UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Implementación de Plataforma de comercio electrónico

INFORME DE ACTIVIDADES PROFESIONALES

para obtener el título de

Ingeniero en Computación

P R E S E N T A

Juan Carlos Villegas Montiel

ASESOR DE INFORME

M.C. Alejandro Velázquez Mena

Ciudad Universitaria, Cd. Mx., 2018

Indice:

Introducción:

Este informe contiene el diseño y definición de la arquitectura de software de un sistema de comercio electrónico basado en una plataforma reconocida a nivel internacional. Dicha definición y arquitectura correspondieron a un proyecto de implementación desde cero de una plataforma capaz de dar servicio a un número elevado de usuarios concurrentes, alrededor de 2000 el primer año y capaz de incrementar este número un 20% en años subsecuentes, dicho proyecto fue realizado por la empresa para la que trabajo en conjunto con un proveedor externo ubicado en Italia y con un equipo de trabajo distribuido en tres locaciones diferentes: Ucrania, Italia y México.

Mi rol y principales actividades en este proyecto fueron el de ser arquitecto de software encargado de diseñar y definir la arquitectura del sistema además de definir de forma explícita todas las interfaces de comunicación con sistemas externos como son: sistema de cobro, de paquetería y de servicios diversos necesarios que para el correcto funcionamiento del sistema. Además de mis actividades como arquitecto también tuve la tarea de liderar al equipo de desarrollo en México y Ucrania, resolver cualquier duda técnica y exponer y justificar cada decisión hecha en el diseño ante el cliente final.

El principal objetivo de este informe es el demostrar la competencia en realización de actividades de Ingeniería de software para obtener el título de Ingeniería de la U.N.A.M.

La empresa para la que trabajo se llama *EPAM Systems* con oficinas principales en Pensilvania y oficinas distribuidas en Europa, Asia y Norte América, el giro de esta empresa es consultoría de software, los proyectos que tiene son mayormente de desarrollo de software además de algunos proyectos de soporte de software.

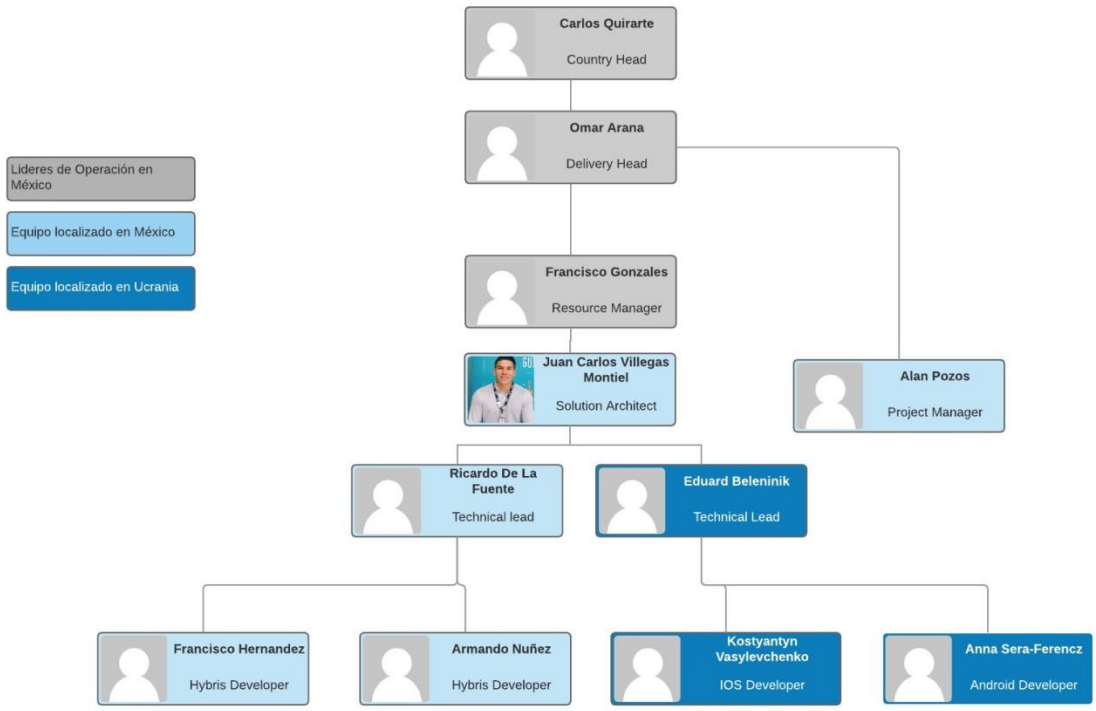
He trabajado por 2 años para EPAM desempeñando el rol de arquitecto de software, las tareas principales de este rol es el diseño y aseguramiento de la correcta implementación de sistemas de software, participo en ocasiones en más de un proyecto al mismo tiempo, realizando las mismas tares de manera específica para el caso de negocio de cada cliente, tengo interacción directa con clientes finales y también con personal técnico tanto del lado de mi empresa, así como de otros proveedores. Dentro de mis tareas habituales es la documentación necesaria para justificar las decisiones de diseño y explicar el flujo y patrones que deben usarse en la implementación.

Capítulo 1. Organigrama

El modelo de organización de EPAM consiste en unidades lideradas por una persona a la cual se le nombra *Resource Manager* (RM por sus siglas en inglés) dichas unidades pueden estar asignadas a un proyecto o varios sin importar si hay colaboradores de otros equipos. Todo empleado en la compañía tiene a un líder inmediato (RM) con el cual dirigirse en caso de alguna preocupación o problema ya sea laboral o personal, estos líderes tienen el rol de facilitadores para los colaboradores.

Cuando un ingeniero es asignado a un proyecto dicho proyecto tiene su propia estructura de acuerdo con las necesidades del negocio, pudiendo contar con un administrador de proyectos, ingeniero de pruebas, ingeniero de software, analista de negocio, etc. Sin embargo, todo colaborador siempre tendrá un RM asignado, aunque dentro de su proyecto haya una jerarquía particular.

El proyecto al que pertenecí tuvo una organización típica en el desarrollo de software, conformado por un líder de proyecto, arquitecto de soluciones, líder técnico y equipo de desarrollo. El equipo completo del proyecto se localizó en Ucrania y México, en el organigrama siguiente se muestra la jerarquía y línea de reporte de cada integrante además de su localización física.



Capítulo 2: Descripción de Proyectos

A continuación, se encuentra una descripción concisa de los proyectos relevantes en los cuales participé como miembro clave de la realización. Cabe mencionar que los diagramas presentados a continuación son una representación cercana a los diagramas originales, por motivos de confidencialidad no es posible usar los diagramas originales así que las ilustraciones aquí presentadas sirven para explicar la situación sin descuidar el factor discrecional de los datos:

Renovación del sistema de Infonavit para la precalificación.

Cliente: INFONAVIT

Tiempo: 2 años (2010-2012)

Nombre del proyecto: Renovación del sistema de Infonavit para la precalificación.

Descripción:

Situación inicial:

El sistema de precalificación de la institución pública INFONAVIT sirve a un alto número de usuarios concurrentes lo que lo convierte en un sistema crucial y propenso a fallos si no está bien implementado y diseñado. La implementación antigua, contemplaba el uso de componentes de viejas tecnologías como Mainframe con lenguaje COBOL y procesos *Shell* directamente desarrollados en el sistema operativo usado en los servidores de aplicaciones, para la interacción con el usuario final se tenían tecnologías basadas en el lenguaje de programación Java y .NET.

La arquitectura planteada en dicho sistema estaba basada en capas, haciendo una separación de intereses entre la parte de datos de usuarios, procesamiento de dicha información y la muestra de información al cliente. El sistema se puede acceder por canal web únicamente debido a que no estaba preparado para ser integrado con dispositivos móviles.

Problemática:

Debido a la alta concurrencia de usuarios y el poco mantenimiento que se le dio al sistema el rendimiento (*performance)* del sistema se vio degradado de tal manera que la experiencia de usuario fue empeorando hasta el momento de tener alto rechazo en su uso y poca fiabilidad. Los procesos de recopilación de datos tardaban demasiado tiempo en procesarse además de la falta de integridad en los datos guardados, es decir muchos datos se perdían en el proceso o carecían de coherencia al momento de leerse, esto

ocasionaba que se realizaran muchas correcciones manuales directamente en base de datos provocando alto riesgo de errores e incoherencias.

Se identificaron <<cuellos de botella>> sensibles que degradaban el rendimiento del sistema en demasía, dichos problemas se debían a la arquitectura monolítica de los sistemas usados basados en tecnologías viejas como Mainframe, Java 1.4, db2 en versiones viejas y *portlets.* La cantidad de transacciones que atendía el sistema de manera concurrente no era consecuente con el *throughput* (cantidad de transacciones atendidas en un intervalo de tiempo) que el sistema tenia de acuerdo con su diseño.

El sistema no tenía capacidad de integración con otros canales, es decir únicamente se podía tener acceso a sus servicios mediante la interfaz web diseñada y que además estaba altamente acoplada a capas inferiores en la arquitectura dificultando mucho su integración y mantenimiento.

La interfaz de usuario no era intuitiva y tenía un aspecto poco atractivo para el cliente esto ocasionaba que los usuarios fueran renuentes a utilizar dicha plataforma.

El uso de tecnologías antiguas dificultaba en demasía el mantenimiento y convirtió el sistema en dependiente de sólo algunas personas que conocían su uso, esto aunado a la documentación pobre convirtieron el sistema en un sistema altamente inestable.

A continuación, se muestra un diagrama de la arquitectura en el estado inicial:

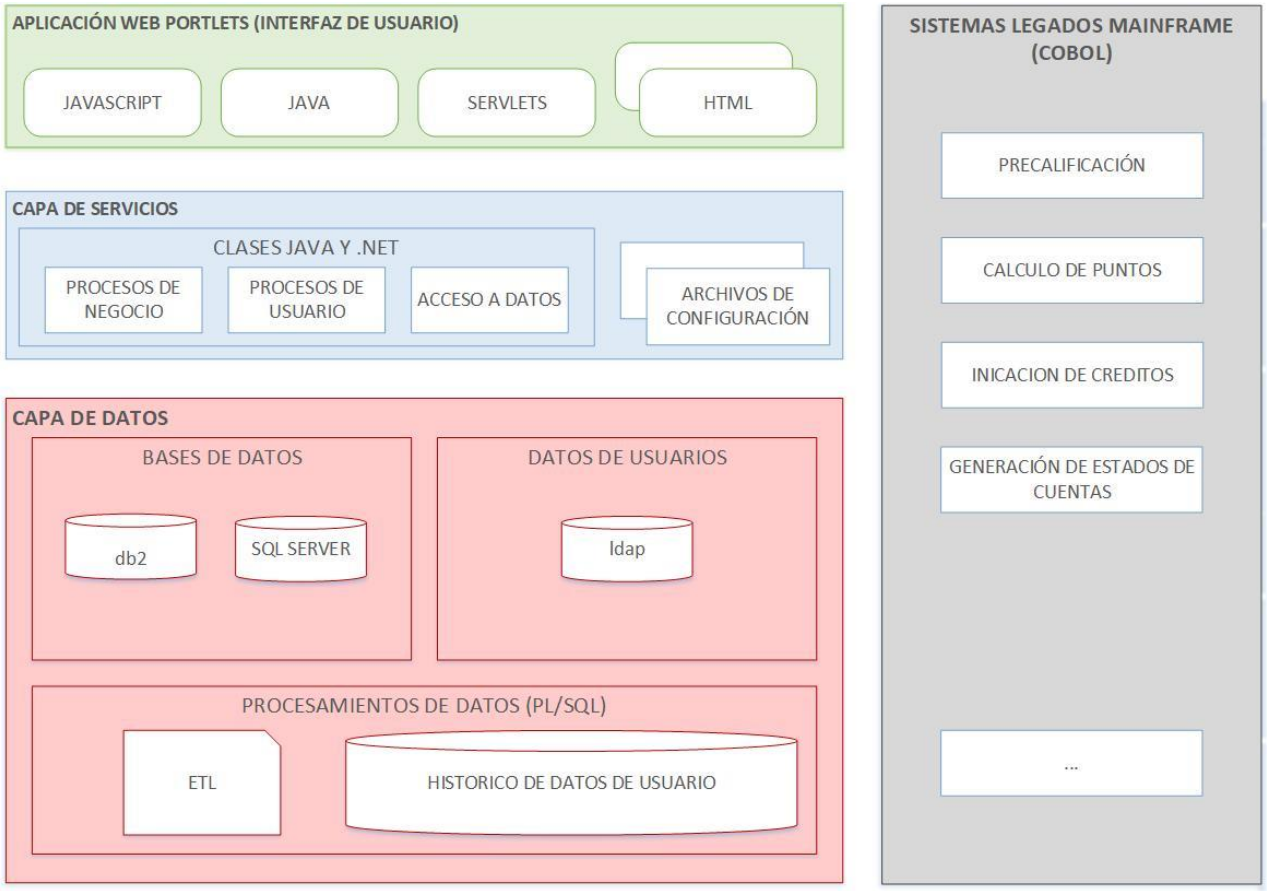


Figura 1.- arquitectura del sistema de Infonavit en su estado inicial

Solución y participación:

En este proyecto tome el rol de arquitecto de software Jr haciendo equipo con un arquitecto Sr. Juntos hicimos el diseño de la nueva arquitectura a la cual se migró Infonavit, en concreto mis actividades durante este proyecto comprenden:

Hice en primer lugar una identificación de los principales puntos de fallo del sistema comenzando desde la capa de datos y los procesos de carga y manejo de datos, identifiqué los procesos principales en esta capa que debían ser reutilizados en las siguientes fases dichos procesos programados en PL/SQL tenían mucha lógica de negocio por lo cual propuse hacer una abstracción de dicha lógica y desacoplar la capa de datos lo más posible para tener toda la lógica de negocio en una sola capa y de esta manera facilitar el mantenimiento.

Hice un análisis de la capa de interfaz de usuario para hacer una abstracción de la lógica de negocio y desacoplarla de dicha capa, de esta manera tendría un mejor control en cambios respecto a la interfaz sin afectar propiamente el funcionamiento principal del sistema.

Participe en el diseño e implementación de patrones de diseño de software conocidos como *DAO, factory, strategy, facade, command,* etc. que facilitan la lectura y además agregan más robustez y calidad al código, dicha implementación involucró una refactorización del código además de cambiar la versión de Java usada por la 1.6 (la más estable en ese entonces), así que analicé el impacto de cambiar la versión en los servidores de aplicaciones usados provistos por IBM, una vez hecho el análisis y asegurarme de la compatibilidad participé en pruebas de concepto de aplicativos con la nueva versión de java.

Una vez hecho el análisis de cada capa respecto a cuellos de botella y código a refactorizar procedí a diseñar la nueva arquitectura a la que migraríamos, este diseño lo hice en conjunto con el arquitecto sr. que fue parte de mi equipo. La nueva arquitectura fue basada en SOA (*Service Oriented Architecture*), su interoperabilidad y desacople entre capas dio la flexibilidad necesaria para facilitar el mantenimiento y funcionamiento del sistema en términos de rendimiento, para llevar a cabo dicho cambio en la arquitectura elaboré un documento de migración en donde expuse pros y contras de dicha migración además del esfuerzo necesario por recursos humanos y físicos para hacerla posible, enfaticé de manera puntual los riesgos que conllevaba no hacerla y lo que se ganaría en términos de negocio si era implantada.

Presenté dicho documento con un equipo técnico del cliente en una serie de reuniones en las cuales expertos en los flujos de negocio eran invitados para saber su opinión al respecto al igual que personal técnico del Infonavit, hice algunos cambios en dicho documento de acuerdo a comentarios y expectativas del cliente así como el contrato de servicios prestados establecido por la empresa para la cual trabajaba y una vez que todas las partes aprobaron el documento y por consiguiente la migración, procedimos con la implementación.

A continuación, muestro el diagrama a alto nivel de la arquitectura propuesta, nótese la inclusión de dos capas extra respecto al diagrama anterior.

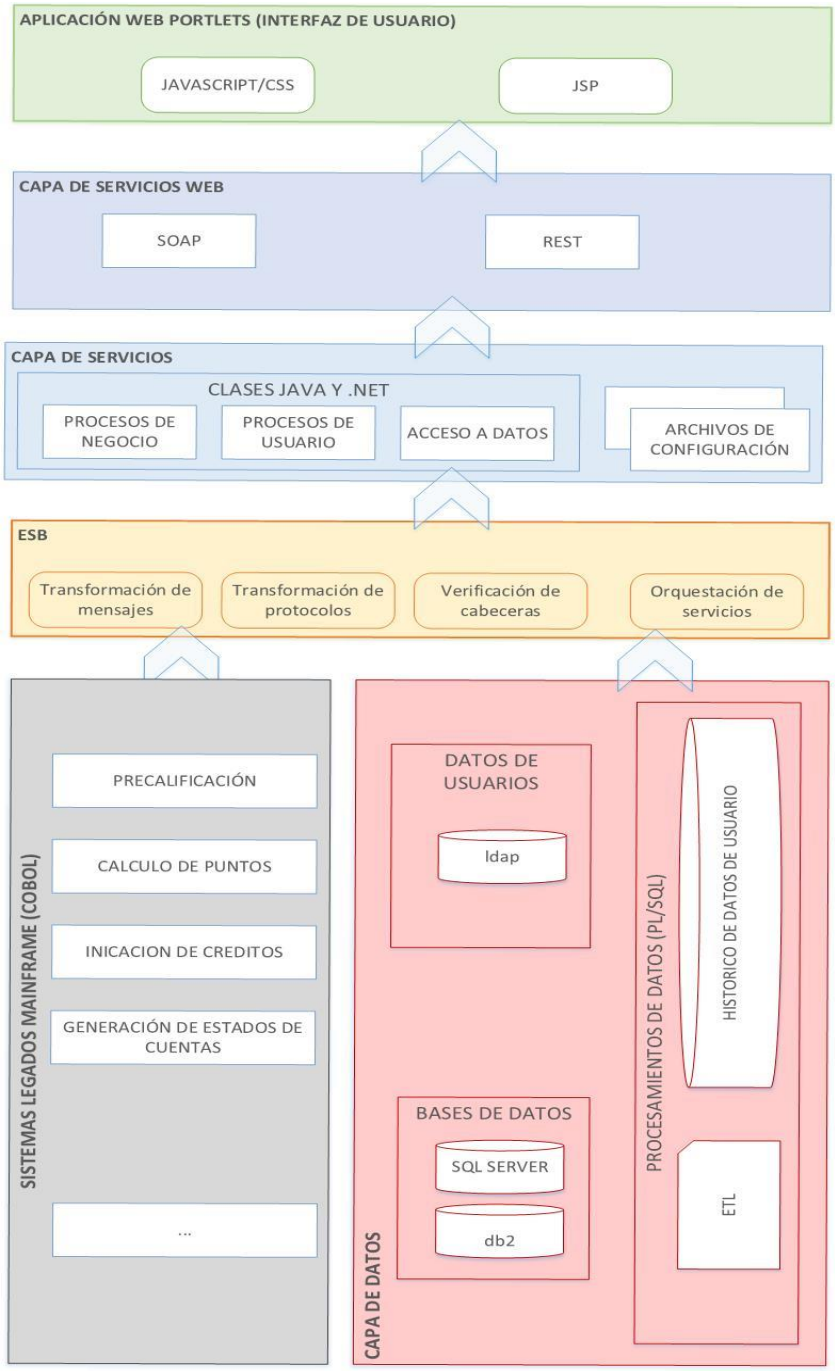


Figura 2.- arquitectura del sistema de Infonavit propuesta en SOA

En el diagrama anterior plasmé en términos generales y alto nivel la arquitectura propuesta para la migración del sistema, en términos sencillos hice lo siguiente:

1.- Hice una abstracción de cada capa del estado inicial del sistema para aislar la lógica de negocio embebida y que dificultaba el mantenimiento. En la capa de datos había validaciones de negocio que

extraje, de la misma manera en la capa de interfaz de usuario había mucha lógica de negocio que hacía muy difícil la identificación de problemas o incluso el flujo del proceso mismo.

2.-Una vez con los flujos de negocio debidamente identificados y desacoplados de capa de datos y de interfaz de usuario, procedí a hacer una re-fabricación del código, implementé patrones de diseño que añadieron mayor calidad y estabilidad al código además de delimitar claramente la división de capas a nivel lógica del sistema.

3.-Agregué una capa con un bus de servicios empresariales (ESB por sus siglas en ingles Enterprise Service Bus) para desacoplar los sistemas legados de la implementación en la capa de servicios.

Los sistemas legados del sistema, en este caso Mainframe, se comunican con otros componentes externos mediante cadenas de caracteres, estas cadenas contienen información del sistema o servicio al cual se accede o se requiere información. En el estado inicial del sistema los sistemas legados se integraban con la capa de servicios mediante un análisis de la cadena de caracteres recibida esto provocaba alto acoplamiento entre dichas capas y poca tolerancia a cambios.

Participé en la implementación y en el diseño de la capa de ESB, esta capa tuvo la principal función de desacoplar la integración de los sistemas legados con la capa de servicios, de esta manera dicha integración se volvió más flexible, debido a la naturaleza del ESB se puede hacer la integración de diferentes mensajes realizando una transformación dinámica es decir que tanto el cliente (capa de servicios) como el servidor (sistemas legados) se desentienden de dicha integración y la capa de ESB se encarga de hacer dicha transformación aunado a esto el ESB también puede hacer una conversión de protocolos, añadir y quitar cabeceras, esto resulta muy útil para exponer los servicios de sistemas legados como si fuesen servicios web usando http o enviar dicho contenido vía JMS, lo cual resulta muy útil favoreciendo la interoperabilidad entre sistemas internos.

4.- Diseñe y cree un arquetipo para crear servicios web usando el lenguaje java y de esta manera promover los procesos de la capa de servicios a un entorno web incrementando de esta manera las capacidades de integración con múltiples canales (web y móviles) y de una forma distribuida esto quiere decir que ahora con esta nueva capa fui capaz de exponer el nivel de servicios a un entorno web tanto con el protocolo SOAP y con REST, el protocolo SOAP lo usé para integrar la interfaz de usuario y con REST pude integrar exitosamente dichos servicios con sistemas operativos móviles esto fue planeado para futuras fases, de esta manera cubrí el requerimiento no funcional de escalabilidad.

