ingenieros).

En la década del 80 surge el termino de automatización bancaria a partir de la idea automatizar los procesos de back office (sección relacionada con la gestión del banco tales como

contabilidad, logística, etc.), en donde el objetivo era centralizar las operaciones de datos y como consecuencia aumentar la velocidad y rentabilidad.

En ese momento había un servidor central que alojaba la aplicación y datos del sistema bancario. En cada una de las sucursales del banco se podían encontrar diversas terminales dispersas que de algún modo se conectaban a dicho servidor. Un claro ejemplo son las computadoras centrales de IBM (por ejemplo, System/32 en la década de 1970 y AS 400 a fines de 1980) que fueron programadas con el lenguaje RPG (lenguaje de IBM para generar informes comerciales o de negocios) multiparadigmatico de IBM. Su propósito era, por un lado, hacer frente a la creciente cantidad de datos procesados y, por otro lado, admitir nuevas transacciones bancarias.

En los años siguientes, los sistemas bancarios siguieron creciendo constantemente en lo que respecta a funcionalidad, de acuerdo con las distintas necesidades y requisitos por parte del sector financiero y bancario.

Es importante destacar que en esta década surgen los primeros servicios de cajeros automáticos en línea 24 horas al día en las entidades bancarias.

A fines de los 80, con la llegada comercial de Internet impactó significativamente los aspectos tecnológicos de los sistemas bancarios, lo que a su vez impulso a realizar más transacciones bancarias a través de canales electrónicos, ahora con una velocidad mucho mayor que antes.

Como consecuencia de la aparición de Internet a fines de los 80 y principios de los 90, los bancos comenzaron a ofrecer canales de banca en línea a sus clientes para que puedan realizar operaciones bancarias fácilmente desde sus hogares. Esta tendencia fue creciendo exponencialmente después del año 2000, lo que llevo a que los sistemas bancarios respaldaran tecnológicamente las operaciones de cuentas y las transacciones casi en tiempo real.

Surgen los primeros sistemas orientados al cliente los cuales son flexibles y escalables. Esto trajo como consecuencia cambios en las nuevas arquitecturas de cliente servidor, con varios clientes según el grupo de usuarios (empleado del banco o cliente del banco) y sus necesidades. El sistema hibrido comienza a no poder hacer frente ante las tendencias emergentes como la banca en la nube, la banca móvil y el sistema integrado (Gruber, 2019).
En la Figura **1** se muestra en forma resumida la evolución de los sistemas bancarios.

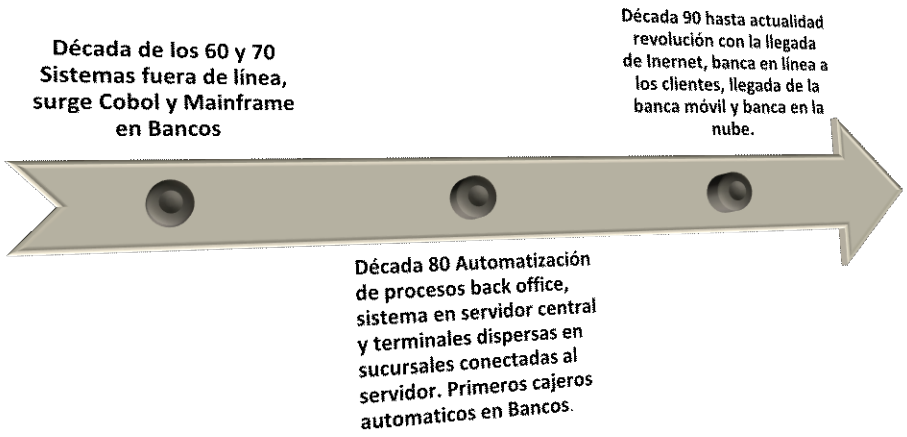

Figura 1 Evolución Sistemas Bancarios

Lenguaje COBOL

El lenguaje de programación COBOL, es un lenguaje de programación de alto nivel (acrónimo de *COmmon Business-Oriented Language*, Lenguaje Común Orientado a Negocios) nació en el año 1959. Es un lenguaje de programación puramente orientado a negocios, es por eso que fue construido principalmente para implementar sistemas fuertemente relacionados con el ámbito financiero, económico y administrativo. Independiente de la plataforma donde se encuentra corriendo, es decir que no depende del sistema operativo. Puede conectarse a cualquier base de datos existente en el mercado. Se adapta a la tecnología cliente-servidor, a la tecnología de eventos e incluso puede estar en la web.
Su principal fortaleza se relaciona con la velocidad de cálculo numérico y el grado de precisión. Esto último sumado a que puede manejar una gran cantidad de datos lo hace muy atractivo para el manejo de la operatoria en los bancos (Valenciano, s.f).

Características principales de COBOL

El lenguaje Cobol fue un pionero del desarrollo en los sistemas bancarios. En la actualidad, muchos de los bancos conservan su sistema o parte del sistema en Cobol, debido a la complejidad y criticidad de migrar muchos de sus desarrollos. Muchos de esos desarrollos de sistemas bancarios fueron realizados en Cobol porque el lenguaje se distinguía por las siguientes ventajas y características:

1. Es un lenguaje autodocumentado: COBOL fue creado con la idea que sería un lenguaje accesible para no programadores, o sea se hablaba de que estos pudieran revisar el código sin tener conocimientos de programación, la idea a pesar de que no fue completamente efectiva, hizo que cobol se convirtiera en el lenguaje autodocumentado más fácil de entender.
2. Es un lenguaje simple con una funcionalidad limitada: Cobol a diferencias de otros lenguajes como C, no tiene punteros ni funciones ni tipos definidos por el usuario. Gracias a la similitud

con el lenguaje inglés, distintos desarrolladores pueden entender con facilidad el código y modificarlo sin ningún problema.

3. Es portable: Cobol puede ser llevado a todo tipo de máquinas por ejemplo Windows, UNIX, OS/2 entre otros. Su capacidad de funcionamiento en casi todas las plataformas ha dotado a COBOL de una magnífica portabilidad, esta es la principal razón por la que en el ámbito de los negocios COBOL está tan extendido. Fue creado en una época en la que las diferentes arquitecturas y modelos de plataformas hardware dificultaban mucho la portabilidad de aplicaciones y programas. Por ello COBOL tuvo tanta difusión.
   COBOL surge con la idea de terminar con los problemas de compatibilidad y portabilidad, por ellos es tan sencilla la migración de sistemas COBOL a nuevas plataformas de hardware más potentes y sin necesidad de realizar grandes modificaciones de código.
4. Es mantenible: COBOL es fácil de entender debido a que es un lenguaje de gran legibilidad. Esto hace que programas de varias líneas sean fáciles de entender y mantener (Valenciano, s.f).

Mainframe

Lo primero que debemos saber, es a que nos referimos cuando se habla de Mainframe, para esto se tomaron las siguientes definiciones y características:

Podríamos definir al Mainframe como una computadora grande, donde las empresas o bancos alojan sus bases de datos, servidores transaccionales y aplicaciones que requieren mayor nivel de seguridad del que podrían ser proporcionadas por maquinas más pequeñas. Pensemos en la necesidad de los bancos en realizar transacciones monetarias, manejar información de clientes, etc. Todo esto debe ser manejado bajo un gran nivel de seguridad.

Además de la seguridad, por medio de los Mainframe podemos obtener una enorme capacidad de procesamiento de trabajos, lo cual su uso los hace muy beneficioso en los bancos, ya que estos procesan millones de transacciones diarias.

Los datos en el Mainframe son guardados en un único repositorio, es decir en el mismo Mainframe con lo cual podemos decir que se tiene una forma de procesamiento centralizada, de esta manera se evitan conflictos en actualizaciones o problemas de congruencia entre los datos.

Características principales de Mainframe

En los grandes sistemas Mainframe, se deben tener en cuenta tres conceptos:

Fiabilidad, disponibilidad, facilidad de mantenimiento.

- Fiabilidad: Mientras un Mainframe está funcionando es capaz de autodetectar, corregir y notificar diferentes fallos que pueden ocasionar un mal funcionamiento del sistema.
- Disponibilidad: Es el tiempo en que el sistema está funcionando. El Mainframe tiene la capacidad de que al momento de producirse un fallo entonces se pueda aislar las partes afectadas y seguir funcionando con capacidad limitada. De esta manera se logra una alta disponibilidad de funcionamiento por parte del sistema.

- Facilidad de mantenimiento: Es la rapidez o la sencillez que tiene el sistema de ser reparado o mantenido. Gracias a las excelentes capacidades de autodiagnóstico y autorecuperación del Mainframe, esta máquina es muy sencilla de mantener. La propia máquina puede detectar un daño antes de que el componente en cuestión falle y se venga abajo causando un problema mayor. En vez de eso, la máquina detecta dicho daño y lo notifica al servicio técnico para que pueda reparar el fallo antes de que ocurra. Esta capacidad es una de las razones por las que un Mainframe asegura la alta disponibilidad (Madariaga, 2016).

Combinar Mainframe con Cobol llevo a que podamos manejar volúmenes grandes de datos, con el plus que la codificación en Cobol es sumamente sencilla. Es por eso, que hoy en día Cobol tiene multitud de presencia en el ámbito de los negocios, sobre todo en grandes negocios que requieren una gran capacidad de procesamiento por lotes (batch). Grandes empresas y sectores de banca que disponen de sistemas Mainframe utilizan Cobol para explotar al máximo las cualidades de este lenguaje de programación.

Cobol es versátil, es capaz de hacer desde un simple “Hola Mundo” hasta el más pesado de los “batch” pasando por los más modernos y actuales modelos de Cliente Servidor de internet. Principalmente maneja dos tipos de trabajos los JOB Batch y las Transacciones Online (Figura **2** y Figura **3**)

Tipos de trabajo en sistemas Mainframe

- JOB Batch(asincrónico): se envían al computador, se leen y procesan los datos.
    - No hay interacción con el programador, siendo este quien ejecuta el proceso y espera los resultados (finalización correcta o cancelación del proceso con error).
- Transacción Online(sincrónico): acceso compartido a datos entre todos los usuarios online.
    - Hay interacción entre el usuario, programador o empleado del banco que este accediendo a la transacción, ya que es el encargado de ingresar los datos en la transacción.

Usos comunes para los trabajos Batch (JOB):

- Generalmente, por la noche se ejecutan los procesos batch que consolidan las transacciones online que fueron realizadas durante el día.
- Mediante los procesos Batch se generan todo tipo de reportes, por ejemplo, reportes de tarjetas, cuentas, clientes, depósitos, prestamos, etc.
- Los procesos Batch permiten realizar los respaldos de archivos y base de datos.

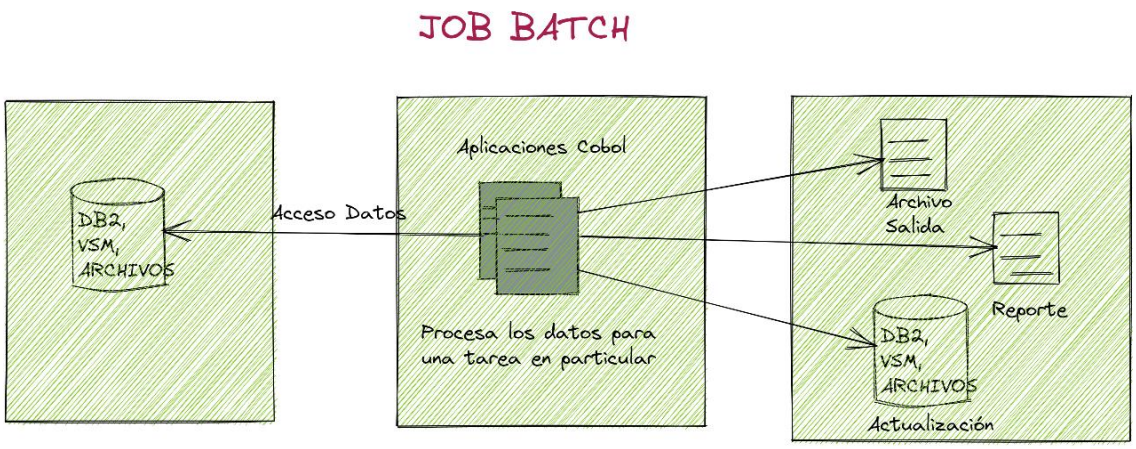

Figura 2 JOB BATCH

Usos comunes para trabajo Online:

- La utilización del cajero ATM por parte de un cliente.
- Un empleado del banco accediendo a una consulta sobre la operatoria de cualquier producto del banco.
- Analista funcional modificando las transacciones, ya sea para mejorar o crear un nuevo producto.

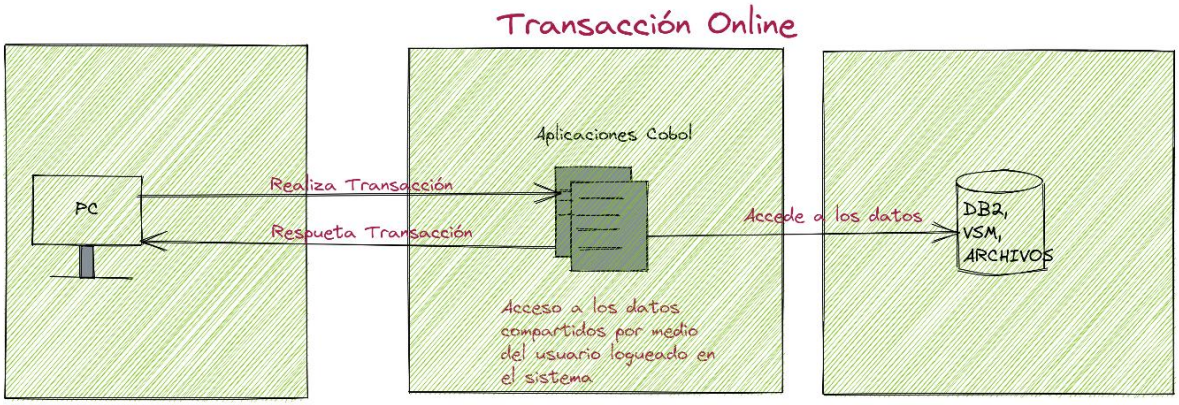

Figura 3 TRANSACCIÓN ONLINE

Problemas en la actualidad

En la actualidad los bancos tienen el gran desafío de convivir con los llamados sistemas heredados. Estos sistemas fueron desarrollados hace décadas, normalmente escritos en

lenguaje Cobol. Ante el avance de la tecnología y del negocio bancario, en la actualidad los bancos ven la necesidad de migrar hacia otros sistemas más modernos encontrándose con algunos problemas que se pasan a enumerar y describir.

- Falta de documentación. La falta de documentación es uno de los principales problemas al que se enfrentan en la actualidad los desarrolladores Cobol. Un desarrollador puede tener que agregar funcionalidad de programas que hace 20 o 30 años fueron creados. A menudo, los desarrolladores suelen encontrarse con especificaciones de diseños que no reflejan los cambios continuos que se dieron en esos 20 o 30 años. Esto es debido a la cantidad de cambios continuos que fueron realizándose en el sistema lo cual incrementa considerablemente la complejidad. (ALAN MARQUEZ ESCORCIA, 2018)
- Reescribir código. El código viejo ha sido usado, probado; se han detectados errores que fueron corregidos. Cuando se cambia el código y se empieza desde cero, no solo se está perdiendo el código viejo sino todo ese conocimiento técnico/funcional, el cual ha probado ser confiable a lo largo de la vida de este tipo de sistemas. Además, la problemática asociada a ellos, es su evolución, básicamente porque el nuevo sistema debe cubrir no sólo los nuevos requisitos, sino que además deberá cubrir los requisitos que cubría el sistema heredado (ALAN MARQUEZ ESCORCIA, 2018).Por lo tanto, no existe una forma directa de especificar un nuevo sistema que sea funcionalmente idéntico al que ya se utiliza. Migrar los sistemas heredados y reemplazados con software moderno conduce a grandes riesgos de negocios.
- Los sistemas de software heredado fueron desarrollados hace varias décadas y han sido modificados de manera continua para que satisfagan los cambios en los requerimientos de los negocios y plataformas de computación. El incremento de tales sistemas es causa de dolores de cabeza para los bancos, a los cuales resulta costoso mantenerlos y riesgoso hacerlos evolucionar (Pressman, 2010).
- También debemos considerar que muchos de estos sistemas son críticos, e inclusive muchos de ellos funcionan las 24 horas del día. Esto quiere decir, que la detención de dichos sistemas puede traer grandes pérdidas en dinero.
- La integración de los sistemas heredados. Las integraciones de los sistemas heredados con sistemas de información más recientes suponen un gran esfuerzo debido a que las interfaces de comunicación no están bien definidas o son tecnológicamente obsoletas. Otro problema, es que cada sistema posee una visión en particular del dominio de la aplicación al cual pertenece, incluye modelos de información, y también estructuras de datos para ese dominio.
- Escasez de profesionales en Cobol. Los sistemas que generalmente fueron construidos en Cobol, el desafío de mantenimiento y evolución del sistema es aún mayor, ya que con el correr del tiempo la cantidad de expertos(tanto programadores como analistas funcionales) en la implementación de dicha tecnología ha decaído considerablemente, por lo que, lidiar con código fuente desconocido y poco familiar, que generalmente está construido por personas que no están disponibles para ser consultadas y, que además en su construcción no han utilizado métodos y técnicas modernas de programación. A este punto también es importante agregar el poco interés que tienen los jóvenes en programar en una tecnología tan antigua, con una interfaz poco amigable. Los jóvenes prefieren lenguajes más modernos, innovadores y avanzados, los cuales tienen una mayor demanda (ALAN MARQUEZ ESCORCIA, 2018).