5.-Con la nueva capa de servicios web la interfaz de usuario usando *portlets* quedó obsoleta así que diseñé el uso de una nueva tecnología basada en javascript y hojas de estilo (CSS) y JSP (*Java Server Pages*) con esto añadí flexibilidad a la interfaz de usuario y múltiples posibilidades de hacerla más atractiva, esto fue unos de los requerimientos explícitos del usuario y con este cambio de nuevas tecnologías logré darle ese atractivo a la interfaz haciendo que fuese mucho más amigable e intuitiva.

En resumen como arquitecto de software diseñé e implemente nuevos componentes que sirvieron para dar servicio a la interfaz de usuario y ejecutar los procesos de negocio requeridos por el sistema, diseñé una nueva arquitectura para la capa intermedia, es decir la capa encargada de comunicar la capa de usuario y los procesos de bases de datos, basada en la arquitectura SOA (Arquitectura orientada a Servicios) con esta nueva arquitectura y renovado enfoque pude lograr un escalamiento horizontal flexible que permitió mejorar el rendimiento del sistema y facilitar su mantenimiento por la naturaleza Inter operativa de dicha arquitectura. Lideré el equipo de desarrollo al mismo tiempo que implementé por mi cuenta servicios web cruciales en los procesos de negocio.

Trabajé en conjunto con otros ingenieros encargados de implementar y diseñar capas de interfaz de usuario y de base de datos. La comunicación entre los distintos equipos de desarrollo y arquitectura fue crucial para la correcta mejor del sistema.

Evolución operativa digital

Cliente: GNP

Tiempo: 2 años y medio (2012-2014)

Nombre del proyecto: Evolución operativa digital

Descripción:

Situación inicial:

Grupo Nacional Provincial (GNP) es una institución privada cuyo giro está basado en el servicio de seguros de distintos tipos desde gastos médicos mayores, autos, hasta seguro de vida.

Para su funcionamiento GNP utiliza diversos sistemas para sus procesos de negocio diarios estos sistemas tienen su propia base de datos además cuentan con sus propios dueños, es decir cada departamento encargado de algún proceso de negocio tiene su propia aplicación o sistema, equipo de desarrollo, su propia lógica de negocio y fue implementado en una tecnología elegida por convicción propia sin previa estandarización.

Durante la operación diaria hay varios procesos de negocio que convergen y dependen de áreas comunes como, documentación, actuaria, cotización de seguros, cálculo de primas, siniestros, entre otros; debido a dicha dependencia entre procesos los sistemas asociados a cada proceso también crearon dependencias entre ellos aun cuando no tienen conocimiento certero de su funcionamiento o incluso de dichas dependencias.

Para completar un flujo de negocio completo, por ejemplo, una emisión de una póliza de seguro era necesario el uso de más de dos sistemas para llevar a cabo todas las tareas asociadas al flujo principal, cada agente requería aprender a usar dichos sistemas en interfaces de usuario no estandarizadas y en los cuales se repetía información previamente cargada en otro sistema.

A continuación, muestro a manera ilustrativa un flujo de negocio que involucra el uso de diferentes aplicativos con 2 agentes involucrados, aquí podemos ver claramente que completar un flujo requería del uso de más de una sola plataforma.

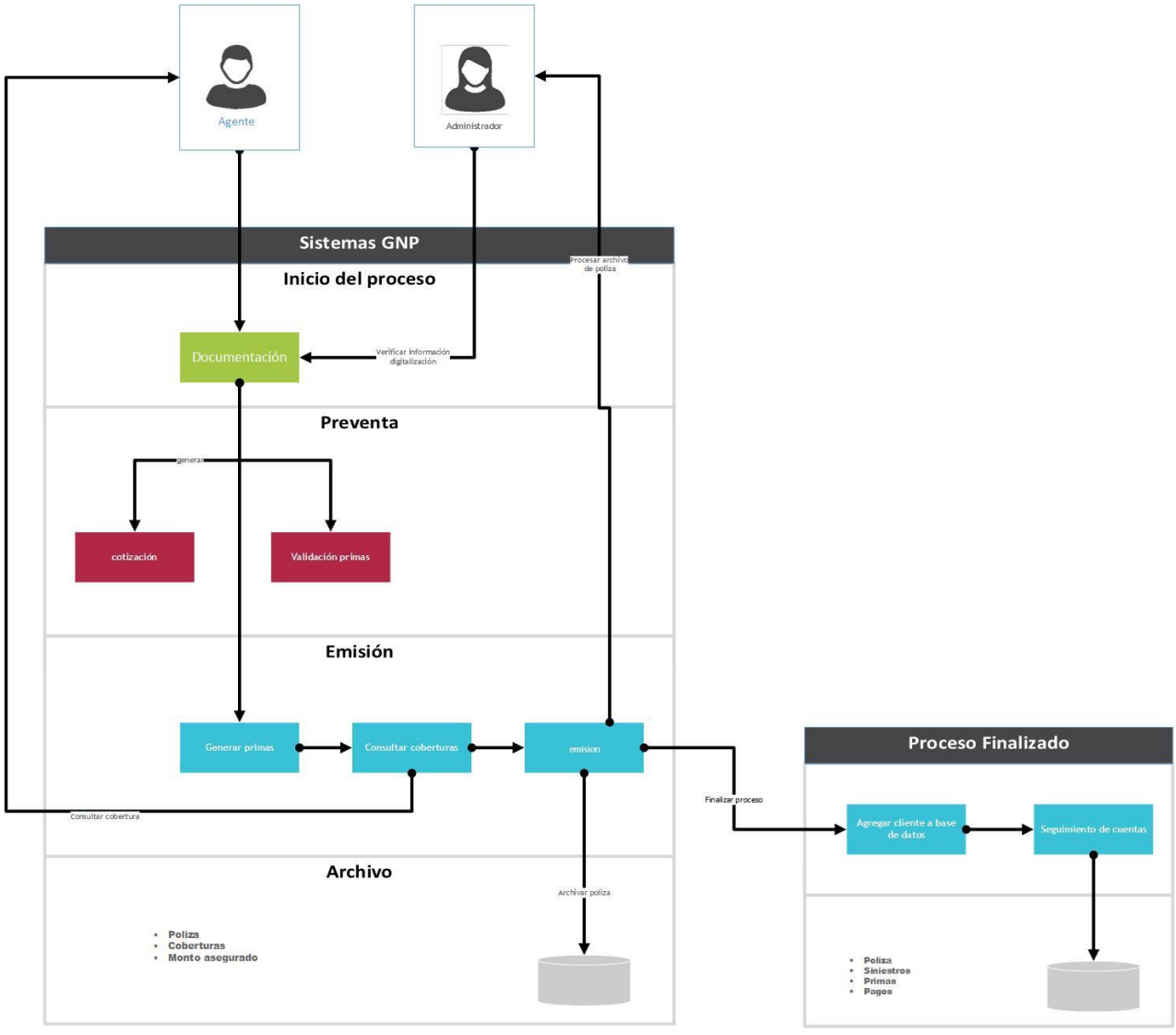


Figura 3.- Flujo de un proceso de negocio en GNP usando múltiples sistemas.

GNP no tenía un área formal encargada de estandarizar y supervisar/gobernar el desarrollo de nuevos sistemas para cada área de trabajo.

Problemática:

Los diferentes sistemas que usaba el personal interno requerían de mucho mantenimiento y el aprendizaje de cada sistema hacía que la operación se viera afectada sobre todo para nuevos ingresos, debido al uso de dichos sistemas la información era duplicada en diferentes lo cual era susceptible a errores de congruencia en información y al mismo tiempo se perdía información vital para el proceso como documentos de identidad personal del cliente final lo que provocaba la molestia al retrasar la tarea o requerir recolectar dichos documentos físicos una vez más.

La situación anterior ocasionaba que cada agente y persona involucrada en la emisión de seguros tuviera que aprender al menos 3 distintos sistemas totalmente distintos y con diferentes manuales de uso, la capacitación de dichos agentes era costosa en términos de dinero y tiempo, además que esto no garantizaba el correcto uso y carga de información, era muy común que un paso se olvidará durante el proceso y provocará la realización desde el principio de dicho proceso, además de información duplicada en el sistema.

La falta de normalización entre aplicaciones hacía que cada equipo dueño de cada sistema tuviera falta de visibilidad sobre el esfuerzo de otros equipos, esto ocasionaba el retrabajo y duplicidad de esfuerzo, muchos procesos comunes fueron implementados en distintos sitios en diferentes lenguajes de programación por diferentes equipos de desarrollo, dicho esfuerzo duplicado no era visible para todos debido a la falta de liderazgo en el área sistemas y gobierno de las aplicaciones.

Las diferentes líneas de negocio en GNP necesitaban un gobierno de aplicativos que estandarizara el desarrollo además de poner fin a la duplicidad de esfuerzos, el problema es que no había un área encargada o personal con los expertos para poder realizar estas tareas, debido a esta situación y problemática se contrataron proveedores externos que proporcionaran dicha consultoría y fueran parte de la plantilla de transformación del área de sistemas y desarrollo en GNP.

Solución y participación:

Fui contratado por GNP para formar parte del proyecto <<evolución operativa>> dicho proyecto tuvo el objetivo de unificar en una sola interfaz de usuario todos los sistemas usados en GNP, es decir eliminar por completo los procesos duplicados y el manejo excesivo de información repetida en diferentes sistemas internos, este reto involucró la creación de una nueva área en la compañía, es decir un área que fuese capaz de dirigir y coordinar dicho proyecto y de la misma manera fuesen administradores del

gobierno de todas las aplicaciones, dicha dirección involucró la creación de un equipo de desarrollo, un equipo de analistas de negocio, un grupo de administradores de proyecto y un grupo de arquitectura.

Fui parte del equipo de arquitectura con el rol de Arquitecto java/SOA participé en la implementación y diseño de un sistema distribuido capaz de unificar los procesos de negocio dispersos en las diferentes áreas, mis principales labores fueron las siguientes:

Diseñé la reutilización de procesos existentes, así como su integración con una nueva plataforma distribuida.

Hice un análisis de los diferentes sistemas existentes y su colaboración y procesos comunes para poder determinar su nivel de acoplamiento.

Participé en la definición de la arquitectura de la nueva plataforma basada en JAVA/SOA específicamente en la capa *Middleware* esta capa estaba encargada de exponer servicios web para las interfaces móviles y web del sistema, además de orquestar procesos de negocio mediante el uso de herramientas *Bussines Process Management* (BPM) diseñé arquetipos para la creación uniforme de servicios web con el uso de lenguaje java y *frameworks* bien conocidos como CXF, Spring, Hibernate. Configuré los servidores de aplicaciones que servirían de *host* para dichos servicios web de tal manera que fueran escalables y tuvieran soporte para alta disponibilidad y *failover* del sistema.

Documenté la arquitectura de los servicios y la configuración de los mismo además diseñé la propagación de errores para capas superiores del sistema, la arquitectura implementada fue SOA (Arquitectura Orientada a Servicios) debido a su interoperabilidad fue posible unificar diversos procesos de negocio que no estaban unificados e integrarlos a la nueva plataforma.

Participé en la mayoría de las definiciones de arquitectura de capas superiores e inferiores para asegurar la correcta integración, tuve exposición directa con el cliente final que en este caso eran perfiles de dirección dentro de la compañía GNP.

Participé en la realización de pruebas de concepto para asegurar la correcta integración de tecnologías nuevas con tecnologías legadas como Cobol.

Diseñé en conjunto con dos desarrolladores más la integración de la herramienta *Documentum* para fungir como repositorio digital para todo el sistema, de tal manera que se redujera al mínimo el archivado de documentos físicos dicha herramienta se accesa por medio de un mecanismo de integración basado en mensajería asíncrona así que hice uso de JMS (Java Message Service) de java para proveer dicha funcionalidad, cómo parte del aseguramiento de integración realicé la prueba de concepto correspondiente.

Como parte de la documentación del sistema colaboré con otros arquitectos en la elaboración de diagramas de arquitectura para la exposición y aclaración de la nueva arquitectura propuesta, a continuación, se encuentra una parte del diseño cuidando la privacidad propia del proyecto.

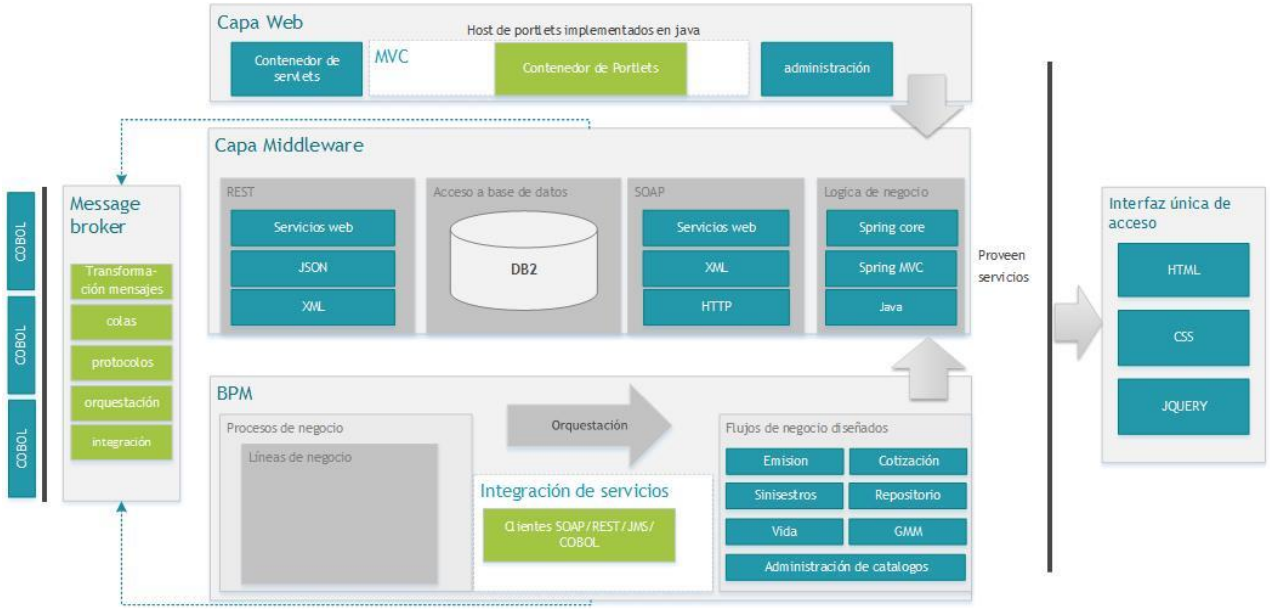


Figura 4.- Diagrama de arquitectura de sistema basado en SOA para GNP

La solución implementada trajo consigo cambios sustanciales en los procesos de GNP, sin embargo, el esfuerzo de unificación también optimizó los procesos realizados vía digital, el esfuerzo de cambios duró aproximadamente 3 años yo participé los primeros 2 años y medio hasta que llevamos a producción los primeros módulos del sistema.

El proceso de despliegue de los aplicativos constituyo un tema aparte. Participé diseñando el proceso de integración continúa usando herramientas como *Jenkins* y *Maven* para la construcción de los artefactos además de la verificación de pruebas de código y cobertura, con dichos procesos aseguré la calidad de los componentes que se desplegaron haciéndolos pasar por una serie de pruebas unitarias y de integración descritas mediante reglas y usando herramientas tales como Sonar.

Refactorización de sistemas Banamex-Citibank.

Cliente: Citibank

Tiempo: 2 años (2014-2016)

Nombre del proyecto: Renovación de arquitectura Banamex.

Descripción:

Situación inicial:

El banco de México BANAMEX es uno de los bancos con más clientes a nivel nacional, debido a un movimiento estratégico BANAMEX paso a ser parte de Citibank.

Los sistemas de BANAMEX tienen procesos de negocio con alta transaccionalidad y con información altamente sensible, toda la información es transmitida en una compleja serie de sistemas con diferentes tecnologías, java, .net, *mainframe*, *tuxedo*, etc. Así mismo hay diversos proveedores que proporcionan servicios a los sistemas de Banamex, proveedores de renombre como Oracle, IBM y Tibco son algunos de los más importantes en la arquitectura.

Debido a la complejidad de la arquitectura y los diversos sistemas involucrados en las distintas líneas de negocio del banco, la tarea de dar soporte es una tarea complicada al igual que la detección de errores. Complejos modelos de resolución de incidentes son implantados con el fin de agilizar la resolución en las aplicaciones que están funcionando en producción. El modelo de ITIL de *Problem Management* e *Incident Management* son usados para llevar una adecuada atención de los incidentes y responder en tiempo de nivel de atención a cada suceso registrado.

Hay muchos sistemas trabajando juntos para proveer servicio a los clientes de BANAMEX y para uso interno de sucursales y a nivel operativo, es decir tanto los sistemas de uso interno para procesos como inicialización de créditos, auditoria de cajeros, detección de fraudes, monitoreo de actividad sospechosa, etc. así mismo hay sistemas que proveen servicios al cliente final, como bancanet, sistemas de cajeros y de promociones basados en campañas. Todos los sistemas mencionados convergen en componentes comunes basados en una arquitectura que es producto de una combinación entre SOA, microservicios y *modelo monolítico*.

Todos los sistemas están divididos en líneas de negocio (Line of Business LOB por sus siglas en ingles), cada línea de negocio tiene propietarios de cada aplicación y una persona a cargo del buen funcionamiento de cada aplicativo, es decir una línea de negocio representa una parte importante en la operación del banco, por ejemplo, tenemos las líneas:

Digital. - para canales digitales como web y móviles, esta línea agrupa todas las aplicaciones que proveen servicios para los sistemas que comprenden la operación de la banca en línea por medio de un navegador web y también para la banca en aplicaciones móviles nativas. Para proporcionar estos servicios se requieren varios aplicativos funcionando en conjunto, cada aplicativo tiene un <<dueño del producto>> (Product Owner) quien se encarga de atender cualquier incidente respecto a su aplicativo y trabajar en conjunto con otros dueños en caso de afectación a sistemas dependientes.

Operaciones y fraudes. - para la detección de transacciones ilícitas y sospechosas; esta línea agrupa a todos los aplicativos que brindan servicios para la detención de fraudes en el banco lo cual puede comprender operaciones sospechosas, transacciones fuera de lo común, fondos ilícitos, área legal y reportes de comportamientos de clientes sospechosos, de igual manera que en otras líneas cada aplicativo tiene un <<dueño del producto >> que es encargado de responder en caso de cualquier incidente que involucre a su aplicativo o a aplicaciones dependientes de él.

De la misma manera existen más líneas de negocio que funcionan de manera similar y agrupan otros aplicativos.

Cabe mencionar que cada sistema o aplicativo es categorizado en prioridades, dicha prioridad es delimitada por colores, de tal manera que los aplicativos con bajo impacto a producción son categorizados como <<bronce>>, los de medio impacto como <<plata>> y los de alto impacto como <<oro>>.

Los aplicativos o sistemas categorizados como oro son usados por varios sistemas por lo que una falla en estos aplicativos involucra un paro en la producción tanto a nivel sucursal como a nivel cliente final, es por ello por lo que el monitoreo y nivel de servicio de respuesta en caso de incidentes es sumamente estricto. Cada aplicativo categorizado como oro es considerado como sensible y cada línea de negocio al menos tiene una aplicación oro.

Problemática:

Debido a la compra de BANAMEX por parte de Citibank, toda la arquitectura de los sistemas de Banamex tuvo que alinearse a los estándares y lineamientos de la arquitectura de Citibank esto conllevó varias refactorizaciones de código y modificaciones de componentes. Incluso cuando ambas arquitecturas tenían similitudes y en algunas ocasiones los mismos proveedores de servicios arriba mencionados (Oracle, IBM, Tibco, etc), en la realidad ambas arquitecturas distaban mucho de ser homologas, uno de los principales problemas fue implementar el paradigma de servicios agnósticos y orquestarlos por medio de una herramienta tipo BPM (*Business Process Management*) del proveedor Tibco llamaba *Business Works,* es decir hacer que las dependencias entre servicios desaparecieran y pudieran ser interoperables para de esta manera reducir el impacto en caso que alguno de esos servicios dejara de funcionar.

Tomando como motivo esta nueva estandarización se tomó la iniciativa de analizar, detectar y corregir <<cuellos de botella>> en el sistema que provocaban alta latencia y una mala experiencia de usuario. Estos cuellos de botella eran un problema constante en los sistemas de BANAMEX, no era extraño escuchar que <<el sistema de había caído>> hablando de los sistemas que servían como servicio a clientes finales.

Se detectaron varios cuellos de botellas a continuación describo dos de los más significativos:

1.- Múltiples transacciones concurrentes autorizadas por un solo sistema. Cada transacción ya sea de débito o deposito debe pasar por un sistema autorizador que a su vez comprueba que dichas transacciones no sean sospechosas de fraude mediante un monitoreo y comparación sobre comportamiento <<regular>> del cliente de manera que cada transacción que salga de dicha <<regularidad>> es sometida a monitoreo y comprobación de fraudes (ej. Un depósito de 100,000 pesos con una media de depósitos de 5,000 sería sometida a comprobación de fraude) , este sistema atiende a múltiples sistemas autorizando transacciones de canales digitales (web y móviles) y tradicionales (sistemas de cajeros, de ejecutivos y cajeros automáticos), usa diferentes instancias para atender todas las peticiones sin embargo si una falla se debe llevar a cabo un <<proceso de *batch*>> para recuperar las transacciones perdidas por esa instancia perdida.

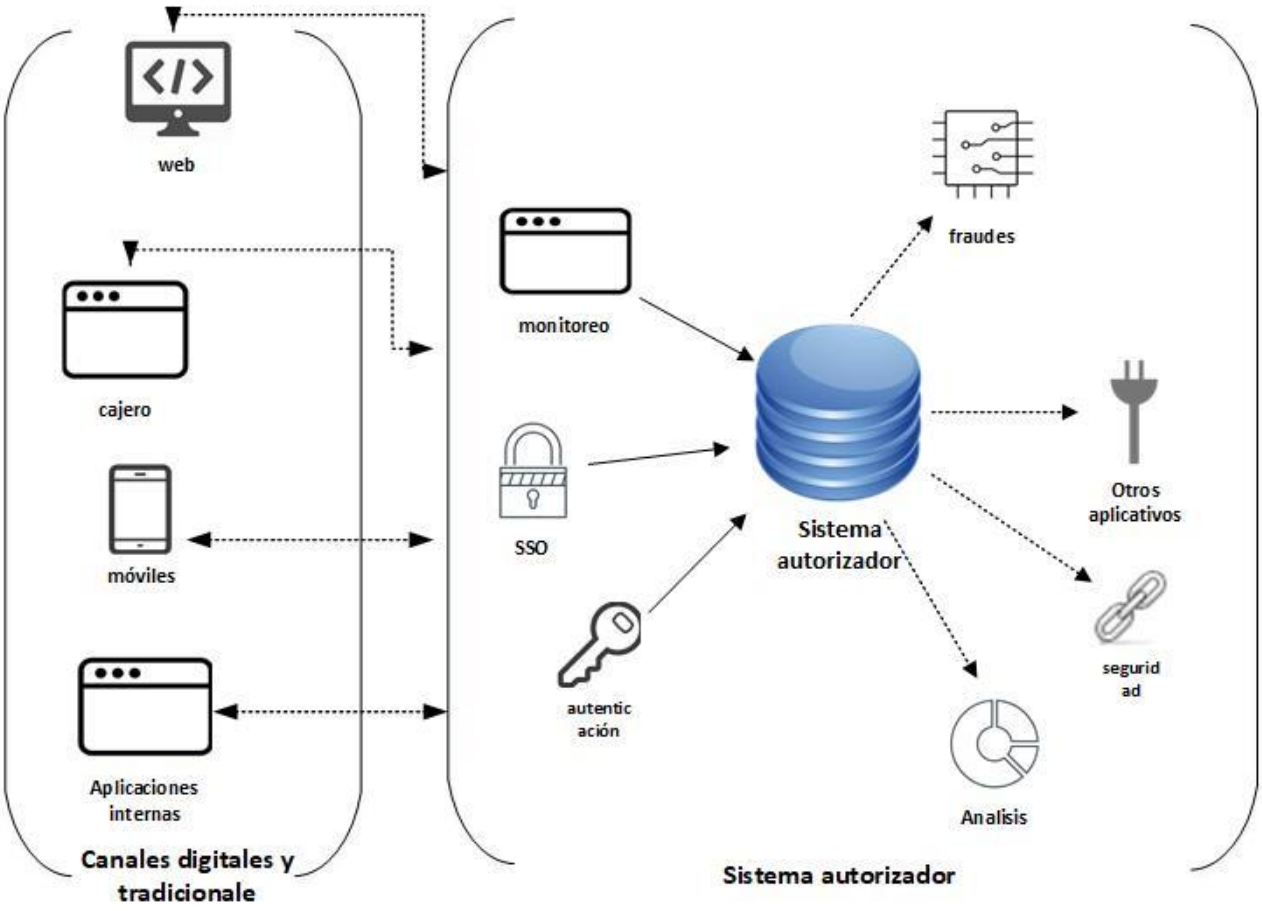


Figura 5.-Sistema autorizador recibiendo transacciones de canales digitales y tradicionales

2.-Transacciones ejecutadas fuera de la ventana de atención eran encoladas de forma paralela para su ejecución junto con transacciones hechas dentro de la ventana, lo anterior provocaba saturación del sistema al ejecutar transacciones en el orden de llegada de diferentes colas que podían contener transacciones dentro de la ventana y fuera de la ventana, de esta manera el orden no era el correcto y el sistema presentaba alta latencia en transacciones que debían tener prioridad sobre otras. Todas las transacciones (dentro de la ventana y fuera de ella) son atendidas por el mismo sistema y para dicho sistema no hay distinción de prioridades entre transacciones traídas de diferentes colas.

Por otro lado, hubo muchos *bugs* que debieron corregirse y promoverse como un cambio a producción. Al ser una institución bancaria cada cambio en el ambiente productivo involucraba pasar por una serie de aprobaciones y justificaciones lo que dificultaba la agilidad de despliegue de cambios.

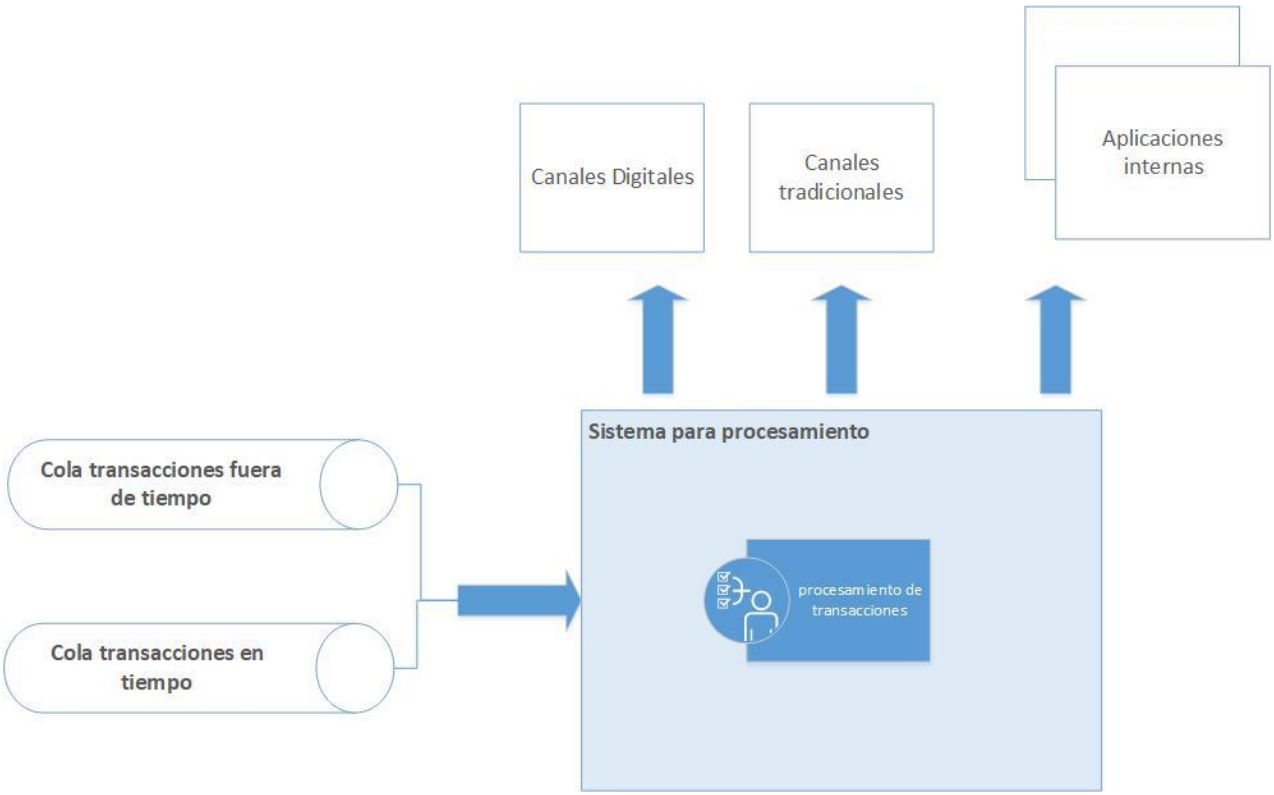


Figura 6.- Colas de transacciones siendo atendidas por el mismo componente provocando latencia en respuesta para aplicaciones cliente.

Solución y participación:

Formé parte del equipo contratado por Citibank para ayudar a Banamex a alinearse a los estándares de arquitectura, mi rol en el proyecto fue de arquitecto de software. En el proyecto, mis principales tareas fueron analizar la arquitectura implantada en Banamex comparándola con la arquitectura de referencia de Citibank, apoyar al área de soporte a producción a identificar las fallas recurrentes y de mayor prioridad para el correcto funcionamiento del sistema y proponer una reestructuración de sistemas para mejorar el rendimiento.

Debido a que los sistemas de Banamex son complejos y muy grandes, trabajé en conjunto con otros arquitectos y nos dividimos los aplicativos de acuerdo con líneas de negocio, yo estuve a cargo de la línea <<canales digitales>> esta línea de negocio comprende banca móvil y banca web.

Tomé diferentes acciones de manera incremental para comenzar a darle estabilidad a los sistemas y en seguida resolver los cuellos de botella descritos, después de estabilizar el sistema, trabajé promoviendo los cambios en el diseño necesarios para alinearse a la arquitectura de Citibank. A continuación, listo de manera puntual y a grandes rasgos las principales tareas que realicé en el proyecto:

1.-Migración de versión de tecnologías: Rediseñé las interfaces de comunicación para alinearse a los estándares de arquitectura cambiando la versión de la plataforma java (de la versión 1.5 a la 1.7), para realizar esta migración realicé un análisis profundo sobre los cambios entre versiones de la plataforma, una vez que hice esto, comparé dichos cambios con la implementación de los componentes usando la versión 1.5 (la versión inicial) para saber el impacto que podría causar esta migración, una vez que identifiqué el impacto, promoví los cambios necesarios, usando el proceso de cambios del banco, para que fueran implementados por el equipo de desarrollo y una vez hechos proseguí a realizar una prueba de despliegue con la aplicación actualizada en un servidor de aplicaciones (IBM *webphere*) en el ambiente de desarrollo con la versión de java 1.7, verifiqué que los aplicativos funcionaran de manera correcta y de la misma forma hice pruebas de integración con aplicaciones dependientes para asegurar la correcta integración y migración de los componentes. Una vez verificado en el ambiente de desarrollo, promoví los cambios entre los diferentes ambientes hasta llegar al ambiente de producción.

2.-Dar solución a los cuellos de botella identificados y descritos anteriormente: La latencia del sistema fue uno de los principales problemas a los que me enfrenté en este proyecto, la solución que propuse fue basada en estándares del banco (Banamex) en conjunto con los lineamientos de arquitectura de Citibank a continuación listo la solución a los dos casos descritos previamente:

a) Múltiples transacciones concurrentes autorizadas por un solo sistema. Para resolver este problema hice pruebas de concepto en el ambiente de desarrollo para replicar la falla y ver específicamente el comportamiento del componente autorizador con alta carga de transacciones, logré descartar fallas propias del procesamiento de transacciones, es decir el procesamiento ya estaba optimizado y no fue la causa del fallo, revisé las orquestaciones de servicios y que se llevaran a cabo solamente los procesos necesarios para las transacciones, pude optimizar en menor medida esta parte de la implementación, sin embargo el rendimiento continuaba siendo un problema.

Me dispuse a hacer un análisis a nivel infraestructura de las instancias del componente autorizador para verificar que el balanceo de carga de transacciones fuese acorde, es decir que la carga no estuviera saturando o cargando de más una instancia más que las otras, logré determinar que las instancias estaba configuradas correctamente revisando a nivel operativo la configuración de la instancia <<maestro>> y las instancias <<esclavo>> y el balanceador también previniendo el *failover,* sin embargo las sesiones estaban configuradas como *sticky-session* esta forma de configuración provocaba que si una instancia dejaba de funcionar todas las sesiones que dicha instancia estuviese atendiendo se perdían y se recurría al proceso *batch* para hacer una doble verificación de transacciones atendidas, así que tuve la determinación de cambiar esta configuración a sesiones persistidas en base de datos no-relacional para tener un acceso rápido a manera de cache, de esta manera, aun cuando una instancia dejaba de funcionar no se perdían las sesiones que se estuviesen atendiendo.

Como ultimo para este caso propuse la utilización de una cola de mensajes asíncrona que funcionó para mantener las transacciones encoladas en mayor medida y como segunda opción

persistirlas como cache en las instancias, esto mejoró el rendimiento notablemente y se alineo a los estándares establecidos por Citibank en cuanto a la seguridad e integridad de la información, una vez que realicé los cambios y los promoví hasta el ambiente de producción el diagrama de arquitectura quedó de la manera siguiente:

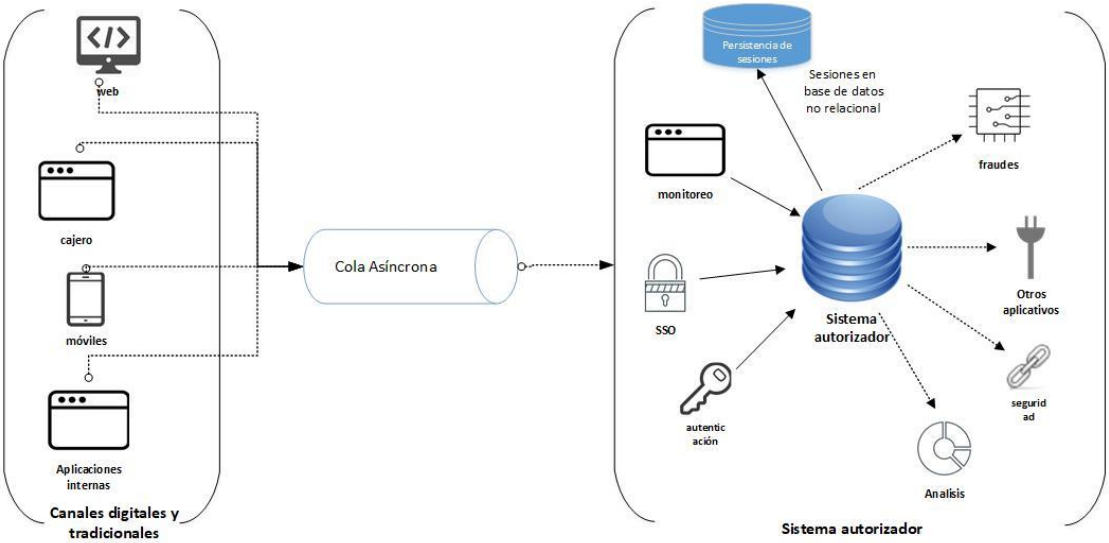


Figura 7.-Sistema autorizador con sesiones persistidas en base de datos no relacional y cola de mensajes asíncrona.

b) Transacciones ejecutadas fuera de la ventana de atención. Para darle solución a esta falla, en primer lugar, analicé el problema original, es decir la mezcla de procesamiento de transacciones por dos diferentes colas sin importar si eran transacciones en tiempo o fuera de tiempo, repliqué el comportamiento con transacciones <<*dummy*>> en el ambiente de desarrollo para determinar porque no se seguía el orden correcto, una vez que pude replicar el escenario logré identificar que el principal problema era la manera en la cual el sistema para procesamiento <<consumía>> los mensajes, es decir la manera en la cual se configuró que el sistema recibiera los mensajes, explicado de una manera sencilla hay dos formas de consumir mensajes por medio de una cola.

Consumir una cola haciendo una conexión directa (*queue*) es decir un consumidor está al pendiente de una o más colas y atenderá los mensajes tan pronto lleguen a dicha cola si ninguna instancia está disponible la cola persiste el mensaje.

La segunda forma de hacer uso de una cola es usando *topics* con esta configuración cada consumidor (en este caso cada instancia del sistema de procesamiento) se subscribe a un *topic* cuando un nuevo mensaje llega se envía una notificación al consumidor que esté disponible, es decir puede ser cualquier consumidor suscrito.

Ahora bien, en el escenario inicial ambas colas estaban configuradas usando *topics* esto provocó que tan pronto llegara un mensaje (transacción) a una de las colas se enviara una notificación al sistema de procesamiento (a una de sus instancias) sin distinción de si era una transacción en tiempo o fuera de tiempo se ilustra mejor en la imagen a continuación:

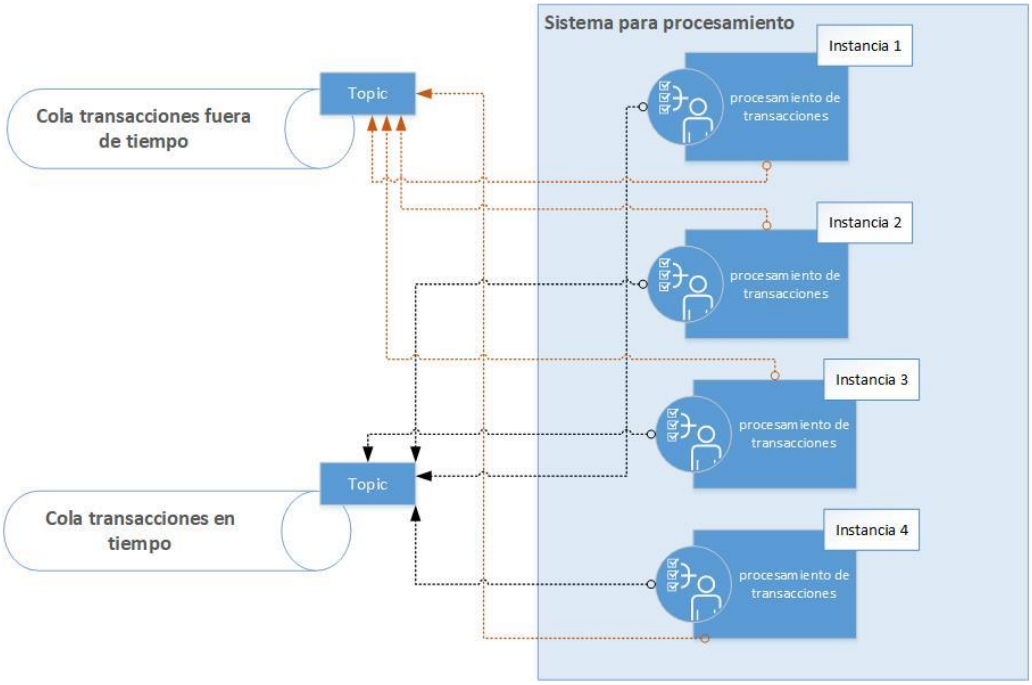


Figura 8.- Colas de transacciones configurada en modo *Topic.*

Como se puede apreciar en el diagrama anterior las 4 instancias estaban suscritas a las dos colas de transacciones para aumentar el rendimiento y cuando un mensaje llegaba era atendido por la instancia que estuviera disponible sin importar si era en tiempo o no, habiendo identificado el problema procedí a analizar la posible solución.

Determiné, después de un análisis profundo, que la solución a la falla radicaba en dos puntos importantes, en primer lugar, cambiar la configuración de la cola de transacciones fuera de tiempo de *topic* a *queue* de esta manera las instancias no recibirían notificaciones si llega una nueva transacción, ahora bien no me fue posible asignar instancias dedicadas únicamente a las transacciones fuera de tiempo debido a que esto reduciría el *throughput* (cantidad de tareas procesadas por unidad de tiempo) por tal motivo diseñé un <<proceso programado>> (*cronjob*) que de manera agendada realizara la conexión de las instancias a la cola de transacciones fuera de tiempo en el momento que sea conveniente y desconectarlas también de la misma manera. El horario en el cual configuré el proceso lo tomé directamente de la ventana de atención establecida por el banco (7pm a 7am), es decir toda transacción ejecutada después de las 7pm y hasta las 7 am del día siguiente entra a la cola de transacciones fuera de tiempo de tal modo que al momento de realizar la conexión por medio de la tarea programada toda transacción después

de las 7pm se ejecutarán únicamente después de procesar las transacciones ya encoladas en la cola de transacciones en tiempo que ya están esperando a ser atendidas. Esto puede apreciarse en el diagrama siguiente:

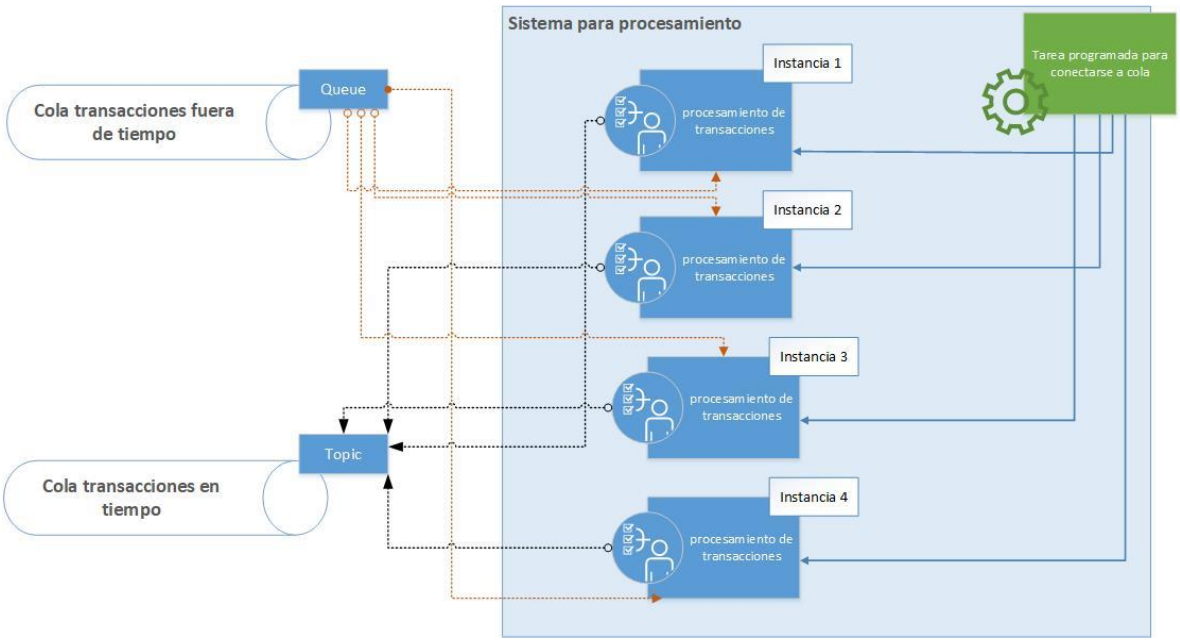


Figura 9.- Colas con dos configuraciones una en *queue* y otra en *topic*, además de la tarea programada para activar o desactivar la conexión a la cola fuera de tiempo.