Cobol sobrevive como lenguaje de programación porque hay demasiado código legado cuya reescritura es costosa tanto en tiempo y dinero. Por lo tanto, los sistemas heredados incluyen muchos cambios hechos a lo largo de su ciclo de vida. Muchos equipos de trabajo estuvieron involucrados para cumplir estos nuevos requerimientos y es inusual para cualquier persona tener un conocimiento completo del sistema. Además, la documentación actual no refleja los cambios que se realizaron en el sistema a lo largo de su ciclo de vida.

Capítulo 3 – Core Bancarios

Introducción

En este capítulo se da una visión de lo que es un Core Bancario, sus objetivos y características. Luego se detallan los componentes arquitectura y módulos de un Core Bancario, y a continuación se define la arquitectura y módulos utilizados en el proceso de reingeniería. En el paso siguiente se describen los motivos por cual elegir los sistemas de aplicaciones core en el desarrollo de sistemas bancarios.
Sobre el final del capítulo se presenta una comparación entre los distintos módulos o funcionalidades proporcionadas por las distintas empresas que proveen soluciones sobre sistemas de aplicaciones core.

Core Bancario

Aplicando el concepto de CORE (de las siglas en inglés Centralized Online Real-Time Exchange y del español “Intercambio en tiempo real centralizado”) a los sistemas bancarios, se define *Core bancario,* como el negocio desarrollado por una institución bancaria con sus clientes minoristas y pequeñas empresas. Muchos bancos tratan a los clientes minoristas como a sus clientes de "Core bancario", y tienen una línea de negocios separada para gestionar las pequeñas empresas. Las grandes empresas son administradas a través de la división de Banca Corporativa de la institución. Normalmente las funciones del Core Bancario incluyen manejo de cuentas, manejo de clientes, depósitos, prestamos, pagos, tarjetas de créditos, y muchas otras más operaciones (TEMENOS, s.f).

Haciendo referencia a la definición anterior, podemos decir que un Core Bancario es el negocio que desarrolla un banco, es decir que es la actividad tradicional de un banco, por lo que generar depósitos, otorgar préstamos, atender transacciones de pago a los clientes forman parte de las operaciones centrales o centralizadas del banco. Esto se debe a que, en los años 70, 80 las sucursales tenían sus servidores en el mismo banco, con lo cual un cliente que requería realizar una operación (sacar dinero, consultar saldo de su cuenta, realizar un depósito, etc.) debía hacer su operación en la sucursal en la cual residía su cuenta. Con la llegada de los cajeros provoca la necesidad de tener sistemas que manejen los datos del sistema en tiempo real.

En la actualidad la mayoría de los bancos realizan sus operaciones en todas sus sucursales, accediendo a fuentes de datos centralizadas, con lo cual por ejemplo los depósitos realizados se reflejan inmediatamente en el sistema del banco y el cliente puede retirar el dinero depositado en cualquiera las sucursales del banco. Hace unas décadas una operación en una cuenta como la mencionada anteriormente tomaba un día en ser reflejada, ya que las sucursales tenían servidores locales, y los datos se enviaban en proceso de lotes(Batch) al final del día.

A lo largo del trabajo utilizaremos el termino Core como el conjunto de aplicaciones que implementan operaciones nativas sobre el modelo de negocio bancario.

Sistema Bancario tradicional

El Core Bancario tiene una actividad central o tradicional que son los depósitos, prestamos, pagos a cliente. Toda esta operatoria es realizada por los clientes.

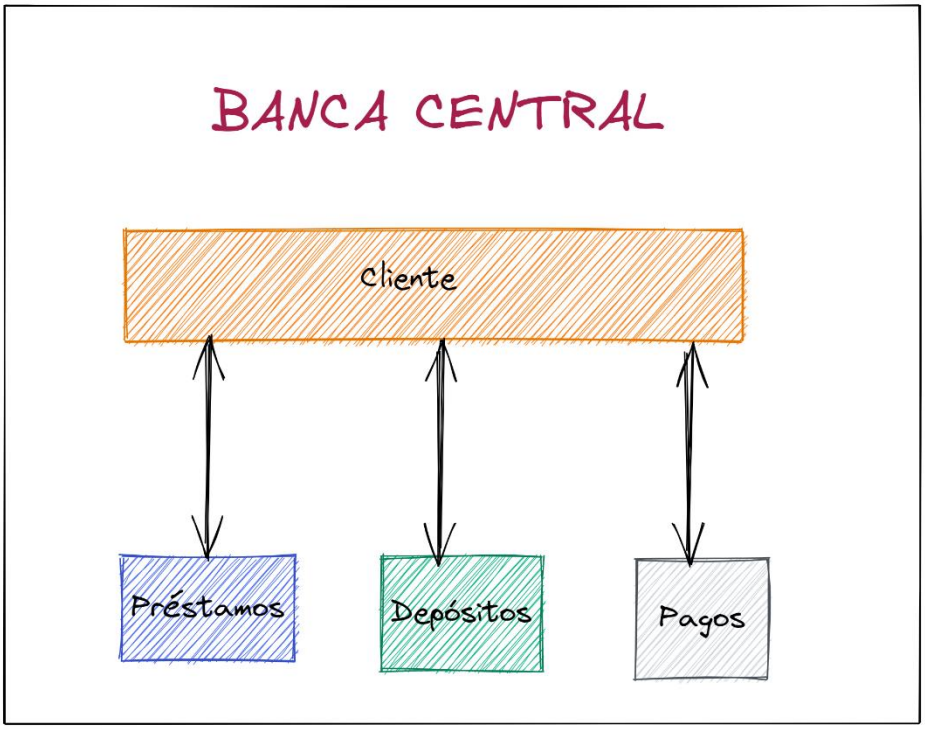

Figura 4 Sistema Bancario Tradicional

En el caso de los préstamos, el sistema necesitaba calcular las tasas de interés, las cuotas, etc. Para los depósitos calcular los intereses de acuerdo a los días, las tasas, etc. Para los pagos se tiene la necesidad de liquidar los pagos con tarjeta, liquidar los pagos realizados por medio de transferencias online. Una característica común seria que un Cliente quiera consultar/modificar algunos de sus datos personales (nombre, dirección, correos, etc.). Todas estas operaciones forman parte de lo que se denomina Sistema Bancario Central Tradicional (ver Figura **4**).

Con el avance de Internet y el auge de la tecnología de la mano de las tablets y celulares en la actualidad, se observó que no se podía mantener toda la operatoria del banco en un sistema bancario tradicional, debido a la gran cantidad de transacciones que se generan diariamente desde cualquier dispositivo como el celular, tablet o PC. Ante esta necesidad surge la creación del canal que comúnmente llamamos Banca Internet (BI), la cual contiene los canales Online (web desde una PC), Móvil (celular, tablets). Con la BI se produce un cambio sin retorno en la forma en la que el cliente accede a su banco y a su dinero, con esta modalidad el cliente adquiere un control y ventajas desconocidas hasta el momento. Internet se convierte en el punto de acceso prioritario al banco (90% de las operaciones pueden ser realizadas) y el acceso a las sucursales pierde peso (10% de las operaciones que no pueden ser realizadas desde BI).
Con Móvil a través de una aplicación el cliente pude llegar a satisfacer todas sus necesidades bancarias, excepto la de retirada o depósito de efectivo.

Para no realizar grandes cambios sobre la tecnología bancarias existentes, lo que hicieron los bancos fue agregar un nuevo canal denominado Banca Internet (BI) que contiene a los canales Online y Móvil los cuales tienen su propia base de datos, manejan lenguajes más modernos que Cobol, y la particularidad de estos canales es que solo manejan la operatoria de clientes que pertenece a BI ver Figura 5.

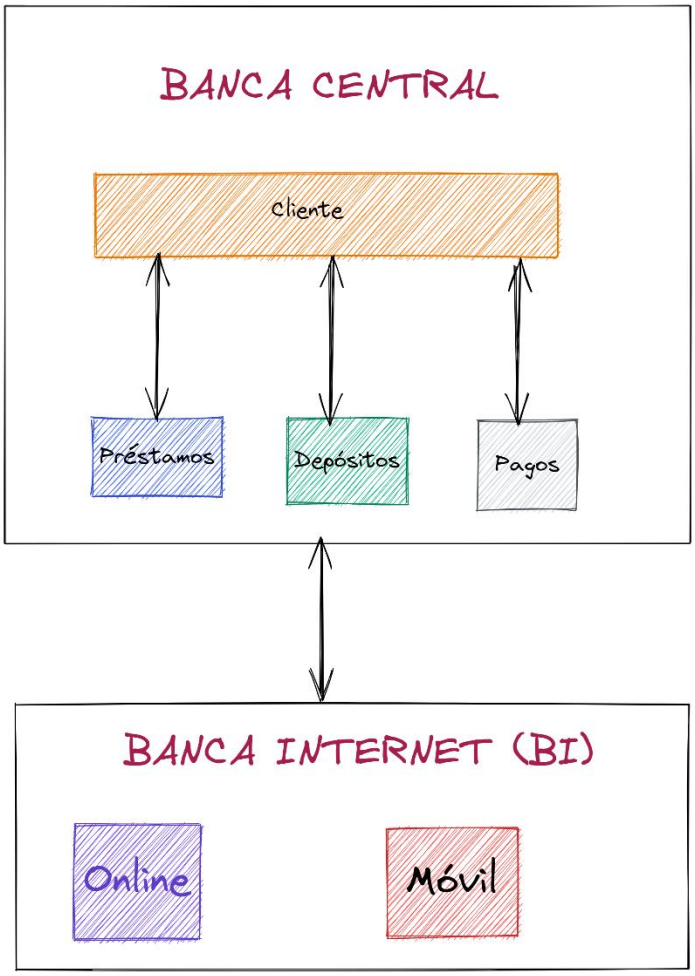

Figura 5 Sistema Bancario Tradicional con Canales Online, Móvil

En la actualidad las tablets y smartphone se convierten en una terminal móvil de venta para ayudar e incentivar el pago con tarjeta (tarjeta de débito, crédito) en pequeños comercios.

Para manejar estas transacciones online, se creó un canal llamado POS (Punto de venta, de sus siglas en ingles Point Of Sale), que engloba todas estas transacciones realizadas desde un posnet (artefacto que permite a un establecimiento comercial cobrar a sus clientes o usuarios mediante una tarjeta de débito o crédito) ver Figura **6**:

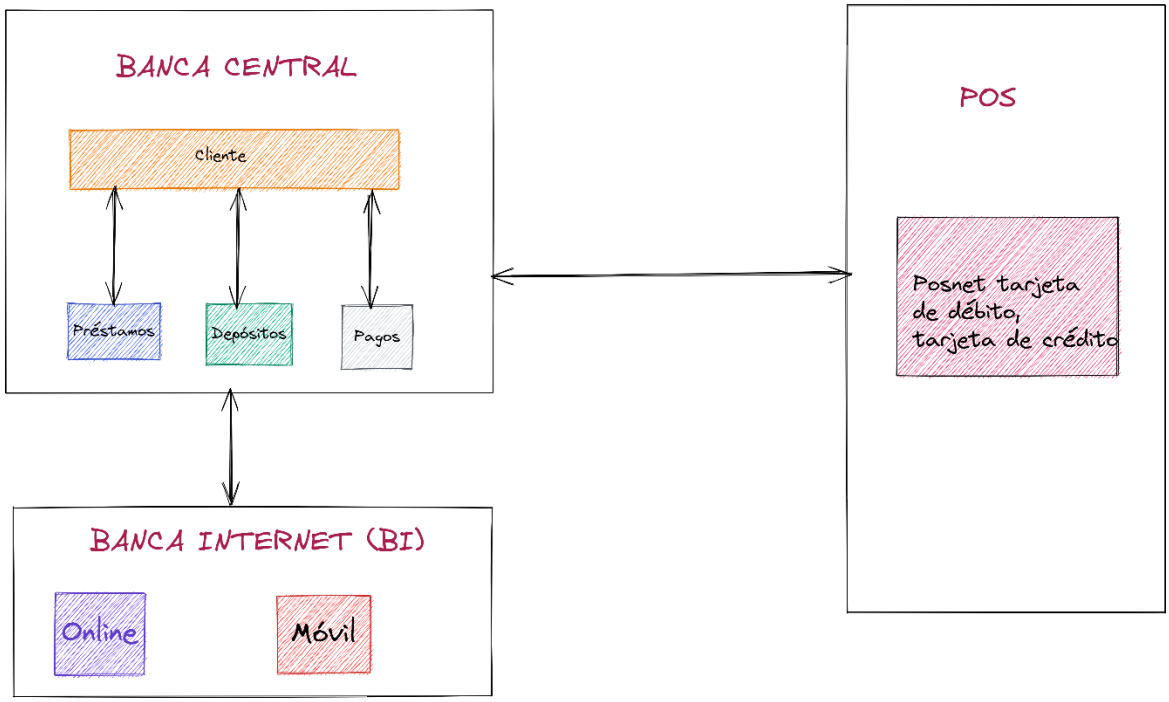

Figura 6 Sistema Bancario Tradicional con Canal Posnet.

Un Canal muy importante para los bancos es el ya mencionado ATM o Cajero. Las terminales ATM ya habían sido implementadas en la década del 90, pero en la actualidad los ATM suelen ser definidos en un canal aparte denominado ATM, como se muestra en la Figura **7**:

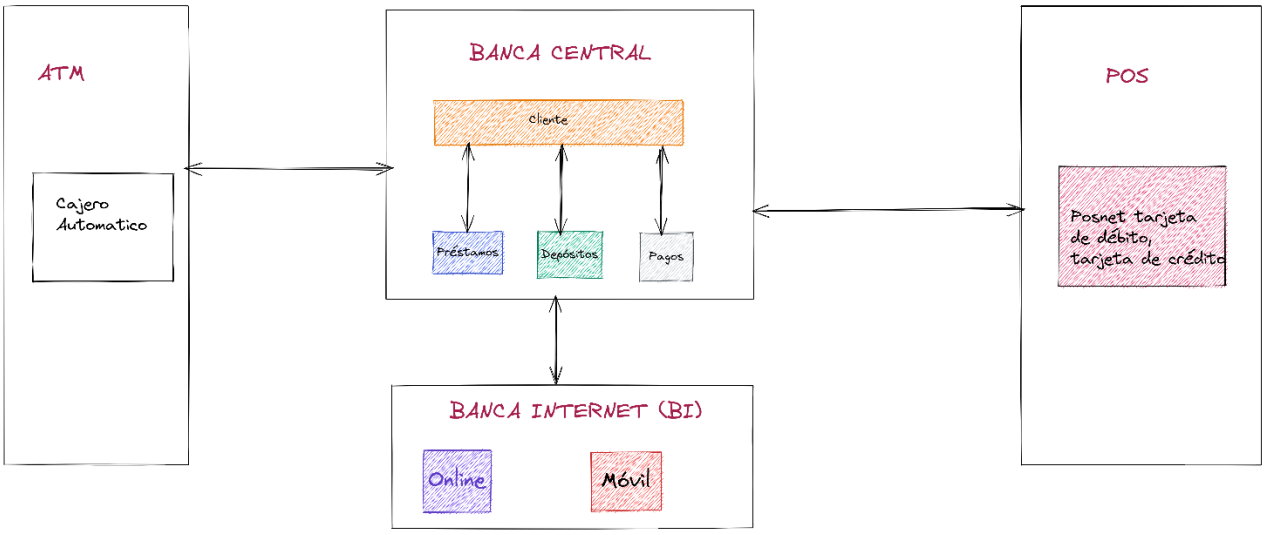

Figura 7 Sistema Bancario Tradicional con canal ATM

El sistema central podría estar integrado o no con el Sistema de Contabilidad (ver Figura **8**), en el cual las transacciones se contabilizan con el fin de presentar informes al directorio del banco, personal administrativo o hacia algún ente regulatorio. Ejemplo de estos informes pueden ser informe de las tarjetas del sistema, informe sobre el pago de los clientes, etc.

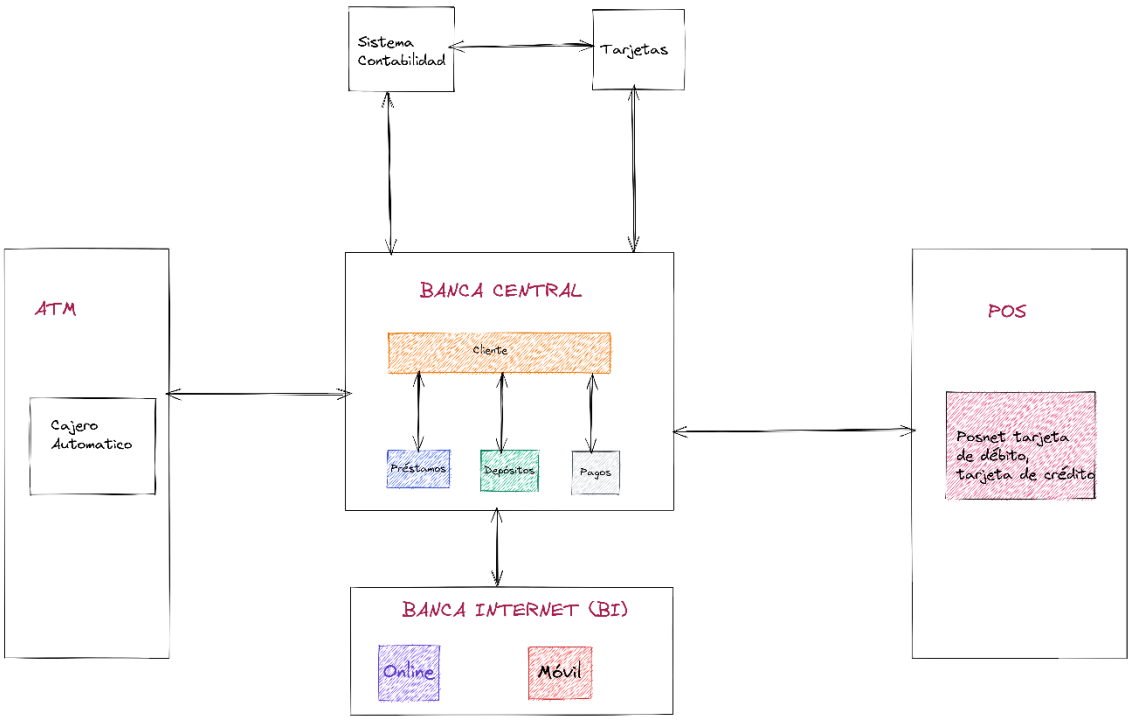

Figura 8 Sistema Bancario Tradicional con canales más comunes.

Lo importante en destacar es que estos todos los canales y sistemas internos/externos definidos utilizan los datos del sistema de Banca Central. Todos estos canales, Banca Central, Sistemas Internos conforman el Core Bancario.

Objetivo de un Core Bancario

Toda entidad bancaria tiene como objetivo poder satisfacer las necesidades de sus clientes, cumplir con las necesidades del negocio bancario. Para poder lograr estos objetivos, es importante apoyarse en un sistema que sea productivo, flexible, moderno, eficiente, rentable. Estos justamente son los objetivos de un Core. El Core puede dar soporte a todas las necesidades de un banco, todo ello con una arquitectura integrada y orientada a servicios que facilita la flexibilidad de los procesos del banco.

Características de un Core Bancario

Los Cores Bancarios se distinguen por ciertas características indispensables que en la actualidad deberían de cumplirse para poder cumplir con la necesidad del negocio. A continuación, se describen algunas de ellas:

- Confiabilidad: Un Core Bancario deber estar pensado para soportar volúmenes exigentes y de alta carga transaccional (millones de transacciones diarias). Hoy en día, con la Banca Online, Banca Móvil debemos pensar que el grado de crecimiento de ejecución de transacciones ha crecido a niveles exponenciales comparados con los años 90. Este es uno de los puntos por el cual se separa de la Banca tradicional y se modulariza en distintos canales que se encargan de procesar estas transacciones en manera independiente, almacenando datos en sus propias Base de Datos (HIDALGO, 2016).

- Disponibilidad: Los clientes de un banco deben tener la posibilidad de acceder a sus servicios financieros las 24 horas del día, los 7 días de la semana y los 365 días del año.
- Accesible: Los distintos productos (Prestamos, Créditos, Pagos, Tarjetas, Cuentas, etc.) y servicios (Creación de Prestamos, Realizar Pagos, Consultar Resumen de Tarjetas, solo por mencionar algunos) deben estar disponibles para usuarios y clientes a través de cualquier canal. Por ejemplo, un cliente podría consultar el Saldo de su cuenta por un ATM, BI o Móvil.
- Escalable: De ser capaz de crecer junto con el banco, es decir el banco puede que tenga una pequeña red de sucursales, o bien puede que sea un banco multinacional con una extensa red de sucursales, desde las cuales se consulta al sistema a través de múltiples canales. (HIDALGO, 2016).
- Seguridad: El Core debe permitir a los clientes y usuarios de las aplicaciones realizar transacciones seguras, las cuales incluyen cuentas, contraseñas bancarias, datos de tarjetas de crédito, información privada de los clientes, etc. No se puede permitir perder la confianza en los clientes ya que esto es lo que da reputación a los bancos.
- Arquitectura De Servicio: El Core debe poder permitir a los bancos realizar cambios de su sistema de acuerdo a sus necesidades de negocio, a medida que estos van cambiando o evolucionando. Este tipo de arquitectura permite realizar programas de migración en forma continua, lo cual trae beneficios para el banco (HIDALGO, 2016).
- En línea y tiempo real: En todo momento se debe disponer toda la información actualizada en línea y en tiempo real, de modo que pueda ser accedida desde cualquier lugar

Componentes de Core Bancario

Arquitectura

La arquitectura es la manera en que se va a estructurar el sistema y la que nos permitirá la evolución del mismo. Es decir, que la Arquitectura es el diseño de más alto nivel de la estructura de un sistema, ocultando los detalles de implementación. Es importante seleccionar una correcta arquitectura del Core Bancario, de modo que nos permita fácilmente reutilizar y crear nueva funcionalidad. Como consecuencia de esto último nos permitirá reducir costos y poder tener un mejor control de crecimiento de nuestro sistema. Además de aportar calidad en los distintos desarrollos.
Las aplicaciones bancarias actuales en su gran mayoría están basadas en una de las siguientes arquitecturas (HIDALGO, 2016):

- Cliente – Servidor.
- Modelo tres capas.
- Orientado a servicios (SOA)
- Microservicios (API REST)

Arquitectura Cliente – Servidor

Esta arquitectura está compuesta por Clientes, Servidores y Red, donde los Clientes consumen los servicios (ver Figura 9).
El funcionamiento de la arquitectura consta en que los clientes de la Red pueden conocer que servidores se encuentran disponibles, y de esta manera poder realizar peticiones Http al servidor

disponible, solicitando un servicio. En cambio, el Servidor no necesita conocer que cliente accedió a la información o cuantos clientes están accediendo a la información del servidor.
El servidor contesta las peticiones enviando uno o varias respuestas http.
La ventaja que tiene una Arquitectura cliente-servidor es que es una arquitectura distribuida, es decir que se puede tener sistemas en red y tener muchos procesadores distribuidos, del mismo modo que es fácil añadir nuevos servidores (Molina Ríos Jimmy, 2015).

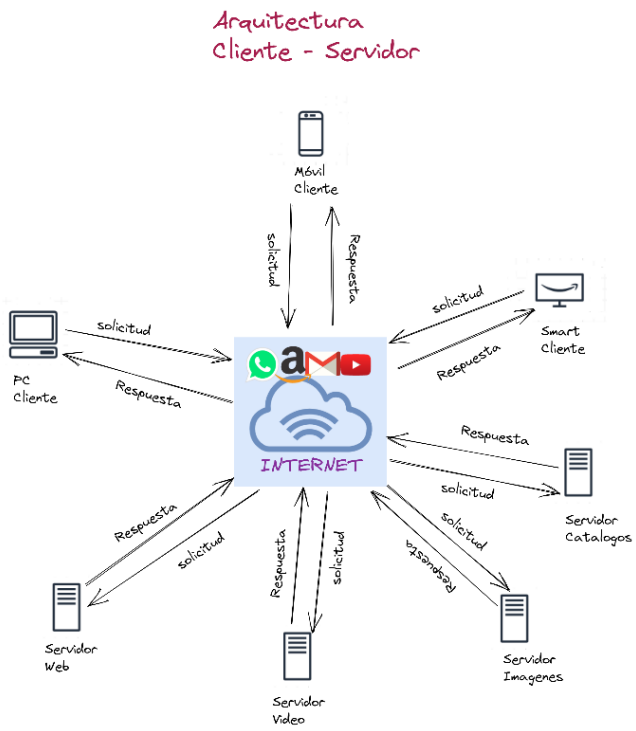

Figura 9 Arquitectura Cliente – Servidor

Modelo tres capas

Esta arquitectura separa o tiene bien definida 3 capas ver Figura **10**:

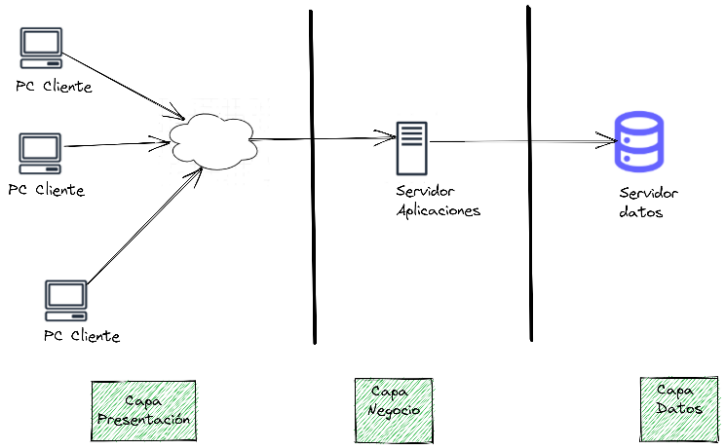

Figura 10 Arquitectura Modelo Tres Capas

- Capa de Presentación: Es la interfaz de usuario encargada de manejar la iteración entre un usuario y el Core Bancario. Al ser una capa que tiene iteración con el usuario, es deseable que esta sea intuitiva y amigable.
- Capa de Negocio: En esta capa se define toda la lógica del negocio a utilizar dentro del Core Bancario. Esta capa es una capa intermedia, es decir que se encuentra conectada tanto a la Capa de Presentación como a la Capa de Datos. Por ejemplo, dentro de esta capa podríamos validar los datos ingresados por el usuario en la Capa de Presentación. Otro ejemplo, podría ser calcular los intereses de un Préstamo, donde los datos de los Prestamos fueron obtenidos/leídos de la Capa de Datos.
- Capa De Datos: En esta capa se realiza la comunicación con base de datos donde se encuentran almacenados los datos, y puede estar conformada por uno o más Sistemas de Gestión de Base de Datos (SGBD) (HIDALGO, 2016).

Arquitectura Orientada a Servicios (SOA)

SOA es una forma de desarrollar sistemas distribuidos en la que los servicios se ejecutan en forma independiente en computadoras distribuidas geográficamente.

SOA permite descomponer el problema en un conjunto de servicios acoplados que se integran a través de un bus de servicio.

Esta arquitectura nos permitirá crear sistemas altamente escalables, que ayudaran a reducir costos y mejorar la flexibilidad en los procesos del negocio, de tal modo que los cambios pueden ser realizados a componentes individuales dentro del sistema sin tener que afectar al resto de los componentes. Esto último es importante, ya que es posible incorporar nuevas aplicaciones y servicios e integrarlas al negocio sin necesidad de realizar una reestructuración general (Chambó, 2015).

Podemos decir que una arquitectura SOA está compuesta por:

- Servicios: son aplicaciones que exponen funcionalidad, esta funcionalidad es diseñada y desarrollada por un proveedor quien además especifica la interfaz del servicio (Chambó, 2015).
- Proceso de Negocio: Es lo que construimos/desarrollamos para cumplir con nuestro negocio, consumiendo las aplicaciones expuestas en otros servicios (Chambó, 2015).
- Componentes de Infraestructura: Son los elementos que permiten la correcta comunicación entre componentes de la arquitectura, por ejemplo, ESB (de sus siglas en ingles Enterprise Service Bus). El ESB brinda la posibilidad de compartir el uso de los servicios entre aplicaciones, redirigiendo y procesando todas las invocaciones a servicios.
  Es decir que, si estamos desarrollando un componente, y necesitamos invocar un servicio construido en otro componente, lo haremos a través del ESB (ver Figura 11).
  Esto hace que cada componente sólo requiere desarrollar, probar y mantener la interfaz con el ESB, en lugar de integrarse de forma diferente con cada una de las aplicaciones que consumen. En síntesis, el ESB nos permite abstraer la comunicación entre componentes y servicios (Chambó, 2015).

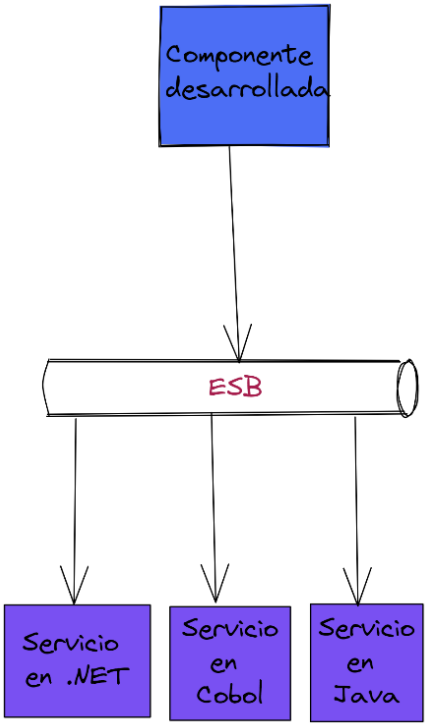

Figura 11 Componente ESB

Arquitectura Microservicios

Una arquitectura de Microservicios despliega aplicaciones independientes, autónomas y modulares, lo cual difiere de la forma tradicional o monolítico (en una aplicación monolítica toda la lógica se ejecuta en un único servidor de aplicaciones).

Podemos decir que es un enfoque para el desarrollo de una aplicación única que está conformada por un conjunto de servicios, cada uno ejecutándose en su propio proceso, sobre un protocolo de transferencia (HTTP) y una interfaz de programación (API) ver Figura **12**:

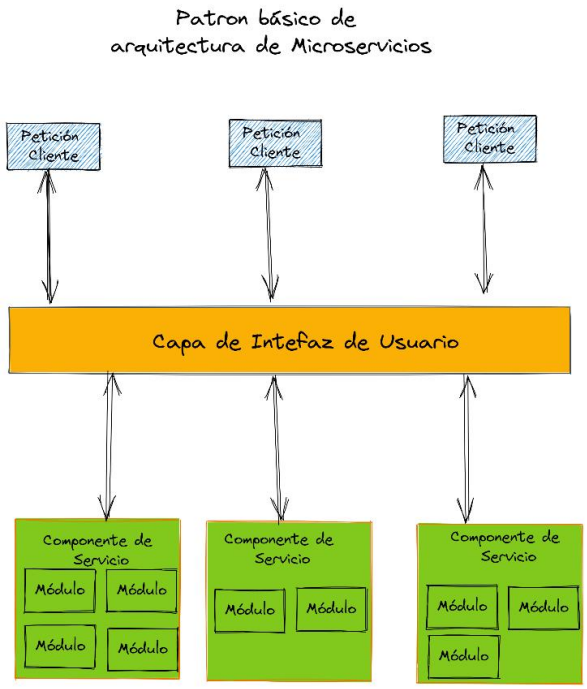

Figura 12 Arquitectura Microservicio - extraída de ***(Daniel López, 2017)***

Los microservicios permiten gestionar grandes aplicaciones, donde esas aplicaciones son descompuestas en un conjunto de aplicaciones más pequeñas conformadas por código y despliegue independiente. Esto es muy importante en el mantenimiento de grandes sistemas, donde solo deben ser modificados y desplegados aquellos servicios que modifiquen o agreguen funcionalidad. A diferencia de los servicios monolíticos, al realizar una modificación se deberá redesplegar toda la aplicación (Daniel López, 2017).