3.- Alinearse a la arquitectura de Citibank. - Una vez resueltos los principales fallos en los aplicativos, hice una lista de características que diferían de la arquitectura y lineamientos de Citibank y la expuse con los dueños de los aplicativos para trabajar en conjunto y promover los cambios necesarios. Justifiqué cada cambio usando el proceso de administración de cambios (*Change Management)* de ITIL, siguiendo dicho proceso pude llevar a cabo de manera exitosa los cambios necesarios para alinear las aplicaciones de la línea de negocio que tuve asignada. El proceso que seguí para los cambios fue el ya establecido por el banco basado en ITIL, a continuación, lo describo de manera general:

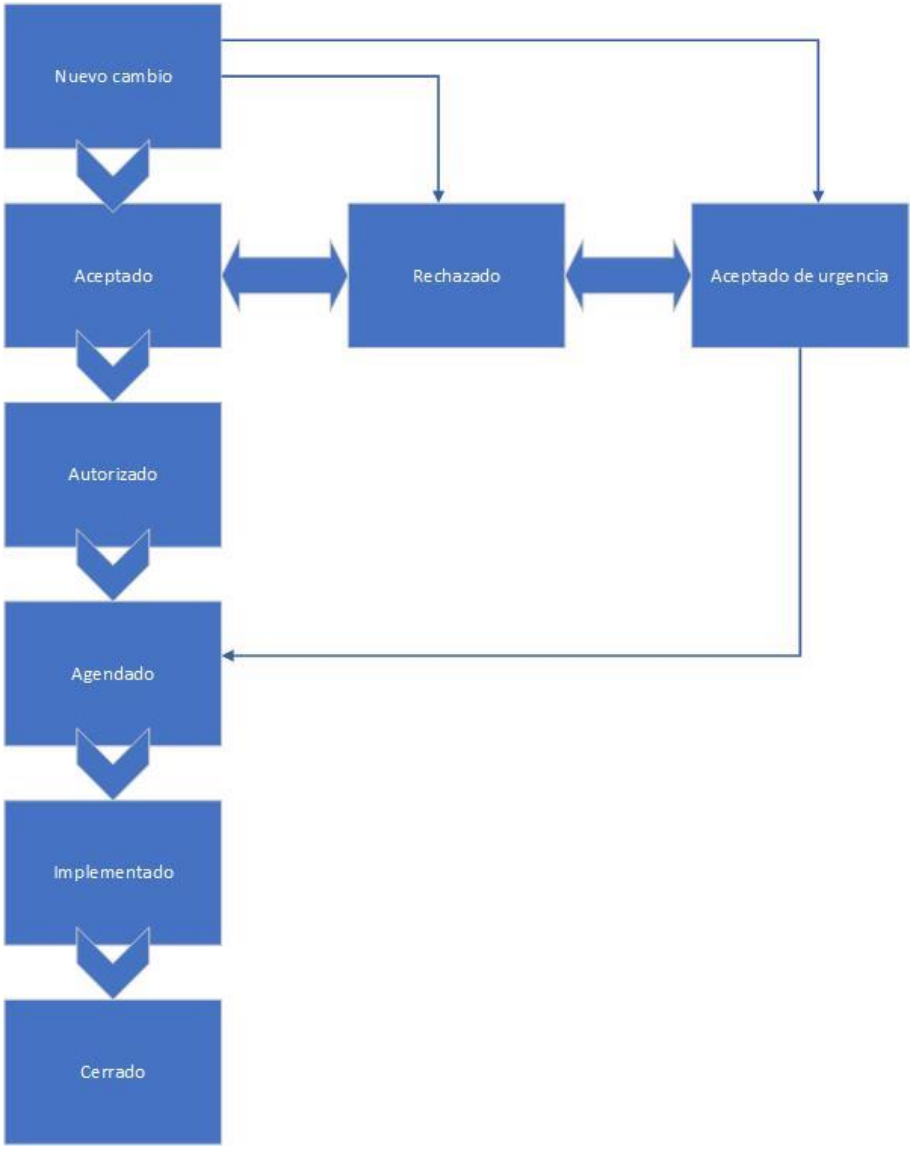


Figura 10.- Proceso de promoción de cambios basado en ITIL propio del banco.

Capítulo 3: Implementación de Plataforma de comercio electrónico

Objetivo

El objetivo de este informe es demostrar mis habilidades aplicadas como Ingeniero en el campo laboral especifico de software. También tiene como meta exponer de forma clara y sintetizada mi participación en el proyecto llamado <<Usablenet Marketplace en Hybris>>.

Marco teórico

Definición

La ingeniería de software se define como la disciplina que esta consiente de todos los aspectos que conlleva el desarrollo de un producto (sistema) informático desde su concepción hasta su uso en un ambiente productivo. Dicho producto involucra una inversión inicial por parte del dueño (cliente) y un proceso establecido de desarrollo por parte del proveedor (equipo de ingenieros de software).

La ingeniería del software surge a partir de la necesidad de llevar a cabo técnicas formales, métodos, soluciones y enfoques aplicados sobre un producto que no es tangible en un mundo físico y puede ser demasiado abstracto para la comprensión de aquellos que no tengan una formación puramente técnica.

De esta manera la ingeniería del software sirve como una disciplina encargada de llevar a buen cause la actividad de desarrollo de un producto (sistema).

Cuando hablamos de todos los aspectos que se ven involucrados en la producción del desarrollo de un software no podemos mencionar sólo aquellos aspectos puramente técnicos, es decir el desarrollo del software involucra muchos otros aspectos más allá del clásico enfoque de programación de componentes, estos aspectos son la administración de proyectos, metodologías de desarrollo, marco de pruebas, análisis del negocio, entrega del producto, infraestructura sobre la cual dicho producto será ejecutado y arquitectura de software.

Ciclo de desarrollo del software

El corazón de la ingeniería del software es la actividad de desarrollar software, para llevar a cabo esta actividad han surgido varias metodologías que pretenden darle trazabilidad al desarrollo y al mismo tiempo un orden establecido para monitorear el avance y posibles problemas, todas las metodologías se han adaptado tomando en cuenta el ciclo de desarrollo del software que tiene lugar en cualquier

desarrollo ya sea a pequeña o gran escala. El ciclo de desarrollo del software define un enfoque bien definido, estricto y tangible en lo que se refiere al desarrollo en todas sus etapas, esto es:

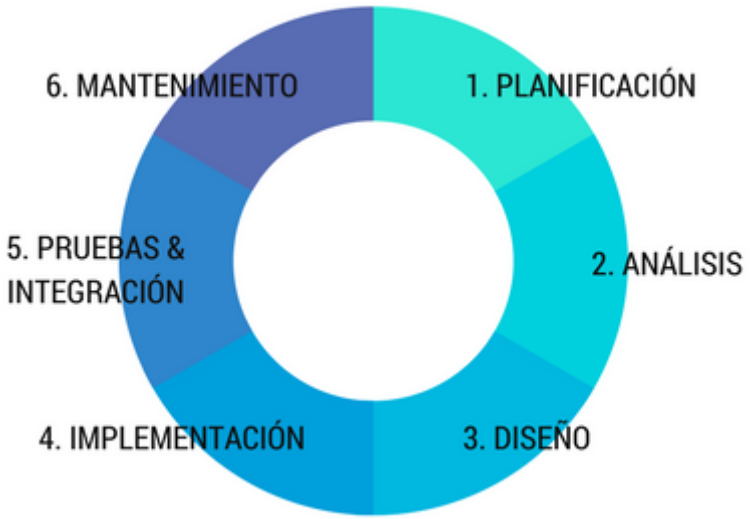


Figura 11.- Ciclo de desarrollo del software.

Procederé a explicar las fases del desarrollo del software y mencionar los principales roles asociados desde el punto de vista del modelo de colaboración: consultoría de software.

Planeación.

Esta fase es el primer paso para lleva a cabo un proyecto exitoso, todas las subsecuentes fases tomarán de base la planeación para acomodar el nivel de exigencia, jornadas laborales, fechas de entrega, presupuesto del cliente en contraste con los requerimientos(propios de la siguiente fase) además en esta fase se lleva a cabo el cálculo de tiempo y personas necesarias para completar el proyecto de manera exitosa, se establecen fechas para las fases subsecuentes y se determina el proceso de comunicación entre los involucrados, puede llevarse a cabo por diferentes canales pero los canales seleccionados deben ser establecidos de manera formal, accesos necesarios a la información sensible, se determinan roles para cada involucrado y su papel dentro del desarrollo del sistema.

Los principales actores de esta fase por parte de la empresa que provee la solución son los Administradores de entrega (*Delivery Manager*), Administradores de proyecto (*Project Manager*) y Administradores de cuenta *(Account Manager*). Por parte del cliente los principales actores son el dueño del producto (*Product Owner*), el administrador del proyecto por lado del cliente y uno o dos directores de operaciones típicamente.

Es esta fase el punto de inflexión que determina si un proyecto de desarrollo de software será ejecutado de forma exitosa o si fallará en el intento, es crucial que la planeación se lleve a cabo de manera correcta.

Análisis

Todos los requerimientos del sistema son obtenidos en esta fase, es decir todo lo que el sistema debe hacer es definido en este lapso. Comúnmente tanto la etapa de análisis como de diseño juntas son llamadas descubrimiento del proyecto (*Project Discovery*).

El análisis es una fase intensa de colaboración entre cliente y proveedor, es en esta fase donde ambas partes llegan a consensos sobre lo que el sistema debe proveer y hacer, como debe responder a ciertas acciones del cliente, como reponerse de errores, las fuentes de datos asociadas y demás datos que sirvan para definir el comportamiento del sistema.

En la etapa de análisis se determinan aquellos colaboradores externos al proveedor y cliente es decir terceras partes (*third party*). Las terceras partes son proveedores de algún servicio que el cliente contrata para resolver alguna necesidad propia del sistema o de su negocio, es decir son otros consultores especialistas en alguna tecnología que han celebrado un contrato de servicios con el cliente y que deberán ser contemplados si el nuevo sistema a desarrollar consume o interacciona con algún servicio de esta <<tercera parte>>.

En la etapa de análisis es una práctica común llevar a cabo sesiones de trabajo, agendadas previamente en la etapa de la planeación, para obtener todos los detalles sobre las expectativas del cliente acerca del sistema a desarrollar, dichas sesiones de trabajo son conducidas por un equipo conformado típicamente por un analista de negocio, un administrador de entrega y un arquitecto de soluciones. Durante estas sesiones los involucrados se esfuerzan en dar respuestas a todas las preguntas del cliente y comenzar a plantear el borrador de un posible diseño y arquitectura del sistema, aunque ciertamente hay una etapa dedicada por completo al diseño, en esta etapa se comienza a vislumbrar lo podría ser el sistema con toda la información recabada.

Dentro de la información más importante a recabar en esta fase están los siguientes puntos clave para la etapa de diseño:

Requerimientos funcionales. – Son todas las especificaciones explicitas que tiene el cliente para el sistema, es decir, es el funcionamiento específico del sistema, cada acción y reacción a la interacción del usuario se define por un requerimiento funcional. Es de suma importancia obtener dichos requerimientos de manera puntual y explicita en la fase de análisis ya que la determinación de si el sistema cumple o no con las expectativas del cliente (a nivel funcional) depende completamente con el cumplimiento de cada requerimiento funcional especificado en esta etapa.

Requerimientos no funcionales. -Son todos los requerimientos impliciticos que requiere un sistema para ser considerado de calidad. Es decir, son las cualidades que debe tener un sistema para completar las tareas (requerimientos funcionales), de la forma en que el usuario tenga la mejor experiencia, dicho de

otro modo, es todo aquello que le permita al sistema darle una mejor experiencia al usuario al realizar sus tareas en el sistema.

Los requerimientos no funcionales son conocidos también como atributos de calidad, su identificación es tarea del arquitecto de soluciones y juegan un papel fundamental en el diseño de la solución del sistema, se considera a un sistema de calidad cuando puede realizar las tareas funcionales proporcionándole al usuario la mejor experiencia, ¿cómo definir si el usuario está teniendo una buena experiencia? es determinado por que tanto el sistema está cumpliendo con los requerimientos funcionales necesarios algunos ejemplos son:

-Seguridad

-Alta disponibilidad

-Rendimiento

-Tolerancia a fallos

-Escalabilidad

-Usabilidad

Restricciones. – En el análisis del sistema o solución es común encontrarnos con restricciones para el diseño que se puede clasificar en los siguientes tipos:

Restricciones del cliente. – Son aquellas restricciones que el cliente plantea desde el principio de los análisis correspondientes al sistema por cuestiones de políticas internas de la organización, restricciones propias del cliente pueden ser: restricciones de uso de proveedores, protocolos, herramientas de colaboración, uso de información, de comunicación, de uso de plataformas libres, etc.

Restricciones de tecnología. - En muchas ocasiones la misma tecnología tiene restricciones de compatibilidad con otras, en otros términos, elegir una tecnología ya sea el cliente o el equipo de análisis determinará alguna restricción de integración que debe ser analizada y enlistada al momento del diseño. En el caso de proyectos de migración las tecnologías que ya están usándose por parte del cliente determinaran estas restricciones.

Restricciones de contrato o alcance. – Las actividades del proyecto será determinadas por un contrato en donde se define claramente el alcance por lo cual en muchas ocasiones se considera una restricción de alcance o contrato la realización de tareas o implementación de requerimientos que no están en el alcance definido en el contrato.

Conjeturas. – En la etapa de análisis de hace un esfuerzo por aclarar cualquier duda y dejar nada incierto o falto de respuesta, sin embargo por la propia naturaleza del sistema o del mismo proyecto es necesario

hacer una lista de conjeturas para acotar el sistema de manera que el alcance no se vea sobrepasado y haya un punto de partida para el diseño, dichas conjeturas pueden ser resueltas y aclaradas en etapas posteriores del proyecto, sin embargo son un recurso necesario para las etapas tempranas de diseño e implementación. Estas conjeturas deben ser planteadas al cliente de tal manera que ellos estén enterados de la falta de certidumbre en ciertos puntos y los supuestos sobre los cuales se comenzará el diseño, por supuesto que es una mala práctica tener demasiadas conjeturas en un análisis, por lo que en la etapa de análisis se hace un esfuerzo exhaustivo por esclarecer en mayor medida todos los inciertos en el proyecto.

Diseño.

Una vez que se ha recabado toda la información necesaria en la etapa de análisis, es decir requerimientos, restricciones, se han hecho conjeturas y se ha definido el alcance es momento de comenzar con el diseño.

En esta fase se lleva a cabo la creación de documentación específica del sistema y se le comienza a dar solución a cada reto que la etapa de análisis detectó, es decir se comienza a diseñar la solución para cumplir con cada requerimiento funcional uno por uno sin falta, tomando en cuenta las restricciones y conjeturas también se comienza a diseñar la solución para cada requerimiento no funcional (atributo de calidad).

El arquitecto de soluciones tiene un papel fundamental en esta etapa, los principales entregables son realizados por la persona que ocupa este rol en el proyecto con ayuda de los demás roles asociados en la etapa de análisis, estamos hablando concretamente de la propuesta de la solución y el documento de arquitectura.

La propuesta de la solución es un documento o presentación en la cual el equipo que participo en el análisis presenta al equipo del cliente la solución propuesta para el sistema, en dicha presentación se contemplan todos los puntos de la fase de análisis y se lleva de la mano a los presentes en la reunión sobre cada aspecto del sistema en un <<alto nivel>> en notación informal es decir en términos de negocio con poca profundidad técnica, esta presentación tiene la intención de dar a conocer a los involucrados como es que el diseño da solución a cada preocupación o requerimiento del usuario final, es algo típico que el equipo del cliente tenga dudas al respecto y se aclaran en dicha reunión, en caso que haya un cambio sumamente significativo al diseño se puede hacer una revisión del análisis y diseñar nuevamente dicho cambio, aunque cabe mencionar que el <<deber ser>> es que todo análisis se lleve a cabo en la etapa correspondiente para no tener un retrabajo.

El documento de arquitectura contempla la especificación de interfaces de comunicación entre componentes, mecanismos de integración con proveedores de servicio externos, nombre y función de cada componente del sistema, diagramas de arquitectura en notación semiformal y formal para explicar y exponer el sistema tanto a roles de negocio como roles técnicos.

Este documento tiene un papel importante durante todo el proyecto porque es donde se explica que debe hacer el sistema y como debe hacerlo con base en toda la información recabada en fases anteriores, el

documento de arquitectura debe ser escrito para dar información a nivel técnico para el equipo de desarrollo y a nivel de negocio para los analistas de negocio, es un artefacto que se utilizará en la incorporación de nuevos desarrolladores y analistas al proyecto, se hará referencia múltiples veces a él pues es el punto de partida para la implementación.

El arquitecto de soluciones es el encargado de diseñar el sistema y contemplar toda la información obtenida en la fase anterior. En muchas ocasiones se llevan a cabo iteraciones entre el diseño y análisis a medida que cada que surge nueva información relevante para el sistema el diseño se puede ver modificado, en la medida de lo posible se procura tener la menor cantidad de iteraciones y esto se logra haciendo desde el principio un buen análisis.

Desarrollo o implementación

El desarrollo del sistema como tal comienza aquí, el equipo de desarrollo que se planteó en la etapa de planeación, es decir número de desarrolladores con el nivel de experiencia requerido, es el protagonista en esta etapa, liderados por un administrador de proyecto, el equipo de desarrollo se encarga de implementar cada requerimiento plasmado en historias o casos de uso de pendiendo de la metodología empleada. El equipo de desarrollo toma como primer punto de referencia la documentación generada en la fase de diseño y a partir de ello comienza a detallar en mayor medida cada requerimiento en piezas más pequeñas suficientemente pequeñas para poder estimar el esfuerzo de implementación.

El arquitecto de soluciones y el administrador del proyecto participan por ejemplo en lidiar con problemas técnicos y estimación de tiempos de desarrollo respectivamente. Dependiendo de la metodología usada (Scrum, RUP, etc.) esta fase puede ser iterativa.

La elección de metodología de implementación es crucial para llevar a buen término la implementación, hoy en día las metodologías que se ajustan al cambio constante en los requerimientos y ofrecen flexibilidad en la constante entrega de avances son las metodologías agiles.

La metodología ágil más usada y la que fue utilizada en el proyecto que expondré más adelante es SCRUM.

Dicha metodología está compuesta por distintos roles y define lineamientos a seguir para llevar a cabo el desarrollo de software a base de iteraciones. Para comprender la metodología en primer lugar es necesario conocer los términos básicos.

Scrum

Maestro de scrum (*Scrum Master*). -es el encargado de asegurar que el resto del equipo no tiene problemas para realizar sus funciones y tareas. El maestro scrum es un guía en el seguimiento de la metodología y ayuda al equipo scrum para garantizar el cumplimiento de objetivos planteados. En otras

palabras, este perfil es un facilitador y ayuda al equipo a mantenerse activo y productivo durante todo el ciclo de desarrollo del producto.

Dueño del producto (*Product owner*). -representa las preferencias del cliente y del resto de interesados no implicados directamente en el proyecto (es decir que no forman parte del equipo de desarrollo). Este perfil es el encargado de definir los objetivos del proyecto y es el que al final de cada iteración dará su visto bueno sobre la entrega del producto además garantiza que el equipo está entregando el resultado esperado.

Equipo scrum (*scrum team*). - Es el equipo encargado de desarrollar y entregar el producto, conformado por un equipo de desarrolladores y aseguradores de calidad. Su trabajo es el más importante en esta etapa pues son los encargados directos de implementar la solución diseñada previamente, su trabajo es lo que al final de cuentas se le entregará al cliente final.

Iteración (*Sprint*). - Es el periodo en el que se realizan todas las acciones pactadas en la reserva de sprint (*Sprint Backlog*), usualmente es un periodo de 2 a 4 semanas máximo y supone entregas parciales para ir probando la calidad el producto final.

Reserva de producto (*Product backlog*). -Se trata de un listado genérico que contiene el conjunto de tareas, los requerimientos y las funcionalidades especificas del proyecto. Cualquier miembro del equipo puede modificar este documento basado en los requerimientos, pero el único con autoridad para designar prioridades es el dueño del producto (*Product Owner)*, quien es el responsable del documento y el que avaluará el producto terminado en cada iteración.

Reserva de sprint (*Sprint backlog*). -este documento también es un listado que contiene las tareas a realizar y quién las desempeñará en cada iteración (*sprint*). Cada tarea tiene un tiempo estimado de realización por el equipo de desarrollo. Al final de cada iteración la reserva de sprint debe quedar vacía suponiendo así una ejecución optima de las tareas programadas para una iteración.

El proceso indica que se deben tener varias iteraciones durante el ciclo de desarrollo en cada iteración se debe crear un entregable funcional y que ha sido probado al final de la iteración, el dueño del producto revisará dicho entregable terminado y dará su retroalimentación respecto a cualquier problema en la funcionalidad que haya detectado.

Conforme se va avanzando en las iteraciones se va incrementado la funcionalidad del producto terminado, cada iteración supone una serie de pasos que deben llevarse a cabo para estar alineados a la metodología scrum, esta serie de pasos son cruciales para llevar a cabo una correcta implementación de la metodología, a continuación, con el siguiente diagrama podremos ver las diferentes fases de una iteración.

Figura 12 Iteración en scrum

Se determina el tiempo de duración de la Iteración pudiendo ser desde 1 a 4 semanas

Una iteración inicia conformando la reserva de iteración desde la reserva del producto, es decir se toman las especificaciones necesarias a implementarse en la iteración actual y se añaden a la reserva de iteración.

Se llevan a cabo asignación de tareas y estimación de tiempo de estas en una reunión inicial ejecutada en cada iteración, dicha reunión es llamada refinamiento de reserva (*grooming).*

Una vez asignadas las tareas y los tiempos estimados por cada tarea, comienza la iteración. Dentro de las tareas diarias en la metodología es tener una junta de 15 min máximo en la cual todos los integrantes del equipo scrum mencionan 3 cosas principales ¿que hice ayer?, ¿qué haré hoy?, ¿algo me detiene? Esta junta es conducida por el maestro scrum y su propósito es tener total visibilidad sobre lo que cada uno está haciendo y surge algún problema que comprometa la fecha de entrega del producto.

Cada iteración tiene una etapa de desarrollo o implementación y una etapa de pruebas unitarias del producto, es decir pruebas de funcionamiento por cada componente de manera aislada y después de manera integral, hablaremos del proceso de pruebas en la siguiente fase.

Una vez terminado y probado el producto se presenta al dueño de producto para su aprobación final o en su caso corrección de errores, una vez aprobado se repite el proceso de iteración de scrum.

Además de la metodología scrum han existido y existen otras metodologías que han surgido como necesidad de un desarrollo más efectivo de productos, tener interacción con el cliente y obtener una rápida entrega se ha convertido hoy en un punto crucial, las diferentes metodologías integran diferentes enfoques teniendo todo un aspecto en común, más calidad en menor tiempo de entrega, algunas de las metodologías más utilizadas en el mercado laboral hoy en día son las siguientes:

Cascada: Esta es una de las metodologías más antiguas y de las primeras referencias en cuanto al desarrollo de software, debe su nombre al enfoque que utiliza para llevar a cabo el desarrollo del software viendo cada paso en la metodología desde una vista general hacia lo más específico de arriba hacia abajo. La metodología en cascada se traduce en una serie de pasos secuenciales que requieren de la fase anterior para realizar las tareas concernientes a la fase actual, las fases involucradas son las siguientes: análisis de las necesidades, el diseño, implementación, pruebas (validación), la integración, y mantenimiento. El inconveniente principal en esta metodología es la falta de retroalimentación del cliente en cada fase, es decir, el cliente verá el producto hasta la fase final y una vez todos los requerimientos implementados, esto se traduce en un complejo proceso para realizar cambios o ajustes a los requerimientos.

Prototipo: Esta metodología se enfoca en un concepto principal, desarrollar modelos del sistema a desarrollar para mostrar al cliente la funcionalidad básica del mismo, esto se lleva a cabo sin mostrar necesariamente todos los componentes y requerimientos del sistema. La principal ventaja de esta metodología es tener la perspectiva del cliente más rápido y poder ajustar el alcance del sistema y requerimientos antes de llevar a una maduración el sistema. Esta metodología favorece la interacción de los diferentes equipos involucrados en el desarrollo del sistema teniendo toda una retroalimentación temprana.

Incremental o ágil: esta metodología se entiende por aquella que se lleva a cabo por medio de ciclos o iteraciones en donde la funcionalidad del sistema se va añadiendo a medida que se avanza, utiliza los principios de la metodología de prototipo añadiendo las fases de análisis y diseño en cada iteración teniendo así la interacción del usuario durante cada fin de iteración y obteniendo una validación que el camino que sigue el desarrollo es correcto.

Espiral: La metodología de espiral también es iterativa y utiliza un enfoque de desarrollo basado en riesgos (*risk-driven development*), esto quiere decir que todo el desarrollo y las fases de cada iteración tiene como objetivo mitigar los riesgos latentes en el desarrollo del sistema. Las etapas de cada iteración son en primer lugar fijar los objetivos después determinar las diferentes alternativas o estrategias que se tienen para cumplir los objetivos fijados en la etapa anterior y como etapa final desarrollar y validar con pruebas la efectividad del desarrollo. Se dice que cada iteración es parte de una espiral por dos parámetros medibles el avance del proyecto en cada iteración (medida angular de la espiral) y el coso que se incrementa por cada nueva iteración en el desarrollo (medida radial de la espiral).

Desarrollo rápido de aplicaciones: Esta metodología se enfoca en la entrega de productos de manera constante al usuario y obtener su retro alimentación en cualquier etapa del desarrollo, es iterativa y

promueve la creación de prototipos para demostrar al cliente el avance del sistema, alienta al uso de marcos de trabajo que ayuden a simplificar el desarrollo maximizar el aprovechamiento del tiempo y minimizar el desarrollo desde cero de cualquier componente, el principal aliciente en esta metodología es el tiempo necesario para poner el sistema en un ambiente productivo (*time to market*). Herramientas para optimizar los tiempos de desarrollo son bienvenidas en esta metodología, marcos de trabajo que reducen el tiempo de desarrollo y herramientas de construcción son típicas en el uso de esta metodología, no se recomienda para sistemas demasiado robustos.

Pruebas e Integración

Esta fase está íntimamente relacionada con la etapa de implementación, van de la mano y muchas veces son consideradas como una sola etapa sin embargo cabe mencionar la importancia de presentar la etapa de pruebas como una tarea de suma importancia por si sola.

En esta fase se llevan a cabo pruebas de tres tipos principales que determina la calidad del entregable que esta por ser usado en un ambiente de usuario, estos tres tipos son los siguientes:

Pruebas unitarias. - Las pruebas unitarias corresponden a la visión de los desarrolladores, es decir que son los que deben elaborarlas y es su responsabilidad asegurarse de hacerlas para cada pieza de código. Esto es así porque cada desarrollador conoce su código, y él es el que tiene la responsabilidad de probar cada operación realizada en su rutina para determinar si es exitosa o no, con este tipo de pruebas se asegura la calidad de cada pieza de forma unitaria y aislada.

Pruebas de integración. – Las pruebas de integración tienen el propósito de evaluar la interacción entre dos o más unidades del software. Este tipo de pruebas verifican que los componentes de la aplicación funcionan correctamente actuando en conjunto. Las pruebas de integración son las que comprobarían que un flujo en la aplicación que usa dos o más piezas de código se complete correctamente.

Cabe mencionar que si se tiene dependencias de sistemas externos también le corresponde a esta prueba verificar su correcta integración con la nueva o nuevas piezas de código.

Este tipo de pruebas son dependientes del entorno en el que se ejecutan. Si llegan a fallar, puede deberse a un cambio en el ambiente de ejecución y no específicamente en la codificación del componente o pieza.

Pruebas funcionales. – Este tipo de pruebas tienen el objetivo de comprobar que el software que se ha desarrollado cumple con la expectativa desde el punto de vista de negocio para la que se había pensado, es decir que estas pruebas están dirigidas a comprobar que un flujo de negocio se pueda completar sin errores, un flujo de negocio es una acción que tiene algún sentido en el negocio del cliente por ejemplo enviar un correo.

En este tipo de pruebas lo que principalmente importa, son las entradas y salidas al software, es decir, si ante una serie de entradas el software devuelve los resultados que nosotros esperábamos.

En estas pruebas no importa la codificación su éxito lo determina que las acciones realizadas por un cliente tengas los resultados esperados, se estudia el desarrollo desde la perspectiva del cliente final y no del desarrollador.

Pruebas de carga. - Las pruebas de carga se refieren a una prueba de calidad correspondiente a la capacidad de transacciones que puede manejar un sistema, es decir se prueba el rendimiento con alta cantidad de peticiones y *throughput* (cantidad de transacciones procesadas por unidad de tiempo).

Con este tipo de pruebas observamos la respuesta de la aplicación ante un escenario en el cual la aplicación reciba el máximo número de transacciones soportado de acuerdo con el diseño y requerimientos no funcionales.

Pruebas de estrés. – Este tipo de prueba no solo se enfoca a la cantidad de transacciones que puede procesar el sistema, sino también a diferentes procesos de diferentes tipos siendo ejecutados en diferentes momentos, es decir se estresa al sistema sometiéndolo a situaciones extremas de picos de carga, procesos largos y cortos. El objetivo de estas pruebas es someter al software a situaciones en las que el sistema <<se caiga>>, es decir deje de funcionar correctamente, la prueba determinará si el sistema, si es capaz de recuperarse o tratar correctamente un error grave.

Pruebas de aceptación. – Las pruebas de aceptación son hechas con el cliente final y son las últimas pruebas del proceso de desarrollo en las cuales el cliente observa que se hayan cumplido todas las expectativas planteadas en el diseño y el sistema sea de la calidad especificada en el documento de arquitectura.

Todos los errores detectados en cada una de las pruebas se clasifican en severidad para ser corregidos de acuerdo con prioridades. Hay un equipo especial encargado de realizar las pruebas y se suele destinar una versión del sistema en un ambiente controlado únicamente para pruebas (ambiente de aseguramiento de calidad). Las pruebas son fundamentales para asegurar la calidad del código y del sistema.

Mantenimiento

Una vez que el sistema es puesto en producción comienza una de las etapas más importantes en el ciclo del software, se inicia la etapa de mantenimiento y aunque suele ser una etapa a la que se le da poca importancia lo cierto es que en esta estaba se terminan de afinar varios detalles del sistema que sólo pueden ser vistos con él funcionando en un ambiente real y no controlado.

La resolución de incidentes es una de las tareas que puede adquirir mucha criticidad si no es abordada de la manera adecuada, varios modelos de resolución de incidentes han aparecido sin embargo el que ha mostrado mayor eficacia es el modelo de ITIL.

ITIL por sus siglas en inglés (*Information Technology Infrastructure Library*), es un detallado conjunto de prácticas acerca de la administración de servicios de TI (Tecnologías de la Información) que se enfoca

principalmente en proveer un marco de trabajo de prácticas de calidad para las organizaciones. Entre los procesos que detalla ITIL en su marco de trabajo se encuentra el proceso de administración de problemas (*Problem Management*) y el proceso de administración de Incidentes (*Incident Management*) que indica que proceso debe seguirse para resolver eficientemente los incidentes que el sistema arroje en tiempo de ejecución.

El proceso de administración de problemas y de incidentes están íntimamente relacionados y aunque pueden ser susceptibles de confusión ambos tienen una diferencia marcada en cuanto al enfoque que le dan a la resolución de lo que ITIL define como un problema en contra un incidente, ITIL define lo siguiente:

Incidente:

ITIL define un incidente como una interrupción no planeada o reducción de calidad de un servicio de TI. El acuerdo de nivel de servicio SLA por sus siglas en inglés (*Service Level Agreement*) define la capacidad de respuesta esperada proporcionada por el proveedor del servicio y esperada por el cliente de dicho servicio.

Los incidentes interrumpen un servicio normal que necesitan ayuda del encargado de proveer dicho servicio de TI para reestablecer el funcionamiento.

Problema:

Un problema es definido por ITIL como la causa de una o más incidentes de forma recurrente. Algunos incidentes son reportados repetidamente tales como baja de servicios de red o bajo performance, en este caso la administración de problemas entra en juego. Un problema está compuesto por uno o varios incidentes que de forma constante provocan una baja en la calidad o servicios de TI provistos.

En términos generales un incidente interrumpe el servicio normal; un problema es una condición identificada a través de una serie de incidentes múltiples con los mismos síntomas. La administración de problemas resuelve la causa raíz del problema; la administración de incidentes restaura los servicios de TI a niveles normales de trabajo.

La resolución de incidentes es la mayor tarea realizada en esta etapa sin embargo también cambios menores son realizados, si un cambio involucra el cambio de la arquitectura del sistema o es considerado como <<cambio mayor>> puede ser sometido a todo el ciclo de software terminando en la liberación de una nueva versión del sistema a producción.

La administración de Problemas entra de forma reactiva cuando uno o varios incidentes son recurrentes en el sistema y la administración de incidentes se enfoca completamente en la resolución de un incidente lo más rápido y eficiente posible.

A continuación, podemos ver de manera general los pasos a seguir para cada uno de los enfoques y la diferencia evidente entre el proceso para resolución de incidentes en comparación con la resolución de

problemas, si bien es cierto que se complementan es importante entender su diferencia para poder elegir el camino a seguir y si es necesario llevar una administración de incidentes o problemas.

Administración de Incidentes	Administración de Problemas
Identificación del incidente Registro de incidentes Categorización de incidentes Priorización de incidentes Diagnóstico inicial Escalado, según sea necesario, al soporte de nivel 2. Resolución de incidentes Cierre de incidentes Comunicación con la comunidad de usuarios a lo largo de la vida del incidente.	Detección de problemas Registro de problemas Categorización de problemas Priorización de problemas Investigación de problemas y diagnóstico. Creando un registro de error conocido Resolución de problemas y cierre. Revisión del problema mayor

Como se puede ver el mantenimiento del sistema es un tema sumamente importante en el ciclo de vida del software y seguir una metodología correcta para la resolución de cada incidente o problema se vuelve crucial para dar un correcto soporte y nivel de servicio al cliente final.

Antecedentes

Evolución del comercio electrónico en México

Hoy en día el comercio electrónico ha cobrado gran relevancia como un canal de ventas en cualquier empresa, hemos visto como poco a poco la mayoría de las empresas que provee servicios y/o productos ha creado o comprado plataformas de comercio electrónico para promocionar sus productos y servicios esta tendencia se ha debido a la creciente necesidad de llegar a la audiencia que recibe y acepta información por medio de canales digitales a través de dispositivos móviles y por medio de computadoras personales.

En México el crecimiento del uso del comercio electrónico fue de un 28.3% en el periodo del 2016 al 2017 con unas ganancias brutas de más de 17 mil millones de dólares según cifras del sitio Asociación de internet (asociaciondeinternet.mx) y se prevé que para finales del 2018 esta cifra aumente notablemente.

Dentro de los datos importantes con respecto a este fenómeno de compras en línea podemos mencionar lo siguiente:

Tres de cada cuatro mexicanos en línea realizaron una compra por medio de comercio electrónico en el 2017.

El aumento de compras en comparación con el año anterior fue impulsado principalmente con el uso de teléfonos inteligentes.

Dentro de los compradores en línea el 54% está conformado por hombres y el 46% por mujeres.

El 86% de los compradores viven en una ciudad de México.

Algunas de las principales ventajas que ofrece el comercio electrónico sobre el comercio tradicional son:

Disponibilidad de comprar las 24 horas al día 365 días al año.

Alcanzar un público incluso internacional sin cambiar mucho el monto de inversión.

Se puede tener un control exacto de las ventas y hacer estadísticas sobre ello.

Es posible reducir al mínimo el personal operativo para la venta de productos.

Es posible manejar promociones, descuentos y trato especial a compradores en comercio electrónico debido a los ahorros en los costos de producción.

Una vez creada la plataforma de comercio electrónico requiere poco mantenimiento.

Llegar al público que está conectado a internet por medio de cualquier dispositivo expande un abanico de posibilidades para vender.

Plataforma de comercio electrónico

La implementación de una plataforma de comercio electrónico es una tarea compleja debido a la cantidad de procesos que se ven involucrados, dentro de los principales procesos que cobran gran relevancia están:

Administración de catálogos: Una de las tareas más socorridas es la capacidad de conformar catálogos de productos para ponerse a la venta en la plataforma, esto es la capacidad de agrupar un conjunto de productos con propiedades similares y administrarlos como unidad, poniéndolos a la venta, aplicando promociones, ordenarlos y navegar a través de ellos.

Administración de productos: El principal elemento del comercio electrónico son los productos, toda plataforma de comercio electrónico debe tener una sección destinada a definir las características de cada producto de forma individual, es decir actualizar las imágenes del producto, su descripción, su precio, tamaño, etc. Cualquier propiedad que le sirva al comprador a decidir si hacer la compra o no.

Administración de usuarios: Es necesario tener una herramienta dentro de la plataforma para gestionar los usuarios que están registrados en la plataforma, para enviarles promociones y programas exclusivos. Además de los usuarios compradores (registrados y anónimos) hay otro tipo de usuarios que se encargan de gestionar la plataforma, son llamados <<usuarios de negocio>> dichos usuarios tienen distintas responsabilidades entre sí, algunas de sus tareas son monitorear la plataforma, ventas, administrar productos y catálogos, así como dar soporte a cualquier inquietud del usuario con respecto a su orden de compra o problemas para completar la venta.

Motor de búsqueda y navegación: Una plataforma electrónica puede contener decenas de miles de productos y realizar una búsqueda de un producto entre toda la gama de variedades puede resultar en una baja en el rendimiento considerable, si a esto aumentamos el factor de concurrencia de usuarios realizando búsquedas de productos por palabras clave, entonces tenemos un factor crítico en la plataforma, la búsqueda de productos.

La navegación entre resultados de búsqueda también se vuelve critico ya que en orden de tener una mejor experiencia de usuario los resultados obtenidos después de una búsqueda suelen ser ordenados de acuerdo con propiedades específicas de dichos productos, es decir, por color, talla, tamaño, etc. La habilidad del usuario para agregar o quitar filtros, para ordenar de acuerdo con criterios se le llama navegación de productos y es crucial para el usuario hoy en día para cualquier plataforma de comercio electrónico.

Calculo de precios e impuestos: Una de las principales ventajas del comercio electrónico es la capacidad de alcanzar clientes potenciales en diferentes latitudes, es por ello por lo que los precios de los productos pueden variar en orden de la moneda del país en el cual se realiza la compra, aunque en algunos casos se

puede estandarizar el precio a una moneda estandarizada como el dólar norteamericano, el cálculo de impuestos siempre será especifico de cada país. La plataforma debe estar preparada para calcular los impuestos de compra de productos para todas las regiones que estén en su alcance definido en un principio.

Procesamiento de pagos: El pago de los productos en la plataforma es uno de los procesos más sensibles dentro del sistema (y de los más complicados también) para realizar transacciones en línea se deben seguir una serie de políticas de seguridad en transacciones electrónicas, desarrollar un módulo de pagos desde el inicio puede conllevar y proyecto muy complejo por sí mismo, usualmente las plataformas utilizan un proveedor (*third party*) ya reconocido y certificado para realizar pagos en línea, la integración con la plataforma varía dependiendo del proveedor, sin embargo el cálculo de impuestos y precio final está a cargo de la plataforma en sí, el módulo de pagos simplemente se encargará de procesar la transacción de manera segura y confidencial.

Cabe mencionar que en un inicio y aún hoy en día la principal preocupación de los clientes potenciales es la seguridad para realizar compras sin verse envueltos en problemas de clonación de tarjetas o cargos no reconocidos, es por ello por lo que este módulo adquiere una especial importancia en la captación de clientes potenciales.

Administración de contenido: Para que un sitio de comercio electrónico tenga éxito, en gran medida se debe a la impresión inicial que provoca con los clientes potenciales, es decir, el sitio tiene que ser atractivo para el cliente, debe lucir fácil de usar y además debe dar la sensación de seguridad al comprador de tal manera que él tenga la disposición de realizar transacciones monetarias en dicho sitio.

Para poder darle una imagen nueva al sitio de acuerdo con la demanda (por ejemplo, por temporadas o estaciones del año) es necesario un manejador de contenidos que dé la oportunidad de agregar o quitar contenido de forma dinámica y sin la necesidad de recurrir al equipo de desarrollo, es decir, el usuario de negocio debe tener la posibilidad de agregar por ejemplo un anuncio para promocionar un nuevo producto, cambiar el color de fondo, personalizar fuentes o la organización de los elementos en pantalla sin la necesidad de ser un experto en la tecnología de la plataforma pero si con conocimientos de su funcionamiento a nivel de usuario.

Procesamiento de envío: Una vez que el usuario comprador ha iniciado su orden y ha realizado el pago en el módulo correspondiente, el siguiente paso es completar su orden con el envío de sus productos. El envío de productos se vuelve una tarea compleja cuando contemplamos los diferentes involucrados en dicha tarea, hablamos del almacén, del servicio de paquetería, del usuario de negocio y por supuesto el usuario comprador.

El módulo de procesamiento de envío debe proporcionar una interfaz en la cual se puede actualizar la información del almacén cuando ha sido recolectado el producto, también debe poder integrarse con el sistema de paquetería para actualizar la información de traslado al usuario. El usuario de negocio debe ser capaz de ver dicha información para poder darle soporte al cliente en caso de que lo necesite. De esta

manera este módulo es el encargado de llevar a buen término una orden de compra, cabe mencionar que en caso de ventas internacionales se debe considerar el alcance de la plataforma definido inicialmente para de esta manera contemplar los impuestos aduanales en caso de haberlos y añadirlos al precio de envío del producto.

Promociones y cupones: Uno de los principales retos del comercio electrónico es ser capaz de proveer una experiencia de usuario optima a los compradores de tal forma que dichos sujetos no pierdan ninguna de las ventajas que tendrían cuando van a una tienda física. De esta manera la plataforma de comercio electrónico debe ser capaz de aplicar descuentos directos a precio de productos, de canjear cupones electrónicos y realizar campañas de promoción que puedan ser acondicionadas a las necesidades del mercado y al plan de estrategia comercial del proveedor de productos tal cual se haría en una tienda física.

El módulo de promociones debe ser dinámico para poder configurar campañas con grado de complejidad medio y pueden ser usadas directamente por el usuario de negocio, dicho de otra manera, debe existir una interfaz en la cual se pueda configurar los detalles de una promoción de manera rápida y que pueda ser usada por los usuarios de negocio.

Procesamiento de ordenes: El objetivo principal de una plataforma electrónica es captar órdenes y completarlas, procesar una orden puede volverse un flujo de negocio complejo dependiendo directamente de cada tarea a realizarse para finalizar una compra, no se trata solamente de elegir productos y pagarlos. Varios módulos trabajan en conjunto para procesar una orden, el módulo de pagos, el módulo de envíos y el módulo de impuestos son los más significativos en este proceso, es por ello por lo que el procesamiento de ordenes en sí engloba el funcionamiento central del sistema y principal objetivo de la plataforma, es decir completar una venta.

Usualmente el procesamiento de ordenes es integrado con otros sistemas de administración de la empresa, tales como un gestor de relación con clientes CRM por sus siglas en inglés (*Customer Relation Management*) o un planificador de recursos empresariales ERP por sus siglas en inglés (*Enterprise Resource Planning*) por lo que se convierte en un punto sensible de integración en la plataforma.

Carro de compra: Aunque el proceso en sí pareciera ser sencillo, lo cierto es que es uno de los módulos más básicos de una plataforma de comercio electrónico y sin el cual no se podría llevar a cabo la compra, el módulo de carrito de compra se encarga de verificar la disponibilidad del producto en el momento que se selecciona, además de seleccionar la variante del producto según la elección del cliente. Hoy en día el carrito de compra también es capaz de brindar una persistencia de datos a los clientes que están registrados y anónimos para de esta manera brindarles una mejor experiencia en su próxima visita.

Administración de inventario: El módulo para la gestión de la existencia de productos es uno de los módulos del corazón del sistema, ya que es utilizado por otros módulos para completar su proceso. El carrito de compra, el envío de productos, el procesamiento de órdenes y el motor de búsqueda y navegación lo usan para determinar operaciones propias. Aunque es un módulo sencillo relativamente,

tiene integración directa con los almacenes, por lo que también es un punto de integración a considerar para el cliente de negocio.

De esta manera podemos ver que la plataforma de comercio electrónico ha sufrido una evolución con el paso del tiempo, convirtiéndose en un complejo sistema de ventas y con el que se pretende dar al usuario una experiencia de ventas integral y todo esto se ha ido logrando con la integración de varias tecnologías, la experiencia de los primeros enfoques y la evolución de los medios digitales.

Definición del problema

Contexto inicial

Grupo Garsa es una compañía mexicana localizada en Monterrey que actualmente tiene 6 líneas de negocio en diferentes campos pasando por la rama alimenticia, energética, inmobiliaria entre otras. Esta compañía tiene la intención de añadir una nueva división de negocio a su corporativo, esta nueva división tiene la misión de crear un mercado premier en línea capaz de proporcionar a los usuarios un sitio web en el cual puedan vender y comprar productos (*Marketplace)* además de administrar su propio catálogo de productos en línea centrándose en las ventas y olvidándose de la difícil administración o creación de un sitio individual para vender dichos productos, todo esto a través de una aplicación móvil y administración basada en la web.

El principal predicamento de la empresa Grupo Garsa fue que no tenía un área especializada en desarrollo de sistemas como tal y mucho menos tenían conocimiento del domino de comercio electrónico, esto provocó que hubiese una total falta de guía de pasos a seguir para llevar a cabo su iniciativa.

En la búsqueda de asesoramiento Grupo Garsa contrató a un nuevo líder de tecnología para dirigir el nuevo proyecto de comercio electrónico, dicha persona tuvo la misión de liderar el proyecto de implementación y ver realizadas las expectativas de su director general en cuanto a la participación de Grupo Garsa en el merco virtual.