En los microservicios para que las aplicaciones desarrolladas se comuniquen con el resto del sistema es necesaria la implementación de servicios web, estos servicios web se comunican a través de la interfaz de comunicación REST (de sus siglas en ingles Representational State Transfer), que permite conectar varios sistemas basados en el protocolo HTTP además de obtener datos o generar operaciones sobre estos en los formatos como XML y JSON (formato más utilizado en la actualidad para la comunicación entre aplicaciones) (Vaicilla, 2021).

Hay que destacar que REST surge como una alternativa al protocolo estándar de intercambio SOAP (de sus siglas en ingles Simple Object Access Protocol), y que en la actualidad es la interfaz de intercambio más utilizada. A continuación, se presentan las principales características de REST.

Características de REST

- Cliente-Servidor: Definen una interfaz de comunicación entre el cliente y servidor, separando la responsabilidad entre ambas partes.
- Sin estado: No es necesario almacenar el estado de la sesión del usuario
- Accesos Similares: Todos los servicios REST se invocan de la misma manera, utilizando métodos similares del protocolo HTTP: GET (obtiene los datos), POST (Crear datos), PUT, DELETE (Elimina datos).
- Es independiente del lenguaje de programación (Vaicilla, 2021).

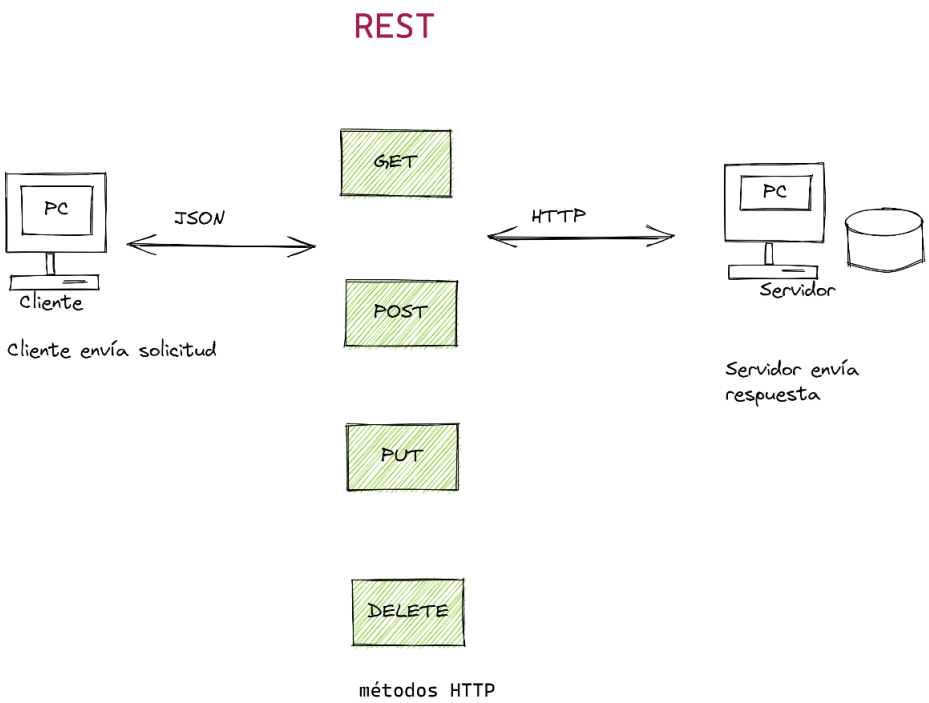

Figura 13 Arquitectura REST – extraída de ***(Vaicilla, 2021)***

Arquitectura del banco

Los bancos están sujetos a influencias externas que aumentan rápidamente (necesidades de los usuarios, negocio bancario, mercado, etc.), lo que a su vez implica que requieren una infraestructura tecnológica más flexible para poder brindar mejor los servicios ofrecidos a costos más bajos.

Los desafíos comunes que enfrentan los bancos en términos de sus infraestructuras tecnológicas radican en el hecho de que cada banco tiene un panorama tecnológico, que suele ser lento y costoso de adaptar y que por lo general representa una falta de interoperabilidad.
Para brindar una solución a eso, se busca estandarizar las infraestructuras tecnológicas bancarias sobre una base orientada al servicio. En general, se adopta una arquitectura orientada a servicios (SOA).
La lógica detrás de este enfoque es la separación topológica de la funcionalidad en servicios separados. A través de esa topología se proporciona un flujo de información mejorado y una mayor flexibilidad arquitectónica.
La funcionalidad completa se encuentra expuesta, por lo tanto, accesible para todos (desde otros servicios, o desde otros componentes externos).
Esto permite la reutilización de los servicios, y como ventaja el sistema se configura de manera más eficiente, reduciendo costos de mantenimiento y desarrollo de software.
Otro beneficio es la reducción de complejidad del software, ya que la composición funcional de los servicios no está anidada, sino que se sitúa en un nivel jerárquico general donde ningún servicio encapsula o controla a otro.
Por lo tanto, todo el sistema es menos complejo ya que los servicios se mantienen en un tamaño funcional moderado y, en consecuencia, son fáciles de mantener y ampliar a largo plazo.
La **Figura 14** muestra una descripción general de la arquitectura del banco y sus servicios.

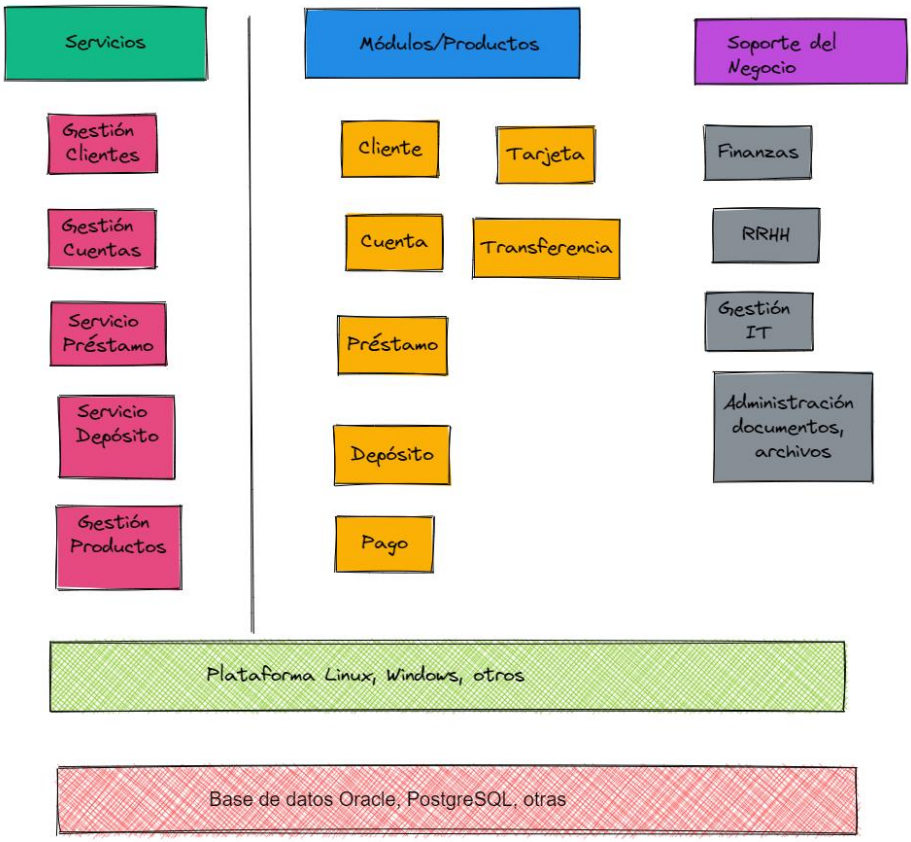

Figura 14 Arquitectura Banco

Módulos de un Core Bancario

Un banco tiene distintos módulos (ver Figura **15**) que son los encargados de poder llevar a cabo las distintas operaciones, mediante la utilización de aplicaciones core para la creación de distintos productos y servicios a clientes. En ocasiones módulos y aplicación core pueden ser difícil de diferenciar, ya que hacen referencia a la misma funcionalidad en la etapa del desarrollo, una en la fase de requerimientos y análisis (módulos) otra en la instancia de implementación (aplicación core).

Los principales módulos que componen un Core bancario son los siguientes:

- Clientes (Customer): Módulo que administra la información de los clientes físicos y jurídicos, permitiendo realizar distintas operaciones como por ejemplo alta de cliente, consulta de datos, modificaciones de sus datos, administrar acceso a las distintas transacciones, etc.
- Cuentas (Account): Módulo que administra las cuentas corrientes, cuentas cajas de ahorro entre otras. Contiene información referido a los saldos, movimientos, titulares de las cuentas, monedas, etc. Entre las operaciones que se pueden realizar se encuentran el alta de cuentas, modificaciones de cuentas, consulta de saldos, etc.
- Prestamos (Loan): Módulo que administra la información referida a las distintas líneas de préstamos que se trabajan en el banco. Contiene información sobre las cuotas, interés a cobrar, montos de las cuotas, mora, plazos, cronogramas de pagos, pago del préstamo, etc.
- Depósitos (Deposit): Módulo que administra la información de los distintos tipos de Depósitos que contiene el banco. Maneja información sobre interés, plazos, cronogramas, montos a pagar al cliente, etc.
- Cajas (Teller): Módulo que permite manejar la operatoria de una caja de un banco, es decir que registra las transacciones realizadas por clientes sobre los distintos productos bancarios. Permite realizar aperturas y cierres de cajas de sucursales, movimientos de dinero a las bóvedas, arqueos de caja, entre otras operaciones.
- Transferencias (Funds Transfer): Módulo que permite realizar transacciones monetarias entre cuentas del mismo banco o entre cuentas de distintos bancos.
- Banca Virtual: Módulo que permite a los clientes permite consumir los distintos productos de banco a través de Internet. Por ejemplo, realizar transferencias, pagar un préstamo, crear un Depósito, consultar el Saldo de una cuenta, etc.
- Móvil: Modulo que permite el acceso a los servicios bancarios a través de los dispositivos móviles las 24 hs los 365 días del año.
- Pagos: Módulo que administra permitiendo registrar y consultar los pagos realizados por los clientes.

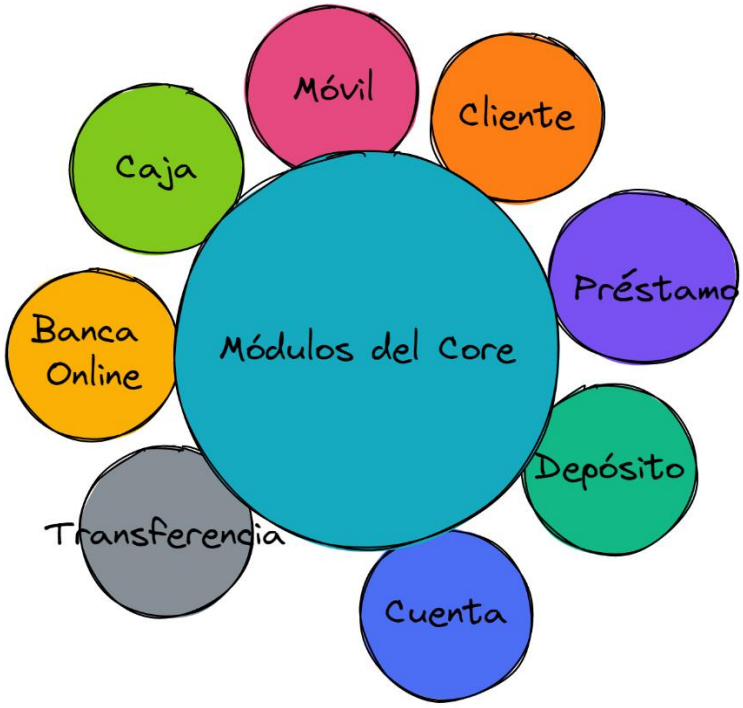

Figura 15 Módulos Core Bancario

Ventajas por el cual migrar a un sistema de aplicaciones core

Un Core se encuentra compuesto por distintos módulos, para los cuales se definen un conjunto de operaciones nativas que pueden ser llevadas a cabo dentro de un banco. En otras palabras, hay operaciones que son comunes a todos los bancos, con lo cual estas operaciones ya se encuentran definidas dentro de un módulo en los sistemas de aplicaciones core. Un ejemplo de esto es la que aplica la definición del módulo de cuenta, donde podemos encontrar la funcionalidad básica que pueden ser realizadas, como por ejemplo alta de cuenta, modificar un dato particular de la cuenta, consultar su saldo, etc. De la misma manera ocurre para cada uno de los distintos módulos (Trasferencia, Caja, Deposito, Préstamo, Cliente, etc.) que componen a los sistemas de aplicaciones core.
La ventaja de poder usar un sistema de aplicaciones core radica en que sobre las operaciones nativas los bancos pueden definir sus productos (distintos tipos de cuentas, diversos tipos de préstamos y depósitos, clasificar sus clientes de alguna forma particular, etc.) mediante la customización y agregado de comportamiento a través de la programación de los módulos que provee el Core.

De esta manera se pueden definir grandes productos con mucha lógica de negocio en tiempos cortos, ya que la operación básica o nativa de cada módulo ya se encuentra implementada en el sistema de aplicaciones core.

En cuanto a un sistema implementado con tecnología Mainframe para definir este tipo de productos, primero deben ser definidos/creados en una vista (un HTML para poder ingresar la

información del producto que desea crear), esa información es ingresada a los sistemas Mainframe, los cuales agregaban toda la lógica de negocio mediante el lenguaje Cobol. En síntesis, el Core desarrollado en Mainframe no tiene modelos de productos con las operaciones básicas mencionadas anteriormente, lo cual deben ser generados por HTML o algún lenguaje del front end, y además se deberá agregar la lógica del producto en el Mainframe mediante Cobol.

Las empresas que proveen solución de sistemas en aplicaciones core, son las que definen los módulos o funcionalidad nativa. A continuación, se detallan otras de las ventajas que podemos tener con el uso de aplicaciones core:

- Uno de los principales motivos es la sencillez, seguridad y eficacia que ofrece para realizar informes, cálculos y utilizar diferentes hojas de codificación para la descripción de entidades.
- Tiene una muy amplia cobertura funcional y un alto grado de flexibilidad para responder a nuevos requerimientos.
- Alto nivel de parametrización para crear nuevos productos (prestamos, depósitos, cuentas, etc.) o modificar los existente.
- Tiene una arquitectura flexible alineada con los requerimientos del banco, basada en estándares abiertos, permite alta disponibilidad y escalabilidad horizontal
- La solución es independiente de plataforma de hardware y sistema operativo, lo cual genera fortalezas al banco, al tener libertad de elección.
- El requerimiento de equipamiento en las sucursales es mínimo (browser).

Enterprise Cores Bancarios

En la actualidad podemos encontrar una diversidad de empresas dedicadas a las soluciones sobre sistemas en aplicaciones core, entre las cuales se tomaron aquellas con mayor experiencia en cuanto a los años que se encuentran en el mercado, así como también por el tamaño de sus carteras de clientes bancarios que poseen. A continuación, se realiza una muy breve descripción de las funcionalidades que ofrecen sus soluciones.

Core Temenos, desarrollado por una compañía de origen suizo

Las soluciones incluyen las siguientes funcionalidades:

- Sector corporativo: préstamos comerciales, sobregiros, transacciones comerciales, cobro de deudas, cuentas.
- Negociación de valores - mercado monetario, divisas, valores, futuros y opciones (derivados financieros).
- Banca minorista y corporativa - Cajas de ahorro, cuentas corrientes, cuentas de ahorro y depósito, sobregiros, cheques, tarjetas de crédito, préstamos hipotecarios, préstamos rotativos, fondos de inversión.
- Banca Privada - Valores y gestión de cartera de clientes, presentación de negocios.
- Operaciones Básicas - CRM del inglés Customer Relationship Management, riesgo de mercado, riesgo de crédito, contabilidad, gestión, rentabilidad, pagos, flujo de negocios.
- Canales alternativos - Internet, call center, banca online, otros canales electrónicos.
- Conexión con ambientes externos.