Grupo GarSa contrató a la empresa EPAM en busca de consultoría y expertos en el comercio electrónico, EPAM y Grupo Garsa celebraron un contrato de servicios en el cual EPAM proveería a las personas expertas tanto en el dominio de comercio electrónico como en la implementación de una plataforma para realizar el proyecto de Grupo Garsa.

Expectativas del cliente

Se requiere tener una plataforma de comercio electrónico que sea totalmente compatible con aplicaciones móviles en sistema operativo Android y IOS además del tradicional sitio de ventas por medio de internet.

Grupo Garsa no tiene ningún conocimiento del comercio electrónico y no dispone de personal que pueda apoyar en la etapa inicial del proyecto, sin embargo, todo desarrollo y diseño de la aplicación tiene que ser aprobada por la compañía antes de pasar a la fase de implementación.

El diseño tanto del sitio web como de la aplicación móvil está a cargo de un proveedor externo llamado Usablenet con el cual el equipo de EPAM no tiene ninguna comunicación directa, de esta manera el diseño de las pantallas requiere una triangulación de información entre Grupo Garsa, Usablenet y EPAM lo que dificulta en mediana medida la especificación y aclaración de funcionalidad en dichas pantallas.

El diseño en comparación con funcionalidad tiene muchas variantes a considerar, muchas ocasiones lo que se plasma en una pantalla diseñada no es factible a nivel implementación o incrementa en demasía el desarrollo de esta.

Es tarea de EPAM coordinar todos los esfuerzos para llevar a cabo de forma correcta todas las fases de desarrollo de software además de guiar en el proceso a Grupo Garsa en el negocio de comercio electrónico. Grupo Garsa a pesar de no tener expertos en el comercio electrónico tienen conocimiento mediano de tecnología y deseaban estar al tanto de cada decisión hecha en el diseño de la plataforma, así como ir adquiriendo conocimiento del comercio electrónico mediante una transferencia de conocimiento de parte de los expertos de EPAM.

Debido a que es un proyecto nuevo sin ningún desarrollo previo Grupo Garsa está abierto a recomendaciones por parte de EPAM para acelerar el proceso de desarrollo con la mejor calidad del mercado y sin descuidar el presupuesto inicial planteado en las juntas ejecutivas.

El proyecto principal se divide en diferentes alcances y es responsabilidad de EPAM la entrega en tiempo y con la mejor calidad cada producto incremental en cada fase se agregarán nuevas características en cada fase nueva. Las fases y su expectativa son las siguientes:

Fase Inicial. – El equipo de EPAM debe plantear una solución para implementar la plataforma electrónica explicando claramente mediante la documentación pertinente cada decisión tomada y su justificación, así como su relación directa con el objetivo del proyecto. Se presentará la solución tomando en cuenta como referente los más altos estándares de calidad de empresas líderes en el ramo. Debido a la inexperiencia de Grupo Garsa en el negocio de comercio electrónico, tanto los requerimientos funcionales y no funcionales serán en un inicio propuestos por el equipo experto de EPAM, dejando margen de modificación una vez reciban la retroalimentación de Grupo Garsa.

Todo el diseño visual de interfaz de usuario diseñado por Usablenet es considerado en la propuesta de solución inicial y cualquier inconveniente en su implementación deberá ser indicado oportunamente en la fase de revisión de dichas interfaces.

Fase de implementación primer alcance. – En la primera fase de implementación se contemplará el enfoque de la plataforma con interfaz de usuario únicamente con dispositivos móviles, considerando únicamente Android y IOS como plataforma de desarrollo de las aplicaciones, EPAM como expertos en el

desarrollo de comercio electrónico proveerá el desarrollo de dichas aplicaciones. La plataforma de comercio electrónico tendrá capacidad de completar una compra desde la selección del producto hasta el proceso de envío del producto, todas las integraciones necesarias para completar el flujo corresponderán a las expuestas en la fase inicial. Esto es: sistema de pagos, gestión de catálogos, navegación y búsqueda de productos, cálculo de impuestos, carrito de compra, administración de productos y administración de usuarios.

Cualquier licencia requerida para la integración y su contratación queda a cargo de Grupo Garsa.

La plataforma tendrá la capacidad de darle al usuario la alternativa de crear una tienda virtual para vender sus propios productos a través del comercio electrónico y de esta manera cumplir el principal objetivo de negocio (*Marketplace*).

Los diseños de las interfaces de usuario necesarios serán proveídos por Usablenet en su versión final con previa revisión con el equipo de EPAM.

En este primer alcance solo se consideran transacciones y envíos en las zonas de México y Estados Unidos esto a tomar en cuenta para el cálculo de impuestos y envío de los productos.

Fase de implementación segundo alcance. – En la segunda fase la plataforma será visible desde una interfaz web siendo compatible con los navegadores IE (Internet Explorer), Chrome, Mozilla, Opera y Safari. El diseño de las interfaces de igual manera corresponde a Usablenet y su implementación queda a cargo de EPAM, en esta fase se añade el módulo de administración de contenidos, proporcionándole así al usuario la capacidad de administrar las páginas de su sitio por medio de la plataforma electrónica.

Todos los servicios provistos para las aplicaciones móviles deberán funcionar exactamente igual para el entorno web y haber compatibilidad e integridad exacta de la información de los productos entre los dos distintos canales.

Se añadirán los procesos adicionales de promociones y cupones pudiéndose administrar únicamente desde la interfaz de usuario de negocio en el entorno web, de igual forma se añadirá un módulo de reportes de la actividad y estadísticas del uso del sitio web.

Se añadirá un módulo para la gestión de productos con alta prioridad, es decir productos que aparecerán en primer lugar en las búsquedas pudiéndose configurar dicha prioridad en una interfaz de usuario de negocio provista por la plataforma.

La solución debe ser diseñada añadiendo toda la información necesaria para su implementación, documentando la arquitectura, interfaces de integración y sistemas externos que serán integrados para proporcionar toda la funcionalidad necesaria.

Fase de cierre. - Se proporcionarán manuales de usuario por parte de EPAM para cada uno de los procesos administrativos a realizar con el usuario de negocio.

Se añadirá a la documentación final una lista de modificaciones necesarias para escalar el sistema de tal forma que soporte un mayor número de usuarios en un futuro, dejando claramente explicada cada interfaz de integración en el sistema para su extensión en futuros proyectos.

Se tendrá una sesión de cierre en donde se dará revisión a los entregables antes descritos para dar cierre al proyecto.

Análisis y metodología empleada

Este proyecto tiene la peculiaridad de que el cliente no tiene ningún conocimiento del negocio del comercio electrónico, por lo que nuestro papel como expertos y el mío en particular fue el de guiarlo a través de los procesos básicos, siendo así, cómo arquitecto de la solución llevé a cabo el análisis del sistema usando la metodología a continuación descrita:

Metodología SEI (*Software Engineering Institute*)

Para llevar a cabo el diseño de la solución del sistema utilicé la metodología de análisis de requerimientos publicada por el SEI por sus siglas en inglés (*Software Engineering Institute),* esta metodología se encuentra expuesta en el libro *Software Architecture in Practice* que rige su estructura de pasos tomando en cuenta principalmente los requerimientos funcionales del sistema y los atributos de calidad (requerimientos no funcionales), la metodología indica el proceso a seguir para crear una propuesta de solución arquitectónica a un problema de software dado por un cliente, el principal agente de cambio es la continua retroalimentación del cliente y su impacto en la propuesta inicial de solución, dando así un proceso iterativo que busca tener la total satisfacción del cliente en la solución final del sistema. La metodología del SEI sigue los pasos descritos en el siguiente diagrama:

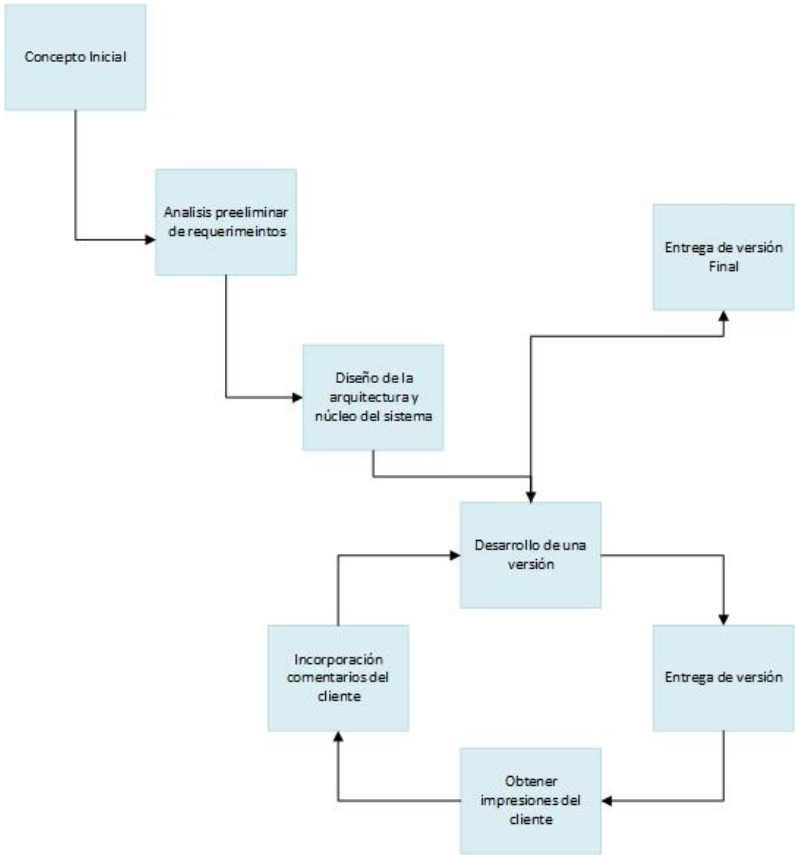


Figura 13.- Metodología del SEI para la propuesta de solución arquitectónica.

Las distintas fases de la metodología son descritas a continuación:

Concepto Inicial. – Es la concepción del proyecto en sí, en otras palabras es la idea principal del cliente o el problema al cual se quiere mitigar con la solución propuesta, en esta fase el cliente expone sus expectativas e idea general de la solución requerida, opcionalmente puede hacer sugerencias a la solución de acuerdo a su experiencia en el negocio o en tecnología, dichas sugerencias serán consideradas sin embargo es tarea del equipo de arquitectura definir si son útiles o no para la solución y propuesta final.

En el caso específico de Grupo Garsa el cliente no tenía conocimiento del negocio, su referencia fueron las plataformas líderes en el comercio electrónico como Amazon y mercado libre, por lo cual dichos conceptos fueron sumados a la experiencia en implementaciones pasadas del equipo de EPAM para concebir de esta forma el concepto inicial de la solución.

Análisis preliminar de requerimientos. – En esta etapa se lleva a cabo una serie de reuniones con el cliente a modo de sesiones de trabajo para obtener los requerimientos funcionales y atributos de calidad del sistema, la metodología llama a estas sesiones de trabajo de cualidades QAW según sus siglas en inglés (*Quality Attributes Workshop*) es importante realizar esta tarea de manera exhaustiva para poder tener una visión amplia de lo que el sistema debe hacer y cómo debe hacerlo.

Como primer punto la metodología indica realizar una serie de entrevistas para determinar que roles y que personas deben estar presentes en la reunión, es importante este paso ya que el éxito o fracaso dependerá de que se encuentren las personas correctas y sean capaces de responder las preguntas indicadas, así como dar sus comentarios e impresiones sobre los conceptos ahí citados.

El segundo paso y muy importante es hacer un listado de requerimientos tanto funcionales como no funcionales, una vez obtenidos es necesario hacer una priorización de cada uno de ellos basándose en los comentarios del cliente y sus representantes en la sesión de trabajo. Esta priorización permite al arquitecto de solución identificar un elemento de suma importancia para el diseño del sistema.

Los requerimientos significativos de arquitectura o ASR's por sus siglas en inglés (*Architecture Significant Requirements)* son obtenidos a partir del listado de requerimientos previamente dicho y priorizado, estos ASRs constituyen un papel fundamental en el diseño de la solución pues, en términos sencillos, son los atributos de calidad o funcionales que causan un impacto mayor en la arquitectura, es decir son aquellos requerimientos que pueden cambiar de manera sustancial el diseño de la solución un ejemplo claro podría ser que el sistema deba guardar cada operación en la base de datos por concepto de auditoria, es un requerimiento funcional que cambiaría el diseño de forma significativa pues la cantidad de transacciones a la base de datos se incrementa exponencialmente y el rendimiento se puede ver degradado en gran medida.

Una de las técnicas propuesta por la metodología del SEI para la identificación de los ASR's es el uso de escenarios en los cuales se exponga una fase del sistema y su posible fallo ante una estimulación dada, mediante esta técnica de escenarios se puede clarificar con ayuda del cliente si un requerimiento es en

efecto significativo para el diseño de la arquitectura. En el ejemplo a continuación muestro un ejemplo de un escenario ante una operación común en la plataforma.

Descripción del escenario: Agregar un producto al carrito de compra es una operación sencilla sin embargo si esta operación no se puede ejecutar de forma correcta se convierte en una falla critica ya que impide que el flujo de compra se complete correctamente, el escenario involucra realizar la operación de adición de un producto bajo un ambiente altamente concurrente.

Fuente. - El agente del cual proviene el estímulo. En este caso el usuario.

Estimulo. -La acción que provoca una reacción en el sistema. En este caso añadir un producto al carrito.

Ambiente. – El sistema bajo alguna situación específica, es decir si está trabajando en condiciones normales o algún momento en específico. En este caso con 1000 usuarios realizando la misma operación al mismo tiempo.

Respuesta. – La salida del sistema después de aplicar el estímulo al sistema en determinado ambiente. En este caso el producto es añadido al carrito de manera exitosa.

Medida. – Es la métrica que se utiliza para determinar la calidad esperada del sistema ante el estímulo dado.

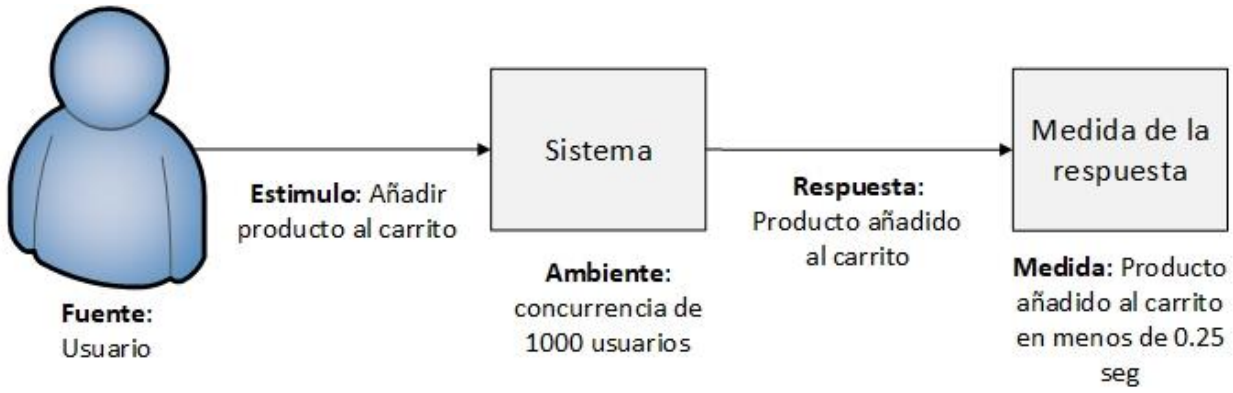


Figura 14.- Escenario para la identificación de ASR, añadir un producto al carrito en concurrencia.

Cabe mencionar un aspecto fundamental en esta etapa y eso es que nunca se debe comenzar a diseñar en las sesiones de trabajo QAW, ya que esto es una mala práctica, es labor del arquitecto conducir la sesión de manera que no se pierda el principal objetivo de esta, captar los requerimientos y determinar los ASRs del sistema.

Diseño de la arquitectura y núcleo del sistema. – En esta etapa tiene lugar el diseño de la solución del sistema. Se toman en cuenta los requerimientos funcionales y no funcionales obtenidos en la etapa anterior poniendo especial cuidado en los ASR's.

Proceso de diseño

Para el diseño, la metodología especifica claramente los elementos que deben tomarse en cuenta para el diseño. Las técnicas usadas para el diseño y herramientas utilizadas para los diagramas dependen en gran medida del criterio del arquitecto en el proyecto, sin embargo, la metodología SEI propone las siguientes prácticas para tener éxito en el diseño de la arquitectura.

I. Patrones de diseño arquitectónicos. – Definidos como soluciones probadas a problemas recurrentes, los patrones de diseño arquitectónicos son de mucha ayuda al momento de diseñar la solución de un sistema, es necesario echar mano de aquellos patrones que hagan sentido para el diseño de acuerdo con los requerimientos. Un patrón de diseño bien aplicado puede solventar más de un requerimiento y atributo de calidad al mismo tiempo.

II. Técnicas conocidas para solventar atributos de calidad. – Mas allá de los patrones de diseño tienen lugar las técnicas conocidas, no son consideradas un patrón de diseño pues están enfocadas en los especifico y no pueden ser aplicadas en escenarios específicos además que pueden quedar obsoletas por el paso del tiempo al contrario de los patrones de diseño. Teniendo esto en cuenta existe un numero grande de técnicas para proveer al sistema de los atributos de calidad necesarios.

   Es labor del Arquitecto tener el conocimiento de dichas técnicas y aplicarlas cuando el sistema lo necesite, la metodología del SEI propone una serie de técnicas por su cuenta para seguridad, rendimiento, alta disponibilidad, mantenimiento, etc.

III. Arquitecturas de referencia. – Una de las principales ideas que debe tener en claro un arquitecto al momento de diseñar es que en un 80% de los casos el problema al que intenta dar solución es muy parecido a una situación común en la línea de negocio en la que esta trabajado, es por ello que las arquitecturas de referencia adquieren gran importancia, si bien es cierto que cada proyecto siempre tendrá peculiaridades, también es verdad que tendrán más similitudes ellos es por eso que distintas autoridades en sistemas computacionales publican arquitecturas de referencia para ser aplicadas en un momento dado. Compañías como Microsoft, SEI, Oracle e IBM son algunos de los líderes que proponen arquitecturas de referencia para problemas dados, así como para microservicios, SOA (*Service Oriented Architecture*), Cliente- Servidor, etc.

   El negocio de comercio electrónico tiene varias arquitecturas de referencia dependiendo de la plataforma y requerimientos de esta.

IV. Pruebas de concepto. – Durante el diseño de la arquitectura es común encontrarnos en situaciones en las cuales no tenemos certeza de la tecnología, es decir no tenemos experiencia o conocimiento en alguna integración de dos o más tecnologías o plataformas. Las pruebas de concepto son utilizadas para disminuir al mínimo las suposiciones de integración o la falta de certeza en compatibilidad, se realizan antes de proponer la solución final pues de su resultado dependerá el que la tecnología sea realmente considerada en el diseño de la arquitectura.

Las pruebas de concepto deben ser bien documentadas en especial los resultados esperados en contraste con los resultados obtenidos de estas.

Es responsabilidad del arquitecto realizar dichas pruebas de concepto, aunque puede ayudarse con un o dos desarrolladores expertos en la tecnología a probar si es necesario.

V. Experiencia previa. – Es un hecho que el diseño de la arquitectura es influenciado por todos los factores que envuelven al proyecto en sí, sin embargo, hay un factor importante que también influencia el diseño y es la experiencia previa del arquitecto en la implementación de diversas tecnologías. Es lógico pensar que si se ha tenido buena experiencia en integrar alguna tecnología se pueda optar por incluirla en el diseño pues su eficacia ha sido comprobada, sin embargo cabe mencionar que la objetividad no debe perderse en este punto, es común ver en arquitectos novatos tratar de forzar la solución a una tecnología conocida por el temor a la incertidumbre de algo nuevo, en este sentido se debe apelar a las pruebas de concepto y nunca olvidar el principal conductor en el diseño, los requerimientos funcionales y atributos de calidad que el diseño debe lograr para considerarse una solución apropiada.

Documentado el diseño:

Una vez contemplados todos los puntos anteriores se concluye el diseño en la documentación apropiada.

El SEI propone distintos tipos de notaciones y una metodología para documentar una arquitectura, dicha metodología hace un perfecto sentido después del análisis hecho y antepone los requerimientos y sobre todo la audiencia del documento antes de la elección de algún diagrama.

El primer paso es elegir la notación adecuada para los diagramas que se elegirán posteriormente, la elección de la notación depende directamente de la audiencia a la cual irá dirigida la documentación, en un proyecto típico el documento de arquitectura está dirigido al equipo de desarrollo y analistas de negocio por lo que la audiencia serán desarrolladores y perfiles no técnicos con conocimientos del negocio.

En una presentación de arquitectura la audiencia típicamente son perfiles no técnicos con influencia alta sobre el proyecto como administradores de proyecto, gerentes o directores.

El SEI define tres tipos de notaciones:

Informal. – Esta notación es la más popular y muy usada en arquitectura, no sigue una notación de figuras estructuradas y permite el uso libre de imágenes y representación de componentes. Las figuras típicas son cajas y flechas, esta notación es útil para perfiles no técnicos que requieren conocer la solución a un nivel alto sin entrar en detalles técnicos.

El punto más importante que considerar en esta notación es el uso de llaves que indique claramente que significa cada flecha o figura en el diagrama, también se pueden usar tipos de colores para diferenciar entre componentes de diferentes capas o plataformas siempre y cuando

se indique claramente en las llaves dichas. Algunos perfiles que pueden ser receptivos a este tipo de notación son ejecutivos de negocio, analistas de negocio y administradores de proyecto.

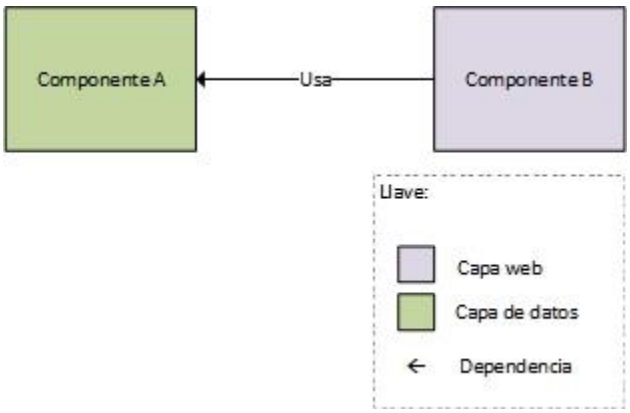


Figura 15.- Ejemplo de notación informal

Semiformal. – Es la más utilizada en el nivel de desarrollo, este tipo de notación utiliza estructuras con una notación estandarizada, hay figuras con un significado definido claramente por su creador y los diagramas se expresan en una notación que utiliza elementos gráficos y reglas de construcción.