Además de lo anterior, también ofrece aplicaciones de cajero, transferencia de fondos, cuentas de ahorro, gestión de cheques y tarjetas de crédito, así como procesamiento de servidores de cajeros automáticos (Kreća, 2015).

Core Pexim, desarrollado por una compañía de origen macedonio

Es una aplicación bancaria integrada diseñada para trabajar con clientes corporativos y minoristas. Sus módulos son:

- Clientes.
- Cuentas.
- Pagos.
- Depositos.
- Préstamos.
- Tarjeta Crédito.
- Aplicaciones de apoyo para la gestión.
- Acciones.

Las soluciones brindan gestión de tarjetas de crédito nacionales y extranjeras (MasterCard, VISA, DINA, etc.), intercambio de datos en línea y fuera de línea, gestión de cajeros automáticos y terminales de pago POS, Internet y banca móvil (Kreća, 2015).

Core Antegra, desarrollado por una compañía de origen canadiense

Las funcionalidades claves del Core son:

- Bancas minoristas (cajas de ahorro, cuentas corrientes, préstamos, depósitos, pagos, etc.).
- Banca corporativa (préstamos, depósitos, clientes, cuentas, pagos, manejo de cheques).
- Reportes.
- Libro de accionistas.
- Conexión con ambientes externos.
- Banca online.

Este software permite la presentación de documentos en papel en forma electrónica (firma e imágenes), así como la capacidad de optimizar la entrada de datos operativos de acuerdo con los requisitos del banco, y también es fácil de usar en términos de configuraciones que son comprensibles para el usuario (Kreća, 2015).

Core Asseco, desarrollado por una compañía de origen polaco

La arquitectura de la aplicación se basa en un modelo estructural y modular con una plataforma cliente-servidor en capas con módulos independientes que permiten una modificación sencilla de los procesos de negocio (Kreća, 2015).

Los principales módulos de este sistema son:

- Información sobre el cliente (puede usarse para marketing directo o análisis de clientes).
- Servicios de cuentas (administración, débito, cheques, cuentas bancarias, pago de facturas, registro de actividades).
- Servicios de pago (capacidad de pago en moneda local y extranjera).
- Depósitos.
- Préstamos.
- Banca minorista (efectivo, cheques, transferencias de fondos, préstamos, nivel de ahorro de los servicios, cambio de moneda, firma electrónica, etc.).
- Tarjetas.
- Soporte de gestión de aplicaciones.
- Conexión con ambientes externos.

Comparación de Cores

Sobre las funcionalidades de las soluciones ofrecidas por las empresas elegidas, se realiza una comparación de dichas funcionalidades (ver Tabla 1) en donde los bancos analizarán de acuerdo a sus necesidades de negocio cual es la mejor opción. Además de los módulos que brindan las distintas empresas, los programadores al momento de realizar una customización sobre un módulo o funcionalidad, en ciertas ocasiones se requiere agregar programación extra (que no se encuentra en el comportamiento nativo del módulo o funcionalidad). Para esto se debe apoyar en los lenguajes de programación y sistemas operativos permitidos (ver Tabla **2**) por cada aplicación core.

Por funcionalidades o módulos

	Core Pexim	Core Antegra	Core Asseco	Core Temenos
Banca minorista	X	X	X	X
Banca corporativa	X	X	X	X
Cuentas	X	-	X	X
Pagos	X	X	X	X
Depósitos	X	X	X	X
Préstamos	X	X	X	X
Tarjetas	X	X	-	X
Contabilidad	X	-	X	X
Acciones	X	X	-	-
Reportes	-	X	X	X
Banca online	-	X	-	X
Conexión con ambientes externos	-	X	X	X

Tabla 1 comparación módulos **c**ore

Por SO y lenguajes

	Core Pexim	Core Antegra	Core Asseco	Core Temenos
JAVA	-	X	X	X
C	-	X	X	X
.NET	X	-	-	-
Linux	X	X	X	X
Windows	X	X	X	X

Tabla 2 Comparación por SO y lenguajes

Capítulo 4 – Reingeniería de sistemas

Introducción

En este capítulo se describe un contexto teórico acerca de la actualización del software. Luego se enumeran y describen los tipos de rejuvenecimiento del software. Además, se realiza una explicación de los beneficios de elegir reingeniería en el banco. Al final del capítulo se explican los pasos de la reingeniería utilizado en el banco para migrar de un sistema Mainframe Cobol hacia un sistema de aplicaciones core.

Contexto

La actualización del software aborda el desafío de mantenimiento al intentar aumentar la calidad de un sistema existente.

En la actualización del software se debe mirar hacia atrás en los componentes que contiene el sistema para tratar de obtener información adicional y poder actualizarlos de una manera más comprensible de forma tal de poder obtener mejores resultados.

Hay varios aspectos de la actualización del software a considerar, incluyendo:

- Redocumentación
- Reestructuración
- Ingeniería inversa
- Reingeniería

Cuando redocumentamos un sistema, se realiza un análisis del código fuente para producir información adicional, de esta forma se ayuda a comprender el código fuente a las personas que realizan el mantenimiento del sistema.

En la redocumentación, el análisis no transforma el código fuente real, sino que simplemente se deriva la información a las personas encargadas de modificar el código.

En cambio, en la reestructuración si se cambia el código fuente transformando el código fuente mal estructurado en código fuente bien estructurado.

Es importante destacar que la técnica de redocumentación y reestructuración se enfocan en el código fuente.

Para aplicar ingeniería inversa a un sistema, miramos hacia atrás desde el código fuente a los productos que se encontraban definidos, creando información de diseños y especificaciones del código fuente.

El concepto de reingeniería es aún más amplio, en donde aplicamos ingeniería inversa a un sistema y luego “Ingeniería Avanzada (progresión hacia adelante a través del proceso)” para hacer cambios en la especificación y el diseño que completan el modelo lógico, luego se genera un nuevo sistema a partir de la especificación revisada y el diseño (Pfleeger, 2010).

La Figura **16** muestra las distintas relaciones de estas cuatro técnicas de actualización de sistemas:

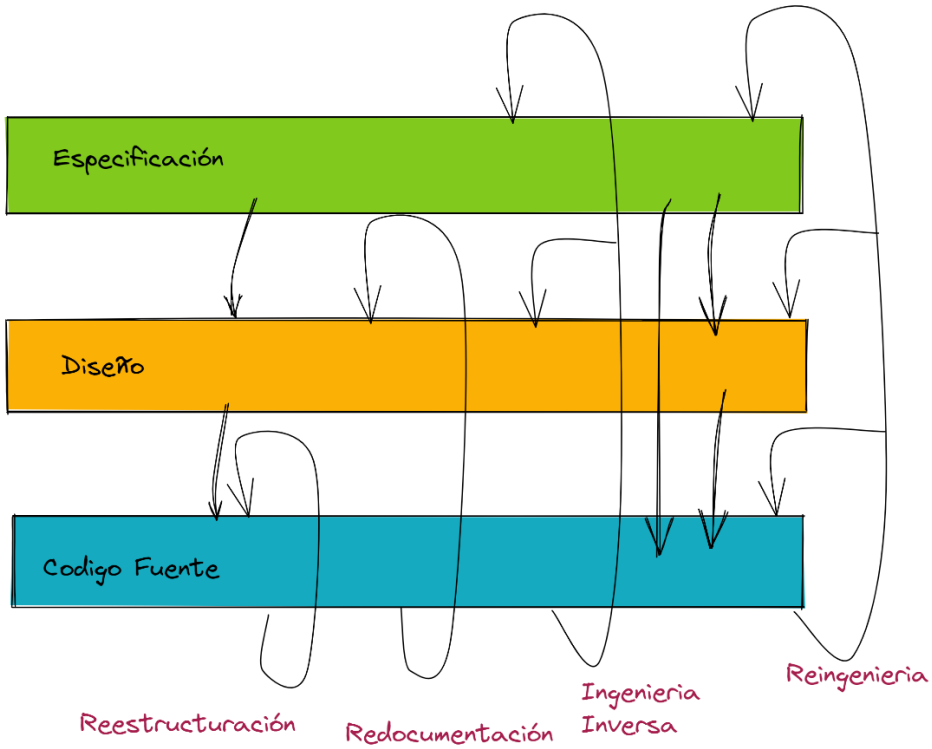

Figura 16 Actualización Del Software – extraída de ***(Pfleeger, 2010)***

Reestructuración de código, estructuras. Solo se enfoca en código fuente.

Redocumentación de código, análisis sobre las estructuras, complejidad, volumen, datos. Se enfoca en código y diseño.

Ingeniería Inversa de código, produce diseño y especificación. Se enfoca en código, diseño y especificación.

Reingeniería de código (regenera código), modifica representación de las estructuras y datos. Aplica ingeniería inversa e ingeniería avanzada. Se enfoca en código, diseño y especificación.

A las cuatro técnicas mencionadas anteriormente se las conoce como tipos de rejuvenecimiento del software.

Tipos de rejuvenecimiento del software

En la actualización del software se deben tomar decisiones difíciles sobre cómo hacer que los sistemas sean lo más mantenible posible. Las opciones pueden variar desde la mejora hasta el reemplazo completo con una nueva tecnología. Cada elección está destinada a preservar o aumentar la calidad del software, manteniendo los costos lo más bajo posible. A continuación, se detallan los cuatro tipos de actualización del software.

Reestructuración

Reestructuramos el software para que sea más fácil de entender y modificar.

Primero se realiza un análisis que proporcione información para poder representar el código fuente. Luego se refina esa representación mediante simplificaciones basadas en técnicas de transformación (reglas de transformación para simplificar la representación interna y los resultados se reformulan como código estructurado). La Figura **17** muestra los pasos sobre la reestructuración.

Finalmente, la representación refinada es la que es utilizada en la codificación del nuevo software (Pfleeger, 2010).

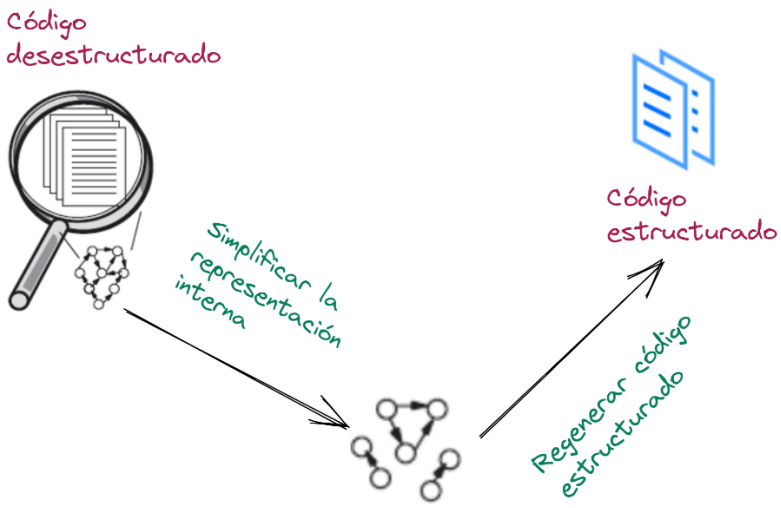

Figura 17 Actualización – Reestructuración – extraída de ***(Pfleeger, 2010)***

Redocumentación

Implica en análisis del código fuente para producir documentación del sistema (ver Figura **18**). La redocumentación incluye analizar el uso de variables, parámetros de las rutinas, como son invocadas las rutinas, tamaño del código fuente, y alguna otra media que pueda ayudarnos a comprender que hace el código fuente y como lo hace. La información que se obtiene en la redocumentación puede ser plasmada en forma textual o en forma gráfica, y dicha información puede incluir (Pfleeger, 2010):

- Diagramas de Flujo
- Llamadas relacionadas en distintos componentes
- Pseudocódigo
- Test paths
- Referencias cruzadas de componentes y variables

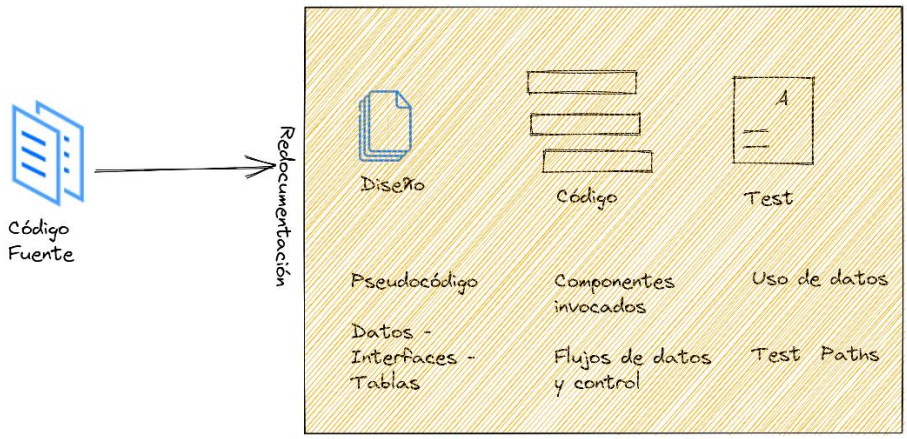

Figura 18 Actualización – Redocumentación – extraída de ***(Pfleeger, 2010)***

Ingeniería inversa

La ingeniería inversa proporciona información de diseño y especificaciones sobre el sistema a partir del código fuente. La información extraída no es necesariamente completa, porque muchos componentes originales suelen estar asociados con uno o más componentes de diseño, por esta razón nuestro sistema obtenido por ingeniería inversa tiene menos información que el sistema original.

El código fuente se envía a una herramienta de ingeniería inversa, que interpreta la estructura y la información de nomenclatura y construye resultados de la misma manera que la redocumentación. Los pasos mencionados se pueden ver en la Figura **19**.

La clave de la ingeniería inversa es su capacidad para abstraer las especificaciones de la implementación detallada del código fuente (Pfleeger, 2010).

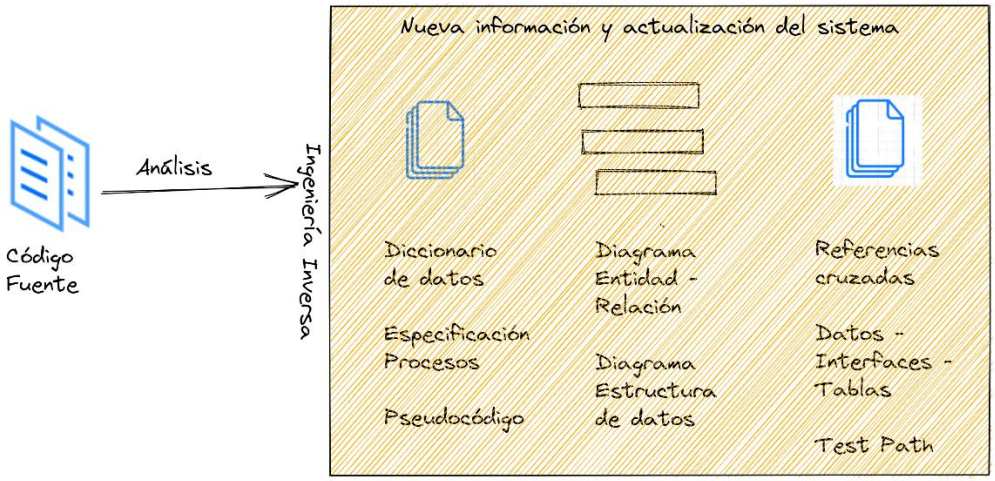

Figura 19 Actualización – Ingeniería Inversa – extraída de ***(Pfleeger, 2010)***

Reingeniería

La reingeniería es una extensión de la ingeniería inversa, mientras que la ingeniería inversa abstrae información, la reingeniería produce un nuevo código fuente de software sin la función general del sistema.

Como primer paso se aplica una ingeniería inversa y se representa internamente para realizar modificaciones logrando diseñar el software. Luego se corrigen completando el modelo del sistema. Como resultado de esto se obtiene una nueva especificación o diseño.

Las entradas al proceso de reingeniería incluyen archivos de código fuente, archivos de bases de datos, archivos de generación de pantallas y archivos similares relacionados con el sistema. Cuando se completa el proceso, genera toda la documentación del sistema, incluidas las especificaciones y el diseño, y el nuevo código fuente ver Figura **20** (Pfleeger, 2010).

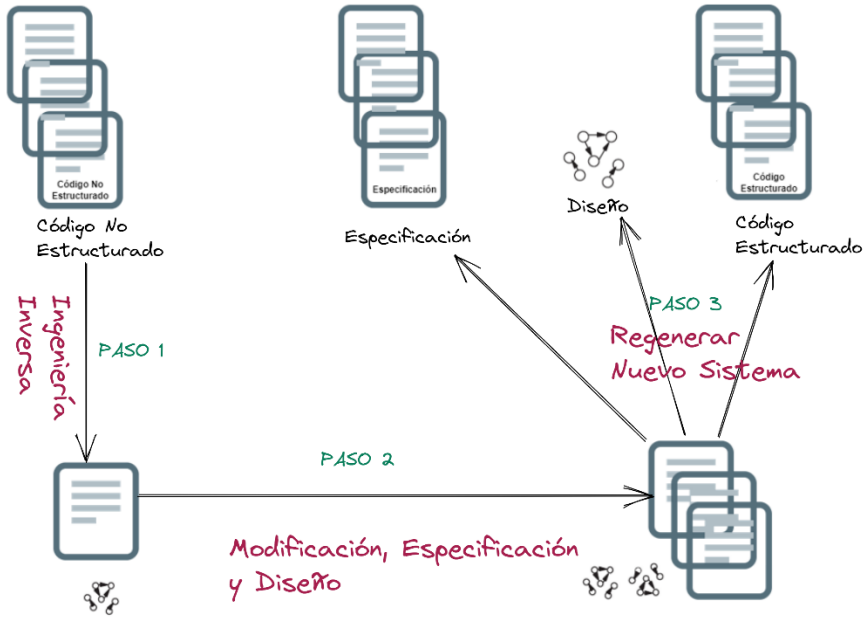

Figura 20 Actualización – Reingeniería – extraída de ***(Pfleeger, 2010)***

Reingeniería bancaria

Los sistemas heredados más antiguos son los que llevan más trabajo para entender y modificar, con lo cual son los sistemas más complejos de mantener.

Para que los sistemas heredados sean más fáciles de mantener se puede aplicar una reingeniería, de esta forma se logra mejorar su entendimiento.

Esta reingeniería puede ser implementada volviéndose a documentar el sistema, refactorizándose su arquitectura, actualizándose a un lenguaje de programación moderno, o modificando la estructura y los valores de los datos del sistema (Sommerville, 2009).

Beneficios de la reingeniería respecto de la sustitución completa del sistema

1) Cuando desarrollamos software critico el riesgo en cuanto a demoras es alto, ya que pueden cometerse errores en la especificación del sistema, así como también se pueden tener diversos problemas durante el desarrollo del mismo.
   Estas demoras pueden ocasionar perdidas o costos adicionales a los bancos.

2) El costo de aplicar reingeniería a un sistema puede ser menor que el costo de desarrollar el mismo sistema completamente nuevo con lenguaje de programación moderno. Este es el motivo por el cual los bancos suelen elegir optar por una reingeniería.

   La mayoría de los bancos, tienen un tiempo y presupuesto para la realizar nuevos desarrollos con modernos lenguajes. Esto requiere un análisis profundo del sistema heredado (en este caso un sistema en Cobol Mainframe) y luego decidir cuál es la mejor estrategia para llevar a cabo la evolución del sistema. Para esto último, hay cuatro escenarios posibles:

   a. Cambiar por completo el sistema. Por lo general este escenario se da cuando ya no se apoya más en el sistema heredado. Es decir, el negocio ha cambiado de forma tal que ya no se realiza una aportación efectiva a los procesos del negocio.
      Los bancos tienen sistemas desarrollados en décadas pasadas, por lo que siempre se apoyaron en ellos y difícilmente opten por una opción de cambiar el sistema completamente.

   b. Realizar operaciones de mantenimiento. El sistema es estable y solo hay pocas necesidades por parte de los usuarios del sistema.

   c. Aplicar reingeniería. Este escenario se debe a cuando ha bajado la calidad del sistema por los cambios continuos y aún restan nuevos cambios por realizar. La reingeniería puede incluir desarrollos de nuevos componentes de interfaz, de modo que el sistema original logré trabajar con otros sistemas más recientes.
      La reingeniería con el avance tecnológico y la necesidad de consultar mediante distintos dispositivos y por diversos canales (móvil, ATM, Posnet, BI, etc.,), hacen que el viejo sistema del banco construidos en épocas en que Internet y los canales que hoy en día consumimos eran nulos o prácticamente no existían, permita incluir nuevos desarrollos al viejo sistema en busca de un nuevo sistema moderno que pueda satisfacer las necesidades del mundo actual.
      Esta técnica de incluir nuevos desarrollos al viejo sistema en busca de un nuevo sistema es la que mejor encaja en muchos de los sistemas Bancarios.

   d. Sustituir todo o parte de un sistema por un nuevo sistema. El viejo sistema no puede continuar por ejemplo porque se actualizo el hardware. En estos casos se puede optar por sustituir el sistema en forma evolutiva, sustituyendo los grandes componentes del sistema y reutilizando aquellos componentes que sea posible.

Pasos del proceso reingeniería

En la migración del sistema Cobol Mainframe hacia un sistema de aplicaciones core, el banco aplica un proceso de reingeniería. En este proceso se involucran varios equipos del banco (equipo de analistas funcionales, equipo de desarrollo, equipo de testing, equipo de producción), con el objetivo de lograr un sistema aplicaciones core estable. Es sumamente importantes que todos los equipos tengan muy en claro el objetivo, ya que el proceso de reingeniería de un sistema bancario es complejo en cuanto al negocio que lo rodea, tiempos largos por la criticidad de muchos componentes del sistema Cobol Mainframe, costos de dinero por diversas cuestiones, entre otras cosas. En la Figura **21** se muestran los pasos de reingeniería aplicados en un componente que se encuentra en Cobol Mainframe y debe ser migrado hacia un componente en un sistema de aplicaciones core.

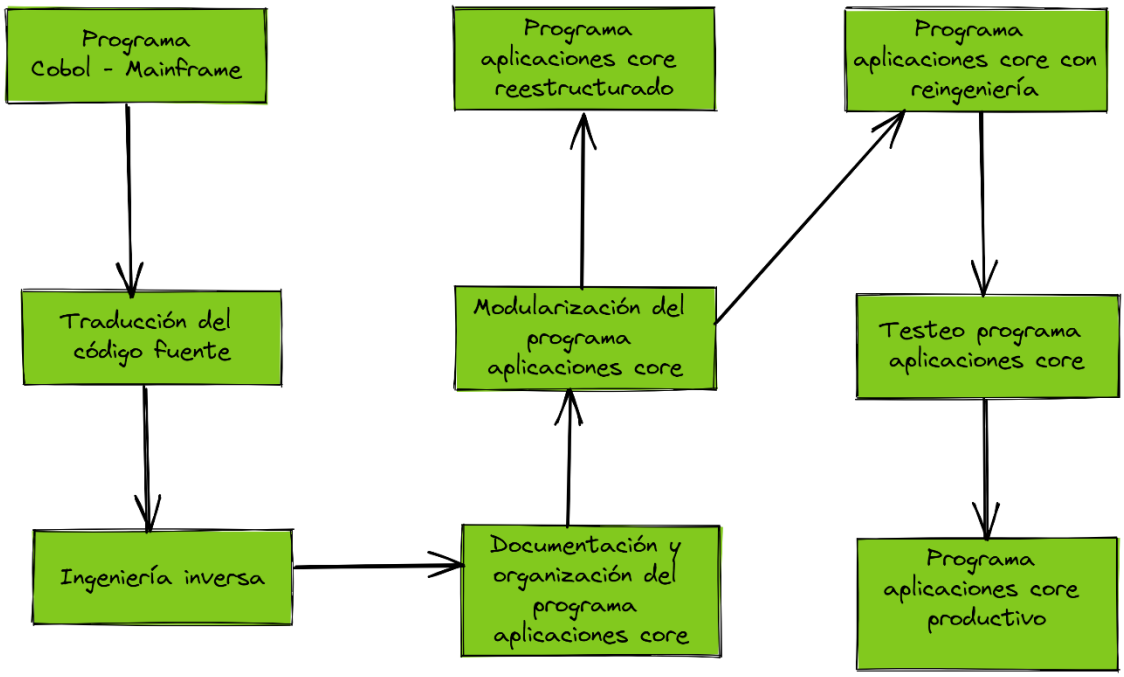

Figura 21 Proceso de reingeniería

A continuación, se pasan a detallar los pasos de reingeniería utilizado por el banco:

1. Traducción del código fuente. Los programas del antiguo sistema se encuentran en lenguaje Cobol Mainframe, y tanto la nueva funcionalidad como las modificaciones de funcionalidad existente son realizadas en aplicaciones core.

El analista Funcional que se encuentra en el banco, de acuerdo al conocimiento que tiene del antiguo sistema, analiza las distintas necesidades de agregar funcionalidad que no existía en el sistema anterior, o bien puede solicitar modificar funcionalidad existente que debe ser adaptada a las nuevas necesidades del banco.

2. Ingeniería Inversa. Cada programa en Cobol Mainframe es analizado y se extrae información de él con el fin de documentar y organizar la funcionalidad. Esta tarea es realizada por los analistas funcionales del banco. Los documentos funcionales contienen el mínimo funcionamiento del módulo (Account, Deposit, Loan, Customer, etc.) en el cual se desea agregar o modificar la funcionalidad, junto con una pequeña explicación de lo deseado. Los Test Case son adjuntados por el Analista Funcional en la especificación funcional.

3. Modularización del programa. Los analistas funcionales validan la redundancia en el código (lecturas, escrituras, código repetido, etc.) de los programas Cobol Mainframe, con la idea de eliminar esa redundancia en el código que será migrado a componentes del sistema de aplicaciones core.

4. El componente es desarrollado por el programador en el ambiente de desarrollo. El programador analiza las especificaciones funcionales, comenta dudas y desarrolla los solicitado por el analista funcional. Cuando el componente es finalizado (codificación y pruebas básicas) por parte del programador, se migra todo el componente hacia el ambiente de Testing, quien será el encargado de realizar las pruebas unitarias e integrales.
   De las pruebas realizadas por Testing depende cuales son los desarrollos que subirán al ambiente productivo del banco.

5. Se sube el componente al ambiente productivo del banco, el cual es consumido por los clientes y usuarios del sistema.

Capítulo 5 – Aplicación del proceso de reingeniería

Introducción

El banco fue creado en el año 1979 con el objeto de prestar servicios financieros a todos sus asociados. Con 482 filiales en todo el país, es el primer banco privado con capital 100% nacional. Es el octavo banco en Argentina considerando el volumen de socios, sexto entre los bancos privados. Se encuentra asociado al ABAPRA (Asociación de Bancos Públicos y Privados de la República Argentina). Su foco de servicio se encuentra en pequeñas y medianas empresas, cooperativas, empresas de economía social, individuos. En la Tabla **3** se muestran los volúmenes estimativos manejados por el banco.

	Volumen Estimativo
Filiales	482
Empleados	7815
Cajeros automáticos	984
Terminales autoservicios	750
Socios Activos(clientes)	6.579.364
Caja de ahorro activas	5.990.000
Caja de cuenta corrientes activas	4.835.000
Tarjetas de debito	5.397.393
Tarjetas de crédito	3.091.497
Empresas con banca internet	1.750.000
Individuos con banca internet	2.250.000

Tabla 3 Volumen manejado en el Banco

IT - Bancario

En la actualidad para poder afrontar los retos de la transformación digital, los bancos se encuentran alineados con áreas de IT de sus siglas en Ingles Information Technology, para permitir un servicio diferenciado y optimizado como objetivo primordial. Para esto último podemos decir que IT se compone de infraestructura, metodologías, ambientes y herramientas, las cuales se pasan a detallar.

De los Core mencionados en el capítulo 3, el banco relevado, selecciono uno de los *Enterprise* que se lo denominara Core-S (Core Seleccionado).

Infraestructura

El banco se apoya en servicios locales y servicios en la nube. El porcentaje de los servicios que aún se encuentran en forma local es menor, ya que originalmente todos los servicios del banco se encontraban en servidores locales, pero con el tiempo se ha ido migrando hacia nuevas tecnologías, de modo tal que ya se ha logrado migrar más del 60% de la infraestructura tecnológica hacia la nube, permitiendo la posibilidad de proporcionar las herramientas suficientes para facilitar el teletrabajo a los equipos del banco, desplegar aplicaciones críticas de negocio en la nube, mejorar la experiencia del cliente gracias al uso de la última tecnología que permite ofrecer servicios y aplicaciones innovadores con foco en el cliente y con mejores tiempos de respuesta. Además, de tener innumerables beneficios tales como la reducción del riesgo de operaciones, elimina obsolescencia de hardware sin incrementar los costes y aporta flexibilidad para poder llevar las cargas de trabajo a la nube pública o privada en función de diversos factores como la seguridad o el rendimiento. Se estima que en el corto plazo el banco migrara todos sus servicios a la nube, convirtiéndose en un banco totalmente digitalizado.

Metodologías

La integración continua es una práctica de desarrollo de software en la que los desarrolladores de un equipo envían con frecuencia cambios de código a un repositorio; por lo general, cada desarrollador envía código al repositorio al menos una vez al día, lo que genera múltiples integraciones por día.

Cada integración se verifica mediante una compilación automatizada (incluida la prueba) para detectar errores de integración.

Muchos equipos encuentran que este enfoque conduce a problemas de integración significativamente reducidos y permite que un equipo desarrolle software cohesivo más rápidamente (Fowler, 2006).

En nuestro desarrollo, la integración continua se realiza mediante herramientas como Bitbucket, definido como el repositorio de código del banco, el cual se apoya en herramientas como GIT para el versionado, y en Bamboo (herramienta de integración continua), que es quien realiza el build mediante la compilación del código y la corrida de los test definidos, generando como resultado una nueva release (es un número de versión). Para que esa release se encuentre disponible para ser deployada en algunos de los ambientes, previamente se debe validar y aprobar el pull request. Luego, el deploy continuo se realiza por medio de la herramienta Harness, quien toma la release generada y la deploya en el ambiente indicado.

Con la entrega continua, se crean, prueban y preparan automáticamente los cambios en el código y se entregan a producción. La diferencia entre la entrega continua e implementación continua es que la forma de aprobar el pasaje al ambiente productivo. En la entrega continua se hace en forma manual, y es la forma utilizada en el banco, ya que antes de pasar a producción debe ser validado/aprobado el deploys (Ver Figura **22**).

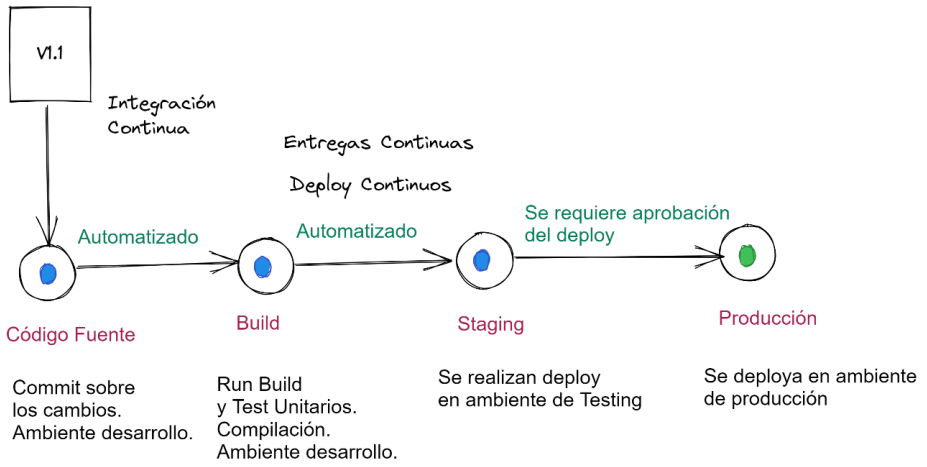

Figura 22 Integración Continua, entrega continua y deploys continuos

Ambientes

El banco tiene tres ambientes definidos para poder implementar su operatoria sobre los módulos del Core, los mimos se muestran en la Figura **23** y se detallan a continuación:

- Ambiente Desarrollo: Ambiente utilizado para el desarrollo de nuevas aplicaciones o mantenimiento de las aplicaciones existentes. El equipo de desarrollo realiza sus pruebas unitarias/integradoras en este ambiente con el fin de no ensuciar el ambiente productivo.
  Cuando el desarrollo de un requerimiento es terminado, se pasan todos los componentes de dicho desarrollo al ambiente de Testing.

- Ambiente Testing: Ambiente solo accedido por el equipo de testing, el cual se encargará de realizar las pruebas exhaustivas de los requerimientos que fueron entregados por el equipo de desarrollo. Todos los errores detectados por el equipo de testing son informados al equipo de desarrollo. Cuando el equipo de desarrollo corrige los errores, entonces se vuelve a deployar la nueva versión del componente en el ambiente de testing.
  Cuando el equipo de testing termina con las pruebas de los componentes de la aplicación, entonces se deployara una nueva versión en el ambiente productivo.