Esta notación es muy útil para explicar al equipo de desarrollo el diseño del sistema desde una perspectiva más técnica y en términos que asocien de inmediato con la implementación técnica algunos ejemplos claro es el lenguaje UML, entidad-relación, notación de administración de procesos de negocio o BPM por sus siglas en inglés (*Business Process Management Notation*).

La audiencia para este tipo de notación son desarrolladores, analistas de negocio y aseguradores de calidad.

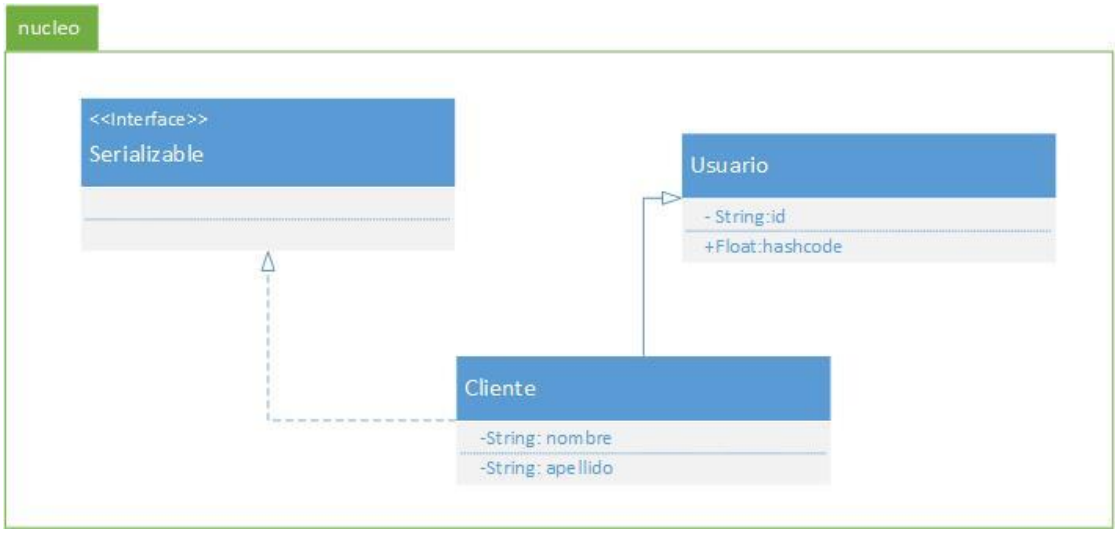


Figura 16.- Ejemplo de diagrama de clases notación semiformal UML.

Formal. – La notación formal es la menos utilizada en la arquitectura de software y son preferidas en la arquitectura de hardware por su naturaleza. Las vistas son descriptas en una notación que tiene una semántica precisa (generalmente basada en matemáticas).

Es posible un análisis formal tanto sintáctico como semántico, es decir la notación es permeable a codificación en software o comandos en hardware de manera que la notación formal puede considerarse un lenguaje descriptivo de configuración de hardware o de generación de código en software.

Son conocidas como Lenguajes de descripción de arquitectura ADLs por sus siglas en inglés (*Architecture Description Languages*), proveen un vocabulario gráfico y una semántica subyacente para representar arquitecturas y son soportadas a través de herramientas asociadas como Lombardi de IBM, PEGA, etc.

La audiencia para este tipo de notaciones usualmente son implementadores de arquitectura de infraestructura, el equipo de desarrollo no es receptivo a esta notación y requiere conocimiento completa de la notación utilizada, este tipo de diagramas son vistos en mayor medida en documentación de administradores de sistemas.

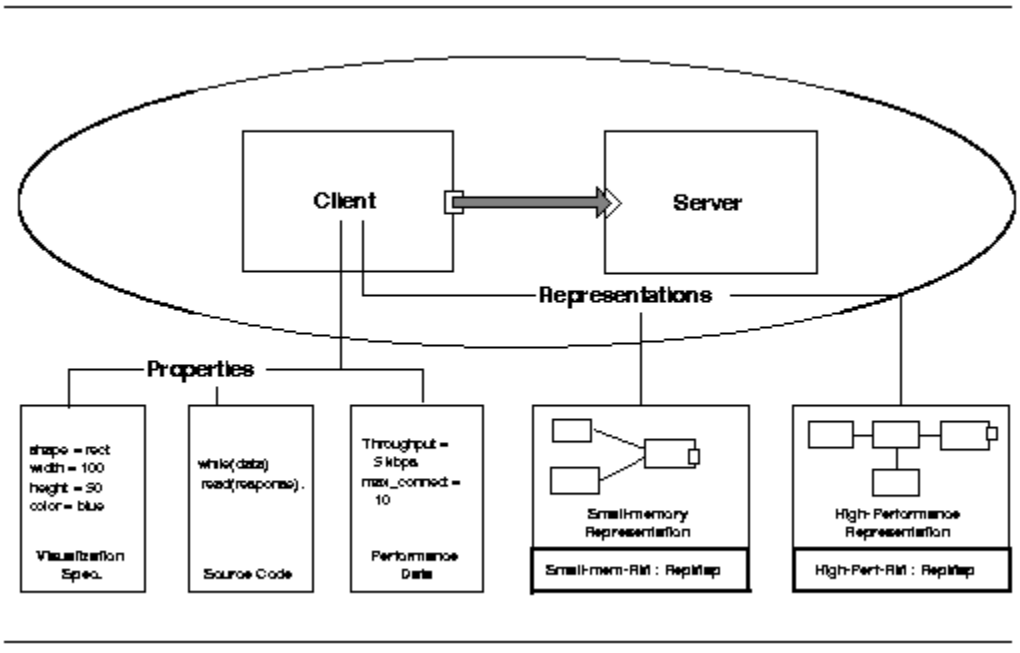


Figura 17.- Ejemplo de notación formal en la herramienta ACME considerada ADL

La elección de una notación clara determinará en gran medida la aceptación o rechazo del diseño de arquitectura, es verdad que el diseño es de suma importancia en la arquitectura, sin embargo, si el arquitecto no es capaz de explicar ese diseño a todos los roles involucrados y tener su aceptación, entonces el diseño por mejor que este pensado, no será aprobado por los roles involucrados.

El siguiente paso en la documentación de la solución es la selección de vistas y diagramas que deben ser presentados a los involucrados de los distintos niveles (desde desarrollador hasta director de tecnología del cliente).

Vistas en la documentación:

La parte más importante en la documentación es la elección de vistas. Un diseño de arquitectura de software es una entidad demasiado compleja para poder ser representada en una sola vista o en un solo diagrama es por ello por lo que para exponer apropiadamente un diseño es necesario documentar varias vistas del sistema.

La elección de visitas es una parte crucial, ¿Qué vistas se deben documentar en el sistema? La respuesta a esta pregunta depende exclusivamente de las metas que tengas en la documentación, es decir que atributos del sistema se quieren resaltar y a qué audiencia se quiere llegar con dicha vista (desarrolladores, gerentes, analistas de negocio). Las vistas que se deben documentar dependen directamente del uso que se le quiere dar a dicha perspectiva del sistema, diferentes vistas deberán resaltar diferentes aspectos del sistema, ya sea una integración crucial del sistema, un proceso de suma importancia para el negocio o una perspectiva de especial interés para alguno de los involucrados en el sistema.

Es importante en esta fase elegir ir la notación correcta de acuerdo con la vista que se quiera exponer tomando en cuenta audiencia y congruencia con el sistema, es un error común tratar de exponer todo el detalle de la arquitectura en una sola vista.

Una vez seleccionada la notación y vistas del sistema el último paso es conformar el documento de arquitectura que será expuesto, dicho documento será un referente del diseño y punto de partida para el desarrollo del sistema, además será uno de los artefactos que deben ser leídos primero cuando un nuevo integrante del equipo se una al proyecto.

Iteración y refinamiento de arquitectura:

Una vez terminada la primera versión de la arquitectura se lleva a cabo una serie de iteraciones para hacer un refinamiento del diseño y obtener comentarios del cliente, formalmente el SEI propone estas etapas en la iteración:

I. Desarrollo de una versión. – es la primera versión del documento de arquitectura.
II. Entrega de versión. – Se entrega el documento de arquitectura al cliente y también es buena práctica exponer el diseño en una presentación, dependiendo de la audiencia se puede hacer una

presentación en alguna herramienta para exponer los detalles en más alto nivel, claro que basándose en el Documento formal creado anteriormente.

III. Obtener impresiones del cliente. – Es crucial obtener las impresiones que el cliente tenga sobre la arquitectura, con base en sus opiniones fundamentadas se pueden llevar a cabo modificaciones en el diseño. Hay que tomar con mesura las opiniones y asegurarse que se está entendiendo completamente el diseño antes de pensar en modificarlo.

IV. Incorporación comentarios cliente. -Los comentarios del cliente que tengan impacto en la arquitectura serán incorporados y tomados en cuenta como consideraciones o cambio en las vistas.

Pueden llevarse a cabo varias iteraciones hasta que todas las partes estén completamente convencidas de la solución presentada, una vez que se ha llegado aún acuerdo en cuanto a todos los aspectos de la solución se entrega la versión final al cliente y se coloca dicha versión disponible para el equipo de desarrollo y sea usada para comenzar la implementación. Para la implementación del sistema se usó Scrum, metodología expuesta anteriormente en el marco teórico.

Análisis preliminar del sistema.

Personas de interés principales en el sistema

Para llevar a cabo el análisis al sistema basado en la metodología descrita es necesario conocer a las personas involucradas en el sistema y principal audiencia a continuación enlisto los principales involucrados en el proyecto:

Vadim Vaiman	Gerente de cuenta	vadim_vaiman@epam.com
Omar Arana	Gerente de entrega	omar_arana@epam.com
Alan Pozos	Gerente de proyecto	Alan_pozos@epam.com
Juan Carlos Villegas	Arquitecto de Solución	carlos_villegas@epam.com
Ricardo De La Fuente	Líder técnico hybris	ricardo_de_la_fuente@epam.com
Eduard Beleninik	Líder técnico móvil	eduar_beleninik@epam.com

Requerimientos Funcionales

Los requerimientos a continuación descritos los obtuve haciendo reuniones constantes con el equipo del cliente, tuve sesiones de trabajó en las cuales pude identificar lo siguiente:

- Habrá dos tipos de usuarios, a uno se le conocerá como <<comprador>> y el otro como <<vendedor>>, el comprador es el usuario final capaz de ver los productos publicados en la plataforma y comprar productos, el vendedor es un usuario que utiliza la plataforma para publicar y vender productos. Un solo usuario puede ser vendedor y comprador al mismo tiempo.
- El comprador debe poder visualizar todos los productos con una oferta disponible configurada en el sitio web de ventas (*Marketplace*)
- El sistema tendrá la funcionalidad <<*MyStore*>> cuya finalidad es la de persistir una lista de productos seleccionados por el usuario de manera similar a una <<*lista de deseos*>> típica en el comercio electrónico.
- El comprador debe tener la capacidad de crear una instancia de funcionalidad <<*MyStore*>>, el sistema debe proveer la interfaz de usuario requerida para la creación de dicho componente.
- El comprador debe poder comprar un producto a través de su dispositivo móvil Android o IOS
- El comprador debe poder recibir notificaciones en su móvil activadas por eventos del Sistema
- El comprador debe ser capaz de crear una instancia de <<*MyStore*>> incluso si no está registrado en el sistema.
- El sistema tendrá la capacidad de gestionar <<*tiendas virtuales*>> para publicar productos a compradores.
- El vendedor debe poder crear una <<tienda virtual>> usando la plataforma y crear ofertas de productos que estarán disponibles para el comprador.
- El vendedor debe poder crear nuevos productos en el sistema y agregarlos a la <<tienda virtual>>.
- El vendedor debe poder revisar cuántas órdenes tiene su propia <<tienda virtual>> y administrar en consecuencia.
- La experiencia de usuario no debe ser afectada por el uso de dispositivos móviles y así mismo la misma plataforma debe poderse extender al uso de páginas de web.
- El Sistema debe ser capaz de aceptar pagos con tarjetas de crédito, *paypal*, *google pay* y *apple pay*.

Requerimientos no funcionales

- El sistema debe ser seguro según la cantidad de transacciones esperadas de los usuarios.
- El Sistema debe ser fácil de mantener y modificar.
- El Sistema debe ser fácil de extenderse para agregar más funcionalidades.
- El rendimiento del Sistema es considera de alta importancia en la determinación de la calidad de la solución.

- La administración del Sistema es muy importante, debe ser fácil de administrar y tener una buena experiencia de usuario.
- El Sistema debe tener una alta disponibilidad (cerca de 99%) y ser confiable en cuanto al manejo de datos.
- El Sistema debe ser fácil de usar e intuitivo para ser usado por cualquier tipo de usuario sin necesidad de ser un experto en dispositivos electrónicos o sitios web.

Acotando el sistema

Para determinar la solución del sistema fue necesario para mi acotar los límites que abarcará la solución que será capaz de hacer el sistema y qué no, en este sentido las siguientes oraciones determinan los límites del sistema y suposiciones que me ayudaron al diseño, confirmé dichas suposiciones en las sesiones de trabajo que tuve con el equipo del cliente.

- El sistema manejara moneda en dólares y pesos mexicanos en primera instancia, con posibilidad de extender el tipo de moneda en un futuro.
- El idioma del sistema será español por defecto pudiéndose cambiar a inglés dependiendo de la localización u configuración de idioma del navegador web.
- Al no tener infraestructura propia en las oficinas del cliente, la solución será montada en un entorno de la nube.
- Los proveedores de infraestructura, las licencias de sistemas externos (*third party*) serán pagadas y gestionadas por el cliente quien en todo momento podrá opinar e influir en la decisión de compra o licenciamiento.
- La solución de comercio electrónico se implementará basada en la plataforma SAP-Hybris que ofrece un entorno <<negocio a cliente>> B2C por sus siglas en inglés (Business-to-customer).
- La plataforma estará disponible para las regiones de Norteamérica y centro América en primera instancia, pudiendo extenderse a otras latitudes fácilmente haciendo una escalación en la infraestructura.
- El usuario proporcionará un catálogo de productos de prueba para ser cargados al sistema y realizar flujos completos de compra.
- El flujo completo de compra será el típico para un negocio electrónico, dejando cualquier modificación fuera del esfuerzo inicial.
- La misma funcionalidad provista para aplicaciones móviles será replicada en el sitio web.
- El proceso de devolución de productos será vía manual y fuera del alcance de este proyecto.
- Para el envío de productos se tendrá disponible una opción por defecto con entregas en México y Estados Unidos.
- El cálculo de impuestos contemplará en este primer alcance a México y Estados Unidos.
- El sistema de pagos deberá encargarse de la gestión de información cumpliendo con los estándares de <<Cumplimiento de la industria de tarjetas de pago>> PCI por sus siglas en inglés (*Payment Card Industry Compliance*).

Participación profesional

Como arquitecto de solución mi principal tarea consistió en conducir todas las sesiones de trabajo de acuerdo con la metodología del SEI descrita anteriormente. Mi principal objetivo con dichas sesiones fue conjuntar las expectativas del cliente en comparación con los procesos estándares del negocio del comercio electrónico.

Análisis de la solución.

Utilicé las sesiones de trabajo para obtener principalmente los requerimientos funcionales y no funcionales (enlistados en la sección anterior), en cada sesión abordé cada uno de los principales procesos en el negocio de comercio electrónico, conduje al cliente y lo orienté en cuanto a los principales puntos de integración que se deben tener en cuenta en el negocio de comercio electrónico.

Una vez obtenidos los datos anteriores determiné los requerimientos significativos del sistema (ASR's) que más adelante me sirvieron para diseñar la solución del sistema, en las sesiones de trabajo me reuní con el equipo técnico del cliente y con el equipo administrativo del proyecto, de tal manera que en cada sesión sostuve conversaciones técnicas y de negocio con los distintos roles del equipo del cliente.

Diseño de la solución.

En la etapa de la resolución y diseño del sistema utilicé toda la información recopilada en las sesiones de trabajo para llevar a cabo el diseño de la solución, en esta etapa es donde mi participación fue crucial ya que dependiendo de mi diseño se llevó a cabo la implementación por el equipo de desarrollo y se llevaron a cabo estimaciones de esfuerzo, por este motivo explicaré a continuación cada decisión que hice de diseño y su justificación con respecto a los requerimientos obtenidos por el cliente anteriormente y mi experiencia con pasadas implementaciones de negocio electrónico.

Selección de tecnologías

Plataforma comercio electrónico: SAP- Hybris

Como ya he explicado una plataforma de comercio electrónico conlleva muchos procesos críticos que involucran un proceso de implementación largo y costoso, la solución óptima hoy en día no es reinventar la rueda con cada implementación, debido a que todo comercio comparte procesos comunes en sus plataformas una buena opción es utilizar una plataforma ya diseñada y probada para implementar su propio negocio electrónico.

Hybris es una plataforma de comercio electrónico desarrollada y distribuida por la compañía SAP, dicha plataforma ofrece un conjunto de procesos de comercio electrónico listos para usarse bajo el concepto OOTB por sus siglas en inglés (*Out Of The Box*) lo que indica que es una plataforma que ofrece un núcleo

de procesos ya implementados en un lenguaje de programación fácil de extenderse y de usarse sin un gran esfuerzo de desarrollo.

La plataforma Hybris es una plataforma de comercio electrónico que funciona con los modelos negocio-a-cliente B2C (business-to-customer) una empresa que vende directo a cliente final y negocio-a-negocio B2B (businees-to-business) una empresa que vende a proveedores que a su vez comercializan los productos como intermediarios.

Hybris ofrece todos los procesos de comercio electrónicos básicos implementados y listos para usarse desde la instalación de la plataforma. Flujos como orden de compra, carrito de compra, administración de productos y catálogos, etc. son algunas de las funcionalidades que la plataforma ya incluye al comprar su licencia empresarial, dicha plataforma está preparada para ser extendida añadiendo o modificando componentes de acuerdo con las necesidades del cliente mediante nuevos módulos utilizando la arquitectura interna y lineamientos de la plataforma indicados en su documentación oficial.

Las tecnologías involucradas en esta plataforma incluyen java, Apache, SQL, Spring MCV, security, ORM, entre otros.

Además de los procesos del núcleo del sistema Hybris ofrece un nuevo concepto llamado <<aceleradores>> este concepto se refiere a una característica de la plataforma en la que se puede generar, a partir de módulos extras, una interfaz web lista para usarse de comercio virtual con la única tarea de personalizar colores e imagen del comercio en sí, dicho en pocas palabras un acelerador de Hybris crea una plataforma de comercio electrónico desde los procesos básicos hasta la interfaz de usuario listos para ser utilizados y dejando al equipo de desarrollo la tarea de personalizar y adaptar la plataforma a las necesidades específicas de cada negocio.

De esta manera analizando todas las bondades de la plataforma Hybris en conjunto con los requerimientos de negocio del cliente determiné que la solución debía ser planteada bajo el uso de dicha plataforma y seguir diseñando, tomando en cuenta la arquitectura y técnicas de integración con ella.

Plataforma Marketplace: Mirakl.

Una vez seleccionada la plataforma de comercio electrónico evalué una solución para integrar la funcionalidad de *Marketplace* es decir, con Hybris tenía cubierta la funcionalidad de la compra de productos y manejo de catálogos incluyendo una interfaz de usuario genérica lista para usarse, sin embargo de acuerdo a los requerimientos funcionales la expectativa del usuario es que los usuarios vendedores tengan la posibilidad de crear sus propios catálogos y tener tiendas virtuales, este tipo de funcionalidad no es provista por Hybris y el desarrollo desde cero puede ser costoso en tiempo tomando en cuenta los estándares de calidad que se necesitan cumplir por el sistema.

Tomando en cuenta los requerimientos no funcionales costo y tiempo decidí optar por integrar la plataforma Hybris con una plataforma de *Marketplace* que proporcionara toda la funcionalidad necesaria para los usuarios vendedores.

Después de un análisis profundo de diversas plataformas de Marketplace decidí usar la plataforma Mirakl, a continuación, enlisto los motivos:

I. *Mirakl* dicha plataforma ofrece un sistema para comercio en línea tipo *Marketplace* es decir una plataforma en donde usuarios pueden cargar y vender sus productos, con los procesos básicos implementados OOTB.
II. Ofrece una <<oficina virtual>> a cada vendedor para administrar su catálogo de productos y sus ofertas.
III. Trabaja bajo el concepto de PaaS (Platform As a Service) es decir que toda la infraestructura de sus servidores esta alojada en la nube privada de Mirakl y ofrece una disponibilidad garantizada de un 99.99% y escalabilidad horizontal elástica para horas pico de tráfico.
IV. Sus formatos de integración son estándares de la industria y usa servicios web por REST y SOAP.
V. Ofrece un conector previamente desarrollado y probado para una integración transparente con la plataforma Hybris. Dicho conector es incluido con la licencia y está desarrollado bajo la arquitectura y recomendaciones de Hybris.
VI. La documentación oficial es robusta y la tecnología sobre la cual se desarrolló ofrece una API (Application Programming Interface) y guía de como extender funcionalidad necesaria en caso de requerirlo.
VII. Prueba de concepto. Realicé una prueba de concepto integrando Hybris y Mirakl usando el conector provisto de Mirakl con licencias de prueba para ambas plataformas, los resultados de dicha integración fueron:
    I. Integré Hybris y Mirakl con una configuración de mediana complejidad guiada por la documentación oficial.
    II. Mirakl ofrece tareas sincronizadas (*cronjobs*) para la importación de productos entre plataformas previamente configurados.
    III. Pude realizar la creación de tiendas virtuales sin mayor esfuerzo solo siguiendo la guía en la documentación.
    IV. Sincronicé la importación de tiendas virtuales y catálogos de marcas y productos con una tarea sincronizada.
    V. Integré el mecanismo estándar de integración de hybris <<*hot folder*>> con Mirakl usando tareas sincronizadas, lo que facilita la integración general.

Al tomar en cuenta los puntos anteriores y haciendo una revisión de los requerimientos del sistema determiné que la plataforma Mirakl sería un componente que aportaría a la solución toda la funcionalidad esperada del *Marketplace*.