- Ambiente Productivo: En el ambiente productivo se encuentran todas las aplicaciones testeadas y estables. Son las aplicaciones utilizadas por los usuarios.
  A este ambiente solo tiene acceso el equipo de producción, que son los encargados de llevar a cabo el monitoreo de las aplicaciones. Es importante destacar que los errores producidos en este ambiente muchas veces deben solventarse en el momento por el equipo de producción debido a la criticidad, y no se puede esperar a que el equipo de desarrollo pueda corregir dicho error. Si el error que se produjo no es tan crítico, entonces en ese caso se delega al equipo de desarrollo para que lo resuelva.

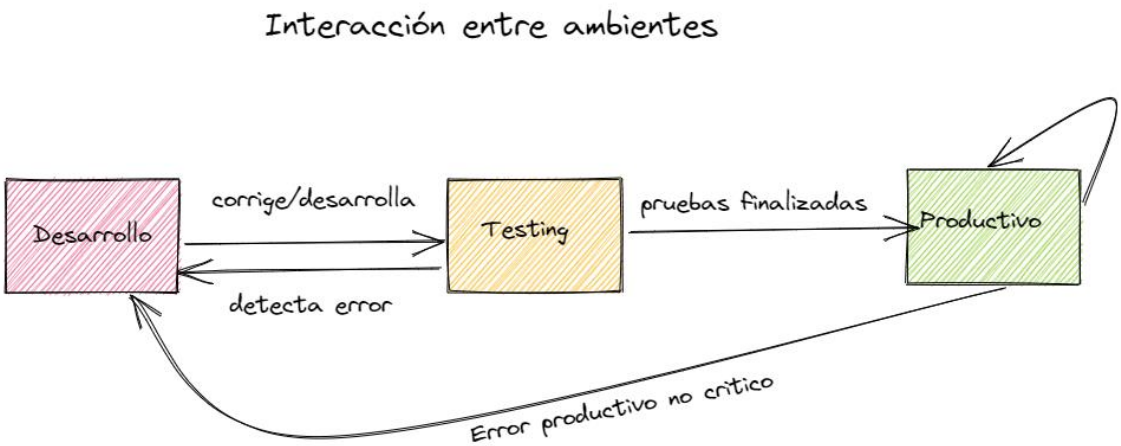

Figura 23 Ambientes Del Banco

Semanalmente se realizan actualizaciones de los ambientes de desarrollo y testing con el contenido del ambiente productivo, con el fin de tener lo último en cada uno de los ambientes.

Herramientas

A continuación, se describen las herramientas utilizadas en el banco. Se detallará el uso de repositorios en Bitbucket, el modelo utilizado en la herramienta GIT, deploys de release de códigos a través de herramientas como Bamboo y Harness, y por último se mostrará la gestión de las tareas a través de JIRA.

Bitbucket

Bitbucket es un repositorio para probar y desplegar código, y en el banco se integra a otras herramientas como GIT, Jira, Bamboo o Harness, con el objetivo de mejorar la calidad del código y desplegar pruebas automatizada. Por otro lado, nos permite mantener el control de versiones mediante aprobaciones previamente designadas, así mismo permite implementar opciones de control sobre las ramas, como por ejemplo la comprobación de fusiones sobre las mismas (Atlassian, 2022).

En el banco disponemos de 2 repositorios; un repositorio de Bitbucket, en el cual colocaremos nuestro código correspondiente a clases, rutinas, etc., y otro repositorio Bitbucket donde alojaremos nuestras aplicaciones de servicios.

En cuanto al desarrollo en el banco, cada desarrollador baja la última versión estable del componente que se encuentra en repositorio Bitbucket, genera su branch o rama (checkout) y trabaja en él. Cuando termina de modificar y probar el componente, el mismo es subido al repositorio Bitbucket a la rama principal (master) realizando un pull request. El pull request deberá ser validado y aprobado para que pase a la rama principal (master).

GIT

Git es un sistema de control de versiones distribuido de código abierto y gratuito, está diseñado para manejar todo tipo de proyectos con velocidad y eficiencia (Git, 2022).

En cuanto a Git en el banco se utiliza el modelo de desarrollo basado en ramas. El desarrollo basado en ramas es una práctica de control de versiones donde los programadores suben sus actualizaciones de forma frecuente en una rama principal (master). La rama master o principal almacena el historial de publicación oficial(estable) y la rama desarrollo sirve como rama de integración para los nuevos cambios ver Figura **24** (ZETTLER, s.f.). Es importante mencionar que GIT maneja el control de versiones (mediante sus comandos pull, commit, push, checkout, etc.) sobre los componentes desarrollados en los repositorios de Bitbucket del banco.

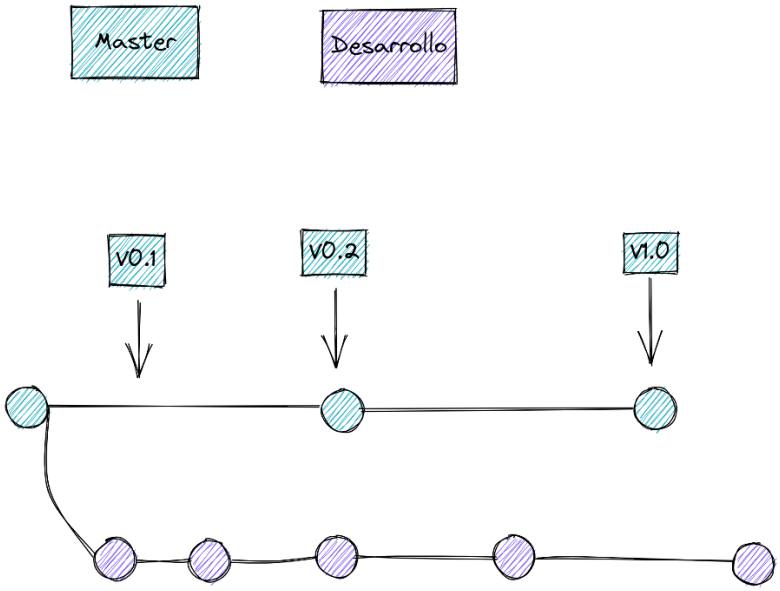

Figura 24 modelo basado en ramas

Principales comandos GIT (GitHub, 2022)

- **Init:** Inicia un repositorio de git dentro de un directorio ya existente, es decir añade todo lo necesario para hacer uso del sistema de control de versiones.
- **Clone:** Descarga todos los archivos de un repositorio creando una copia local.
- **Add**: Rastrea los cambios que se han realizado en los archivos.
- **Commit:** Previamente ejecutado el comando git add, con commit se confirman los cambios y se registra el cambio en el historial del repositorio.
- **Pull:** Actualiza la rama de desarrollo local en referencia a la misma rama que se encuentra en forma remota en el servicio que alberga el repositorio.

- **Push:** Envía al repositorio remoto los cambios de una rama que se realizaron de forma local.
- **Branch:** Consiste en una bifurcación del proyecto en otra línea de cambios, existe la rama principal llamada master, otras ramas que pueden ser creadas de acuerdo a las necesidades.
- **Merge:** Permite la combinación de varias ramas de desarrollo, con el fin de unir los distintos cambios generados en cada una de ellas.

Bamboo

La herramienta Bamboo es también utilizada en la integración continua, ya que puede ser configurada con Bitbucket ,GIT y los tableros (JIRA) obteniendo muy buenos resultados.

Bamboo es una herramienta de integración continua y despliegue que reúne compilaciones, pruebas y versiones automatizadas en un solo flujo.

En el banco mediante la herramienta Bamboo se pueden detectar conflictos entre desarrollos, es decir me permite detectar cuanto antes cuando dos desarrolladores están creando un conflicto entre dos ramas de desarrollo.

Con Bamboo podemos obtener despliegue en forma automática, esto es muy importante porque ahorran tiempo a los desarrolladores, y además se evitan errores. Por ejemplo, si yo tengo un proceso de despliegue en 15 pasos es muy probable que sea más propenso al error que si esos pasos se encuentren automatizados en Bamboo y se realicen todos en forma automática con solo hacer un click.

Cuando Bamboo realiza un despliegue en forma automática, y ninguno de los pasos generaron error entonces lo que genera es una nueva release (por lo general es un tag con un número), la cual ya queda disponible para ser deployada mediante la Herramienta Harness.

En forma resumida, con Bamboo lo que se hace en el banco es la compilación del código de los componentes desarrollados y además realiza una corrida de los Test Case definidos en JUnit (es un conjunto de clases opensource que nos permiten probar nuestras aplicaciones Java). Si la compilación y la corrida de JUnit son exitosos, entonces se generar una nueva release lista para deployar.

Harness

La herramienta Harness facilita el proceso entrega, logrando que éste sea seguro, escalable y autónomo. En el proceso de entrega del banco, luego de generada la release por medio de Bamboo, para que dicha release se encuentre disponible o visible desde Harness, previamente debe ser validado y aprobado el pull request de los cambios realizados sobre el repositorio de Bitbucket. Dicho esto, una vez aprobado el pull request, la release se encuentre disponible para ser deployada por Harness, y solo resta indicar en Harness cuál es la release que deseamos deployar y en qué ambiente los deseamos realizar.

Jira

Es una herramienta utilizada para la gestión del proyecto y administración de incidencias, diseñado para ayudar a visualizar el trabajo, limitar el trabajo en curso y maximizar la eficiencia.

Incidencia y tipos de incidencias

Las incidencias son los objetos de negocio que se manejan en la herramienta Jira. Las incidencias pueden estar relacionadas tanto con el desarrollo de los componentes como así también con los cambios en busca de la mejora del software. Es decir que las incidencias pueden ser de cualquier naturaleza (Guillermo Pablo Marcos, 2020).
Aunque Jira proporcione más tipos de incidencias, el banco maneja los siguientes tipos:

- Bug: Este tipo de incidencia es creada cuando el equipo de testing encuentra un error en el componente desarrollado.
- Improvement: Las incidencias que necesitan realizar una mejora o modificación sobre un componente, son creadas con el tipo de incidencia Improvement.
- New feature: Este tipo de incidencias es creado cuando se necesita desarrollar un nuevo componente.

Estados

Por defecto, los estados definidos en el flujo de trabajo son los siguientes:

- Open: Es el estado inicial luego de creada una incidencia.
- In Progress: Es el estado que indica que la incidencia se encuentra en desarrollo.
- Resolved: Es un estado que indica que la incidencia se encuentra resuelta pero aún no cerrada. Es un paso previo a cerrarse la incidencia, ya que falta algún tipo de revisión antes de realizar la entrega la del componente.
- Closed: Es un estado que indica que la incidencia se encuentra cerrada, ya que se realizó la entrega del componente.
- Reopened: Es un estado de una incidencia que se ha vuelto a abrir después de haber estado resuelta o cerrada.

Flujos de trabajo

Por defecto, Jira proporciona un flujo de trabajo que no puede modificarse.
En la Figura **25** se pueden observar los distintos estados y transiciones entre dichos estados.

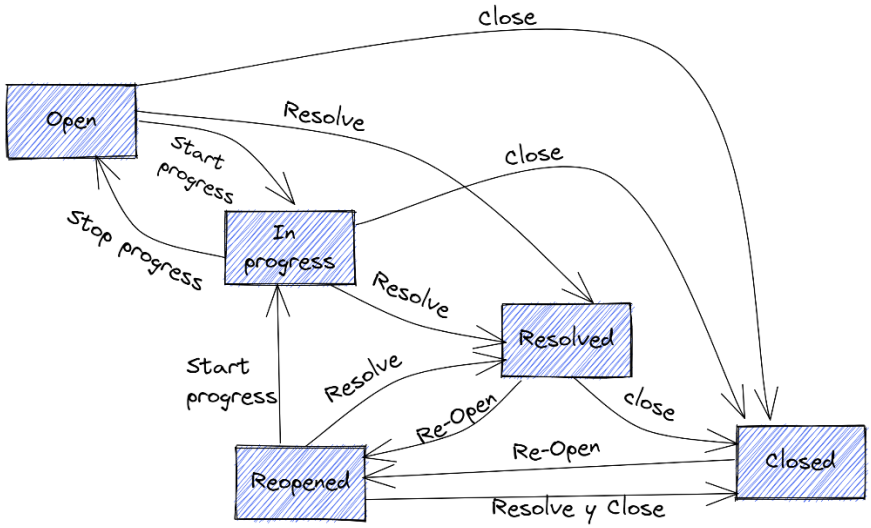

Figura 25 Flujos Trabajo extraída de ***(Guillermo Pablo Marcos, 2020)***

Proceso de reingeniería

La reingeniería de un sistema heredado (en nuestro caso es un sistema Cobol Mainframe) hacia el Core-S, no es una tarea sencilla, la cual demanda un gran esfuerzo de años por parte de los miembros de los distintos equipos del banco para poder lograr una estabilidad en cuanto a funcionamiento del nuevo sistema. A continuación, se describen los pasos de la reingeniería aplicada en el banco, para el desarrollo del componente *apertura de cuenta* en el Core-S.

1. Del sistema Cobol Mainframe se extrae información sobre la *apertura de cuenta* por parte del analista funcional. El analista funcional analiza de acuerdo a las estructuras y datos del nuevo sistema del Core-S como debe ser replicado el comportamiento de *apertura de cuenta*. Por otro lado, el analista funcional deberá analizar si es necesario agregar funcionalidad que no exista en el sistema anterior.

2. El analista funcional genera un documento especificación funcional de *apertura de cuenta*, en el cual se detallan las validaciones necesarias. Entre las validaciones más importantes podemos encontrar:

    - El cliente deberá encontrarse previamente cargado en el sistema al momento de dar de alta la cuenta, esta validación ya viene implementada como funcionalidad nativa del Core-S con lo cual el desarrollador/programador no deberá preocuparse por esta validación.
    - La cuenta (caja de ahorro, cuenta corriente) que se desea realizar la apertura es en moneda local(pesos), con lo cual no será necesario ningún tipo de validación extra, solo deberá validarse que el cliente se encuentre activo en el sistema.

- La cuenta (caja de ahorro, cuenta corriente) que se desea realizar la apertura es en moneda extranjera, se deberá validar que el cliente posee ingresos y/o activos consistentes en moneda extranjera.
- En todos los casos de *apertura de cuenta* se deberán validar en que caso aplica el cobro de comisiones en las distintas transacciones que realiza, o una comisión en el mantenimiento de la cuenta en caso de corresponder.
- En todos los casos de apertura de cuenta se deberá invocar al módulo fiscal para determinar los movimientos que podrán ser realizados.

El analista funcional adjunta a la especificación funcional un documento con los casos de pruebas necesarios para la *apertura de cuenta*.

3. Para el caso que el código de *apertura de cuenta* que se encuentra en el sistema Cobol Mainframe tiene redundancia en el código (lecturas innecesarias, escrituras invocadas incorrectamente, bloques de código repetido, etc.), será el analista funcional quien escriba en la especificación funcional de *apertura de cuenta*, de forma tal de hacer un análisis quitando toda la redundancia de código encontrada, y de esta manera cuando se realice la migración hacia el Core-S ya se hará sin la redundancia en el código. Este paso solo se realiza en caso de ser necesario.

4. El documento de especificación funcional de *apertura de cuenta* es enviado al equipo de desarrollo, donde será asignado a un programador/desarrollador quien será el encargado de analizar el documento, realizar todas las consultas necesarias al analista funcional para así poder avanzar con la codificación y customización necesaria del componente en el ambiente de desarrollo. Para la solución del componente de apertura de cuenta, el programador/desarrollador incluirá las funcionalidades nativas de caja de ahorro y cuentas corrientes del Core elegido. Es importante destacar que para el desarrollo del componente se tiene a disposición una gran variedad de funcionalidad de caja de ahorro y cuentas corrientes que ya se encuentran implementadas por el Core-S. Son ejemplos de esto último, la validación de existencia del cliente, validación de saldo de la cuenta, validación de la categoría (cuenta corriente, cuenta caja de ahorro), etc. Todas estas validaciones que ya se encuentran realizadas por el Core-S, permiten al programador/desarrollador que se pueda centrar más en el detalle del negocio. En el caso que el programador/desarrollador no pueda customizar con funcionalidad nativa, entonces deberá implementar el código que dé solución al negocio. Por ejemplo, para el cobro de comisiones sobre las cuentas, será el programador/desarrollador quien tenga que implementar su código del cobro de las mismas, validando en que caso aplicaría (según los tipos de cuentas) y de cuanto sería el monto a cobrar en caso de aplicar.
   Cuando el componente apertura de cuenta es finalizado por el programador/desarrollador, el siguiente paso es deployar en el ambiente de testing. El equipo de testing toma el componente de apertura de cuenta y será

el encargado de aplicar todas las pruebas unitarias e integrales que fueron adjuntadas por el analista funcional en el documento especificación funcional de apertura de cuenta. Si el equipo de testing no detecta ninguna incidencia, entonces se procederá a deployar en producción el componente de apertura de cuenta. En el caso de que el equipo de testing detecte un error en las pruebas, entonces se le comunica al equipo de desarrollo para que modifique el componente de acuerdo a los errores detectados.

5. Se deploya el componente apertura de cuenta en el ambiente productivo del banco, el cual será consumido por los clientes y usuarios del sistema.
   Esta será la versión más estable de apertura de cuenta, y de aquí se deberán tomar las futuras modificaciones que se realicen al componente.

Ejemplos de funcionalidades redefinidas en Core-S

El proceso descripto en el punto anterior se realiza para cada una de las funcionalidades reimplementadas. A modo de ejemplo, se detalla a continuación, el comportamiento nativo y customización de los módulos *Cuenta* y *Transferencia* tomando como referencia el Core-S.

Operaciones nativas módulo Cuenta

El módulo de Cuenta posee una gran variedad de operaciones que pueden ser realizadas sin el agregado de una customización por parte del desarrollador. La operación nativa más común es la modificación del cliente, categoría (la categoría corresponde al tipo de cuenta, es decir si corresponde a una cuenta corriente, caja de ahorro, etc.), título (corresponde al nombre de la persona titular de la cuenta). La Figura **26** muestra una cuenta en donde se pueden modificar los campos mencionados.

Figura 26 Modificación Datos Cuenta

Es importante remarcar que, si se ingresa un cliente inexistente, se muestra un error por parte del Core-S, indicando que el cliente no existe en el sistema (Ver Figura **27**).

Figura 27 Error Core en Cuenta

Otra operación nativa en las Core-S que vale la pena mencionar es el estado de la cuenta. Se puede acceder directamente a la aplicación Cuenta para cambiar o consultar su estado. Este comportamiento es observado en la Figura **28**.

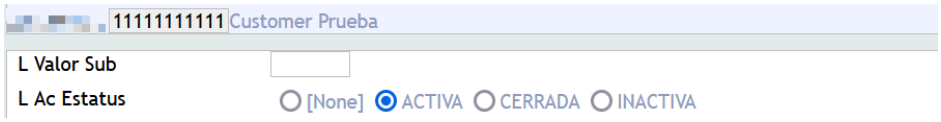

Figura 28 Estado Cuenta

Por último, mostramos una operación de las más comunes en el módulo de cuenta, como lo es consultar su saldo. En la Figura **29** se muestra la consulta del saldo de la cuenta. Si lo que se desea es modificar el saldo, directamente se cambia el valor del campo, y el Core-S realizara la validación que sea necesaria (por ejemplo, en el valor ingresado como saldo el Core-S validará que todos sus dígitos sean numéricos).

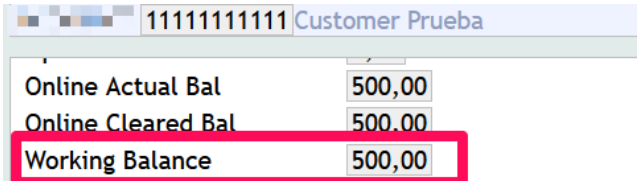

Figura 29 Saldo Cuenta

Operaciones customizables módulo Cuenta

Sobre las operaciones nativas mencionadas anteriormente, puede suceder que no alcance el comportamiento para resolver nuestra necesidad o problema. Por ejemplo, si se requiere consultar más información sobre una cuenta (ver Figura **30**), en ese caso se deberá desarrollar una rutina por parte de un desarrollador, que valide que la cuenta exista, y que extraiga la información correspondiente del módulo de cuenta. Esto es lo que llamamos customización, es decir que le agregamos comportamiento extra al que ya viene implementado como nativa en el módulo de cuenta del Core-S.

Figura 30 Operación No Core sobre Cuenta

Operaciones nativas módulo Transferencia

El módulo de Transferencia posee una gran variedad de operaciones que pueden ser realizadas sin el agregado de una customización por parte del desarrollador. La operación nativa más común es la transferencia entre dos cuentas válidas, en donde se puede observar que, con tan solo ingresar dos cuentas, moneda y monto de la transacción ya podemos realizar una transferencia. Esto se puede ver en la Figura **31**, donde además de los datos ingresados para realizar la transferencia, se puede observar que se ha realizado e impacto el saldo en la cuenta producto de la ejecución de la transferencia.

Figura 31 Transferencia entre cuentas

Es importante destacar que cuando se ingresa una cuenta inexistente en el sistema, se mostrara un error por parte del Core-S indicándonos que la cuenta no existe. Este comportamiento es mostrado en la Figura **32**.

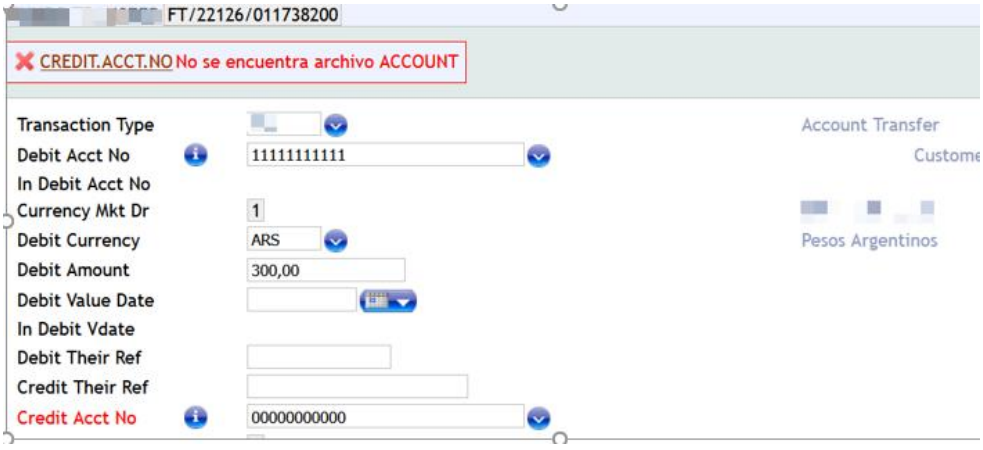

Figura 32 Transferencia Error Core

En lo que respecta a las transferencias bancarias, las mismas pueden ser modificadas para que se realicen operaciones con moneda extranjera, para distintos montos.

Esas operaciones serán validadas por el Core-S si son posibles de realizar, y en caso de no poder realizarse el Core-S será la encargada de informar los errores correspondientes.
Es importante destacar que toda esta funcionalidad ya se encuentra implementada como nativa dentro del Core-S.

Operaciones customizables módulo Transferencia

Si se requiere consultar más información sobre un movimiento bancario en particular (transferencia bancaria), en ese caso no alcanzaría con el comportamiento nativo del módulo de Transferencia, y se deberá desarrollar una rutina por parte de un desarrollador, que valide el movimiento bancario ingresado sea válido, y que extraiga la información correspondiente del módulo de transferencia. Esto se muestra en la Figura **33**. Nuevamente este es un caso en cual debemos customizar el módulo de transferencia del Core-S, ya que su comportamiento nativo no nos alcanza para cubrir nuestra necesidad.

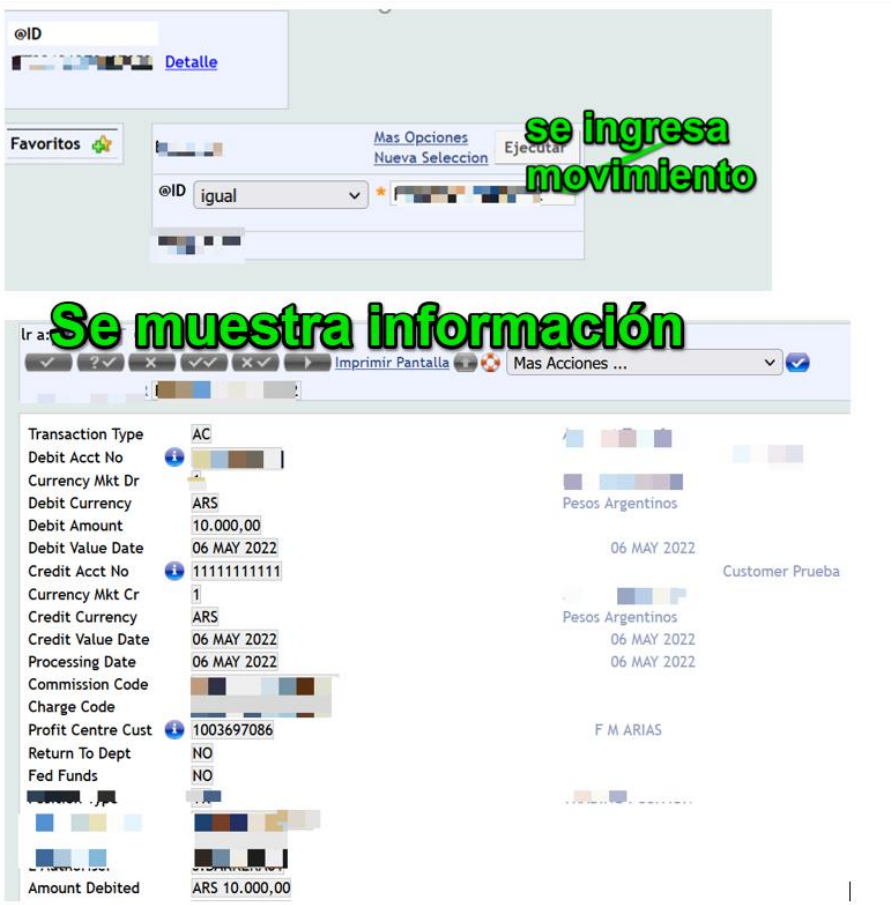

Figura 33 Transferencia – No Core

Capítulo 6 – Conclusiones y Trabajos futuros

Conclusiones

En el marco de la presente tesina se presenta la evolución del sistema bancario desde décadas anteriores hasta la actualidad. A continuación, se detallaron los sistemas en COBOL, en particular aquellos sistemas desarrollados en Mainframe. Se mencionaron las ventajas de la época por los cuales fueron desarrollados, así como también se describieron los distintos problemas que tienen dichos sistemas en la actualidad.
Seguido se definió el concepto de Core Bancario, sus objetivos y sus características, así como también se describieron en forma teórica las distintas arquitecturas existentes, en especial SOA la arquitectura utilizada por el banco. La misma se compone de distintos módulos definidos por el Core Bancario. Por otro lado, se detalló la funcionalidad definida por el Core. Luego, se presentaron las aplicaciones core, enumerando sus ventajas, y además se relevaron cuatro Cores disponibles comparando los beneficios de cada uno.
Posteriormente se estudiaron los diferentes tipos de rejuvenecimiento de software, detallándose en particular los beneficios de la reingeniería en un proceso de migración. A continuación, se describieron los pasos de reingeniería aplicados por el banco en el proceso de migración del sistema Cobol hacia el sistema de aplicaciones core.
En la siguiente sección se describió la infraestructura utilizada en el banco, la cual se apoya en los servicios locales y en la nube. Se definieron las buenas prácticas implementadas tanto para la integración, entrega y despliegue continuo, utilizando herramientas como Bitbucket, GIT, Bamboo y Harness optimizando la salida a producción. La utilización de herramientas mencionadas fue fundamental para el proceso de reingeniería implementado. Se presento Core-S como el sistema destino del proceso de reingeniería implementado en el banco.
El proceso se aplicó a cada una de las funcionalidades migradas. A modo de ejemplo, se detalló el proceso utilizado en la migración del componente apertura de cuenta hacia el Core-S. Se describieron las distintas etapas, desde que el analista funcional realiza la especificación de la apertura de cuenta existente en el sistema Mainframe, hasta que dicha especificación fue desarrollada sobre el Core-S por el programador, testeada y desplegada en el ambiente de producción. Por último, se describieron y detallaron algunos ejemplos sobre las funcionalidades nativas y customización sobre el Core-S, haciendo especial hincapié en los módulos Cuenta y Transferencia.
El proceso de reingeniería implementado, nos ayudó a migrar un sistema monolítico que se encontraba desarrollado en Cobol Mainframe el cual no podía hacer frente a las distintas necesidades actuales de los clientes, las cuales involucran mucha tecnología moderna.
Como conclusión final podemos decir que el banco ha migrado la mayor parte de sus funcionalidades que se encontraban desarrolladas en Cobol hacia el nuevo sistema de aplicaciones core. En particular, el proceso de reingeniería implementado ha beneficiado al banco, ya que ha permitido migrar mucha funcionalidad de procesos críticos, permitiendo mantener una alta tasa de rendimiento y estabilidad en sus servicios productivos (consumidos por los usuarios) durante el proceso de migración. Esto es muy importante para el banco en cuanto al negocio, ya que la funcionalidad del sistema no ha sufrido ningún cambio, siendo la migración transparente a los usuarios del sistema. Por otro lado, el nuevo sistema de

aplicaciones core se adaptó perfectamente al proceso de reingeniería, debido a la gran cantidad de operaciones nativas que contiene y su gran poder de customización que han permitido a el desarrollador que se enfoque en el negocio bancario logrando una migración exitosa del sistema.

Trabajos futuros

Continuar con el proceso de reingeniería en la migración de aquella funcionalidad que aún se encuentra en el sistema Cobol Mainframe, con la finalidad de tener en forma estable toda la funcionalidad del banco en el nuevo sistema de aplicaciones core.
El banco, actualmente tiene como uno de sus objetivos ser un banco completamente digital, para poder lograrlo se deberá seguir avanzando con el proceso de reingeniería en la migración hacia la nube de aquellos servicios que a la fecha se encuentran en forma local.

Bibliografía

ALAN MARQUEZ ESCORCIA, J. L. (2018). *SOFTWARE PARA LA RECUPERACIÓN DEL CONOCIMIENTO DE SISTEMAS HEREDADOS CONSTRUIDOS EN COBOL.*

Atlassian. (Noviembre de 2022). *https://www.atlassian.com/es/software/bitbucket*.

Chambó, F. (2015). *Framework para la automatización de las pruebas de integración en aplicaciones orientadas a servicios.*

Daniel López, E. M. (2017). Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web.

Fowler, M. (2006). *Continuous Integration*. Obtenido de martinfowler.com: https://martinfowler.com/articles/continuousIntegration.html

Git. (Noviembre de 2022). Obtenido de https://git-scm.com/

GitHub. (Noviembre de 2022). *https://docs.github.com/en/get-started/using-git/about-git*. Obtenido de docs.github.com.

Gruber, A. (2019). *Deduction of a Technical Modernization Process for the Sofware Arquitecture of Core Banking System.*

Guillermo Pablo Marcos, F. L. (2020). *Integració n del entórnó Bónitasóft cón la herramienta de Gestión de Incidencias Jira.*

Héctor Sánchez San Blas, P. G. (2021). *Avances en Informática y Automática.*

HIDALGO, R. A. (2016). DESARROLLO DE UNA METODOLOGIA PARA LA IMPLANTACIÓN DE UN CORE BANCARIO.

Kreća, M. (2015). *Comparative Analysis of Core Banking Solutions in Serbia.*

Madariaga, P. P. (2016). Cluster in a box: z Systems, COBOL y DB2.

Molina Ríos Jimmy, V. P. (2015). *Diseño de sistemas.* UTMACH.

Pfleeger, S. L. (2010). *SOFTWARE ENGINEERING THEORY AND PRACTICE.*

Pressman, R. S. (2010). *Ingeniería del software un enfoque práctico.*

Sommerville, I. (2009). *INGENIERÍA DE SOFTWARE.* PEARSON.

TEMENOS. (s.f). *A-survival-guide-for-core-banking-systems*. Obtenido de temenos.com.

Vaicilla, M. (2021). Generación de formularios utilizando la tecnología Angular para interactuar con el core bancario mediante servicios REST.

Valenciano, R. L. (s.f). Lenguajes de programación COBOL y PL/I, historia y.

ZETTLER, K. (s.f.). *Desarrollo basado en troncos*. Obtenido de https://www.atlassian.com/: https://www.atlassian.com/es/continuous-delivery/continuous-integration/trunk-based-development