Con respecto a la Integración Hybris Mirakl hice un análisis exhaustivo sobre ambas plataformas y sus mecanismos de integración, evalué la escalabilidad, manejabilidad, seguridad, usabilidad, rendimiento,

disponibilidad y capacidad de integración juntas, una vez hecha la investigación y hacer pruebas de concepto de integración entre ambas plataformas proseguí con el diseño de la solución.

Sistema de pagos: *Braintree Gateway*

Ya que determiné las plataformas que se integrarían para cumplir los requerimientos funcionales básicos, me di a la tarea de comenzar a darle solución a los procesos que aún no estaban completos del todo en cuanto a requerimientos y uno de los más importantes es el proceso de pago.

La plataforma Hybris ofrece de manera inicial los procesos básicos del comercio electrónico, sin embargo, debido a la naturaleza específica del proceso de pago no puede ofrecer todo el flujo completo en el proceso de compra.

Debido a que ningún dato de los requeridos para hacer un pago (número de tarjeta, fecha de vencimiento, CVV) pueden ser guardados por la plataforma de comercio electrónico por la regulación de la Industria del Pago de tarjetas en su apartado de estándar de seguridad de datos (PCI DSS por sus siglas en inglés) que dicta que ningún dato de un método de pago puede ser guardado en la plataforma, es factible integrar una solución con la certificación de la regulación mencionada en el sentido de ahorrar tiempo y dinero en una implementación propia.

Hice una evaluación de distintas soluciones de pago para integrar con Hybris, tomé en cuenta principalmente el requerimiento funcional especificado por el usuario en donde menciona que:

“El Sistema debe ser capaz de aceptar pagos con tarjetas de crédito, *paypal, google pay y apple pay*.”

Por ello mi principal discriminante para la elección de una plataforma de pagos consistió en el cumplimiento de tener soporte para los métodos de pago mencionados arriba.

Además de los requerimientos funcionales, fue importante para mí cumplir con los requerimientos no funcionales para este particular componente del sistema, es decir que la plataforma ofreciera un nivel de cifrado robusto para los datos almacenados, que los métodos de integración fueran suficientes y compatibles con Hybris y que ofreciera una alta disponibilidad capaz de soportar el número de transacciones proyectadas para el sistema.

Elegí la plataforma *Braintree Gateway* por el cumplimiento de cada requerimiento funcional y no funcional nombrados arriba, además de sus mecanismos de integración basados en servicios web en una arquitectura cliente servidor y un robusto sistema de seguridad.

Antes de incluir de lleno a Braintree como plataforma en la solución, realicé pruebas de concepto de integración usando el ambiente controlado (*sandbox*) de dicha plataforma junto con hybris, los resultados que obtuve fueron los siguientes:

I. Pude integrar la plataforma de pagos Braintree con Hybris de forma exitosa usando la documentación oficial provista por la plataforma de pagos.

II. Di de alta un método de pago en la plataforma de forma exitosa tanto para una tarjeta de crédito, *paypal*, *apple pay* y *google pay.*
III. Realicé un pago con una tarjeta de crédito (datos reales de prueba) usando la integración hecha y obtuve una transacción exitosa. Así mismo realice el mismo ejercicio de forma exitosa con paypal, google pay y apple pay.
IV. Verifiqué que la información que es almacenada en la plataforma de pagos fuera recuperable de forma parcial para presentar datos en la interfaz de usuario.

Así pues, después de analizar dichos puntos y comprobar el funcionamiento mediante las pruebas de concepto mencionadas, seleccioné a la plataforma *Braintree Gateway* como plataforma de pagos para la solución.

Con esta integración el flujo de pagos se completé y cumplí uno de los procesos cruciales de la plataforma de comercio electrónico, es decir la compra de productos y recepción de pagos.

Sistema de notificación: COMAPI-INBOX

Una vez que conformé la solución del proceso de pagos y manejo de catálogo de productos mediante las plataformas descritas, proseguí a analizar las posibilidades para cumplir los requerimientos funcionales referentes a las notificaciones del sistema, como lo enlisté arriba el sistema debe ser capaz de enviar notificaciones a los dispositivos móviles en caso de que allá alguna oferta nueva en los productos publicados.

Tanto Hybris como Mirakl no ofrecen esta funcionalidad de entrada por lo que analicé la posibilidad de hacer una implementación personalizada en comparación con utilizar una plataforma especializada e integrarla a la solución.

Evalué el esfuerzo necesario para realizar una implementación personalizada del sistema de notificaciones, note mediante un análisis de la arquitectura necesaria que dicha implementación involucraría un aumento de horas de trabajo sustancial en la estimación de esfuerzo y la posibilidad de fallos era de un grado mediano debido a la falta de experiencia del equipo de desarrollo en dicho ramo.

Con esto en mente y tomando en cuenta la petición del cliente de tener un sistema en ambiente productivo pronto (*time to market*) determiné que la solución debía incluir una plataforma de notificaciones que cumpliera con los requerimientos descritos por el cliente.

Analicé distintas plataformas para la solución de notificaciones y determiné que aquella que se adaptó mejor a los requerimientos era COMAPI-INBOX a continuación enlisto las razones puntuales de la elección de la plataforma:

I. Ofrece un SDK (*software development kit*) para hacer la integración con el lenguaje Java y javascript.

II. Los mecanismos de integración provistos por COMAPI-INBOX están basados en los estándares de servicios web respecto a REST, esto facilita la integración con la plataforma Hybris.
III. La seguridad usada está basada en oAuth que es un estándar hoy en día para la integración de servicios web basados en REST compatible con Hybris.
IV. Ofrece un sistema de notificaciones de última generación con integración nativa en dispositivos móviles, ofrece un portal de administración integrado para enviar notificaciones masivas (*broadcast*).
V. Tiene la posibilidad de crear grupos de usuarios para enviar notificaciones especificas a un grupo selecto, es decir crear listas de distribución.
VI. Realicé una prueba de concepto de integración entre la plataforma de notificaciones y Hybris, obtuve los siguientes resultados:
    1. Pude integrar COMAPI-INBOX con Hybris de manera exitosa usando el ambiente controlado (*sandbox*) de la plataforma de notificaciones, usé el SDK provisto por la plataforma y con la guía de la documentación oficial.
    2. Pude enviar notificaciones masivas por medio de COMAPI-INBOX usando la integración antes descrita.
    3. Envié notificaciones a grupos específicos emulando promociones disparadas por el sistema.
    4. Utilizando un dispositivo móvil de prueba con sistema operativo Android y IOS comprobé que la plataforma de notificaciones fuera capaz de enviar correctamente los datos y por su parte los dispositivos móviles tuvieran la capacidad de leer dichos datos y asimilarlos a manera de notificaciones dinámicas.

Después de tomar en cuenta todas las consideraciones arriba mencionadas añadí a la plataforma COMAPI-INBOX a la solución integral del sistema.

Documentación de la solución y elección de vistas

Una vez que realicé la selección de tecnologías y comprobé mediante pruebas de concepto su compatibilidad, proseguí a conformar la propuesta de solución final integrando todos los componentes. Para poder exponer la solución hice la documentación apropiada de acuerdo con la metodología del SEI mencionada previamente. Comencé con el diagrama general de la solución y después documentando cada canal de integración crucial para el sistema.

A continuación, muestro el diagrama de arquitectura de la solución general con los componentes mencionados y las integraciones evaluadas:

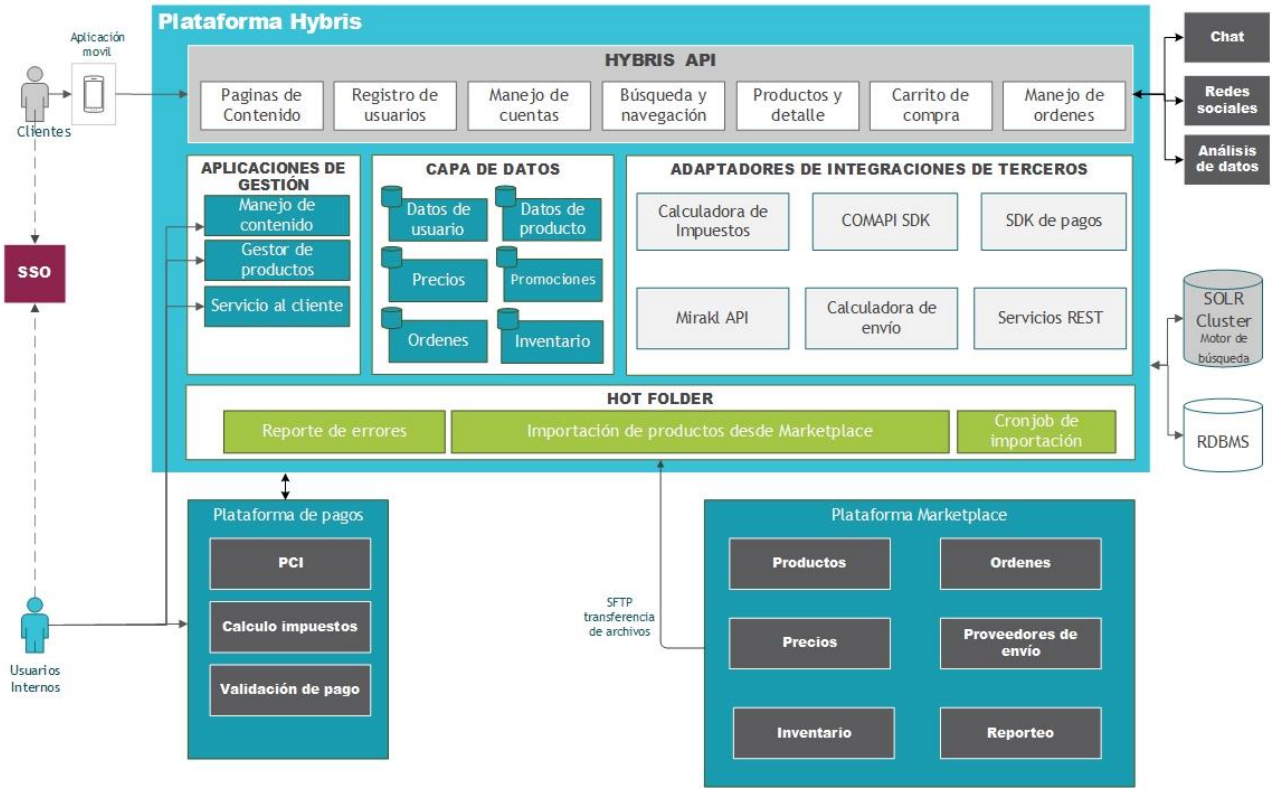


Figura 18. Diagrama a alto nivel de componentes tomado del documento de arquitectura que realicé.

Canales de Integración

Una vez hecho el diagrama a alto nivel proseguí a explicar detalladamente los canales de integración en la solución, cómo lo menciona la metodología del SEI es necesario hacer una documentación apropiada de los puntos de integración del sistema y más significativos para la arquitectura, las vistas del sistema que diseñé las documenté en detalle para aclarar cualquier duda en la integración y sirvieran como punto de partida para el equipo de desarrollo en la fase de implementación.

A continuación, enlisto los canales de integración a detallar:

Sistema	Nombre	Funcionalidad	Comentarios
Sistema de pago	Braintree Gateway	Procesar pagos en la plataforma de comercio electrónico	Hay disponible un conector para integrar este sistema con hybris
Sistema de notificación	COMAPI INBOX	Sistema de notificación externo	Se puede configurar usando el SDK provisto por el sistema COMAPI
Mirakl Marketplace	Marketplace	Plataforma de marketplace	Hay disponible un conector para integrar este sistema con Hybris

Plataforma Hybris B2C	Hybris	Plataforma de comercio electrónico	Plataforma Hybris que provee los procesos elementales para procesar compras

Mirakl-Hybris

Sistema 1: Mirakl Marketplace.

Sistema 2: Hybris.

Nombre del canal: Mirakl-Hybris.

Datos de integración: Productos, ordenes, categorías, cargo de envío, cliente.

Funcionalidad: Manejar importación de productos, importación de categorías, ordenes, tiendas virtuales, ofertas y disponibilidad en stock. Este canal es el encargado de integrar toda la funcionalidad de comercio electrónico de Hybris con el marketplace de Mirakl. Los procesos básicos de comercio electrónico se integrarán con la naturaleza de marketplace sobre tiendas virtuales.

Tipo de Integración: SDK, asíncrona, dirección de dos vías.

Mecanismo de integración: Servicio web REST y *Webhooks*.

Especificación del API: Documentación oficial provista por Mirakl.

Rendimiento esperado: dos mil ordenes por día.

Mirakl ofrece administración y configuración de tiendas virtuales, vendedores, productos y categorías a través de una interfaz de interfaz de usuario de tipo *backoffice,* es decir, es una interfaz para usuarios de negocio en la cual pueden administrar los procesos de importación los productos dados de alta por cada tienda virtual, crear tiendas virtuales y administrar ofertas. Mirakl se integra con Hybris usando un conector que se puede instalar como una extensión común en la plataforma, el proceso de instalación se encuentra publicado en la información oficial de Hybris (sólo accesible después de comprar la licencia de Hybris). Con este conector, Hybris puede importar y exportar información del producto, pedidos, tiendas y categorías del acelerador B2C (*Business to Customer*) y ver toda esta información usando la interfaz *backoffice*.

Las integraciones a Mirakl las realicé a través del conector Mirakl-Hybris y tareas programadas (*cronjobs)* definidos por Mirakl para importar/exportar información entre sistemas. La integración puede ilustrarse con este diagrama:

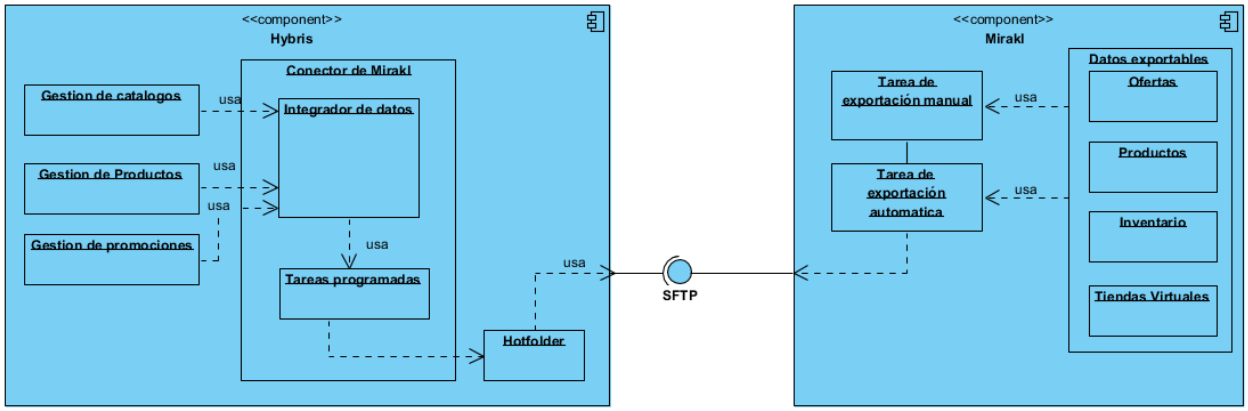


Figura 19.- Vista integración Hybris y Mirakl mediante conector

Diseñé la arquitectura de tal manera que toda la comunicación se realizará a través de un canal seguro lo que garantiza la seguridad de la información y la integridad. La transferencia de datos la hice por un puerto FTP seguro (SFTP) que usé para importar todos los datos de productos, tiendas virtuales y promociones a Hybris mediante el mecanismo <<Hot Folder>> que es un mecanismo estándar de integración de datos de Hybris, este sistema está basado en una locación lógica del sistema (*file system*) en donde se colocarán los archivos importados de Mirakl en formato CVS.

Mirakl ofrece una interfaz de usuario para administrar todas las operaciones, entre ellas monitorear la exitosa importación o exportación de productos además de un registro de actividad para detectar si es que algún producto o tarea programada fallo en la importación o exportación. La interfaz de usuario proporcionada incluye manuales para cada operación que se puede realizar y una sección de errores comunes a la que se puede recurrir para resolver dudas sencillas, dicha documentación se proporciona con la compra de la licencia de Mirakl.

Sistema de Pago

Sistema 1: Sistema de Pago.

Sistema 2: Hybris.

Nombre: Braintree Gateway.

Datos de integración: Métodos de pagos, número de tarjeta, nombre de cliente, fecha de vencimiento, dirección de facturación y monto de pagos.

Funcionalidad: pagar ordenes, reembolsar ordenes, almacenar y asegurar los métodos de pago del cliente.

Tipo de integración: SDK, síncrono, unidireccional.

Mecanismo de integración: REST, servicio web.

Rendimiento esperado: dos mil ordenes por día.

Elegí el sistema de pago del proveedor Braintree consistente en una arquitectura cliente-servidor, hice un análisis sobre la compatibilidad de distintos servicios de pago con la plataforma Hybris y después de evaluar, rendimiento, disponibilidad, niveles de servicio, seguridad, métodos de integración, soporte a versiones anteriores y precio de licencia determiné mediante matrices de decisión y empatando con los requerimientos funcionales y no funcionales que Braintree era la mejor opción para cumplir los requerimientos del sistema.

Braintree es un componente desarrollado originalmente para integrarse con la interfaz de usuario en conjunto con los componentes del servidor, javascript y java respectivamente, decidí usar el mecanismo de seguridad basado en tokens uno del lado del servidor y otro del lado del sistema de pago (llamado *nonce* por Braintree) para el intercambio de información debido la alta sensibilidad de los datos a procesar.

El dato *nonce* es un número arbitrario que se usa una única vez para una autenticación segura, es un número diferente para cada método de pago y asegura la autenticidad de la transacción proporcionando integridad a los datos de pago y ayuda a prevenir fraudes.

Así pues, usando el concepto de *nonce* y mediante el uso de *tokens* dinámicos logré una autenticación de la transacción segura para dar de alta métodos de pagos.

Un *token* es un numero arbitrario y único que identifica a un recurso del sistema, usualmente pude ser conformado por datos del usuario especifico que lo utiliza a los que se les aplica algún tipo de cifrado como funciones hash.

Una vez que se da de alta un método de pago con el uso de *nonce* y *token*, se guarda en la base de datos de Hybris un *token* generado por Braintree específico para ese método de pago y ese usuario que se tendrá que enviar cada vez que se utilice dicho método de pago, de esta forma el sistema puede persistir métodos de pagos como tarjetas de crédito guardadas para usuarios registrados en la plataforma, de la misma manera sirvió para guardar métodos de pagos para cada usuario para el caso de *apple pay*, *google pay* y *paypal*.

A continuación, detallé en el documento de arquitectura el flujo de interacción con el servidor de Braintree para almacenar y procesar pagos con la verificación de seguridad especificada por Braintree.

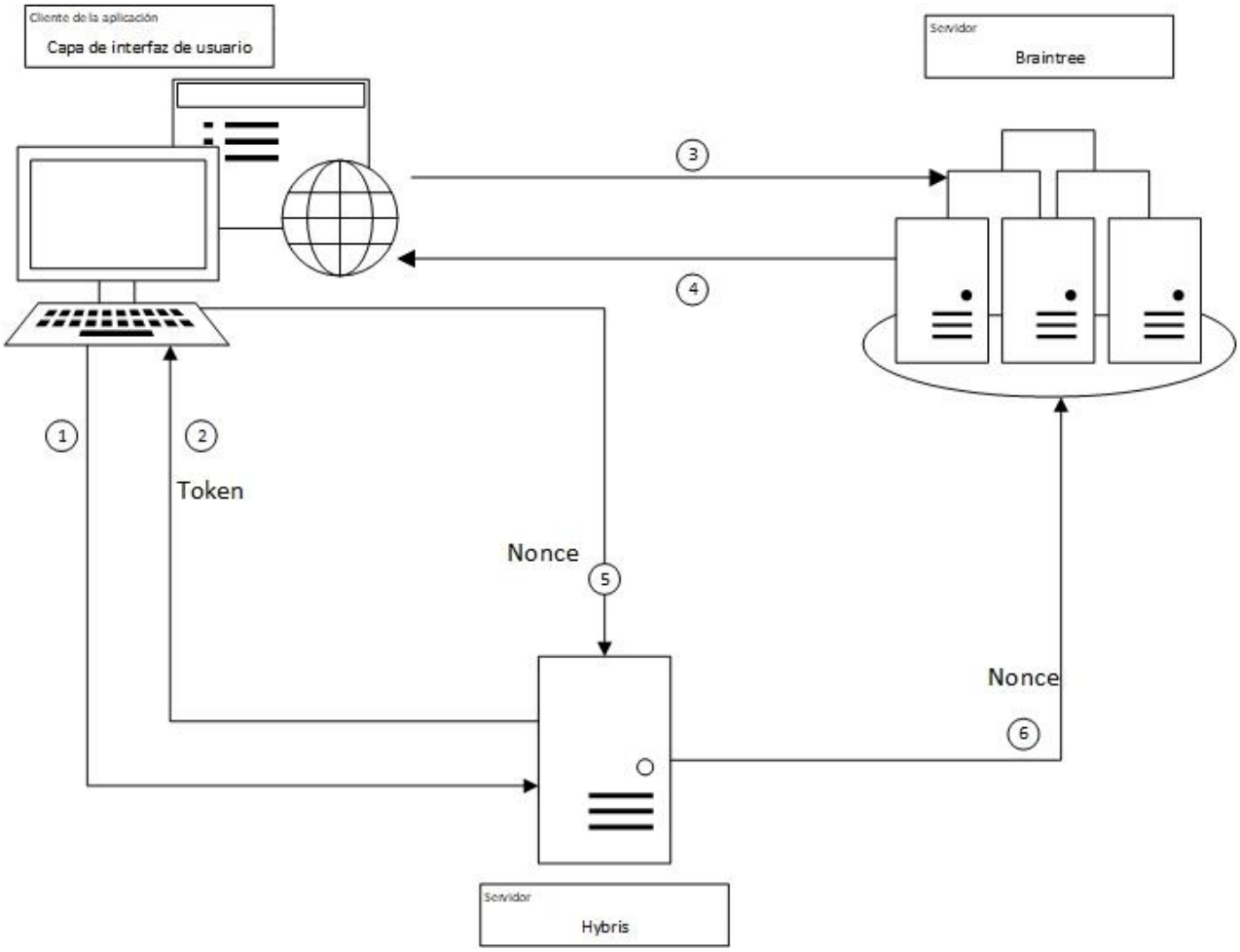


Figura 20.- Integración de Sistema de pagos Braintree

Este es el diagrama de Arquitectura general de Braintree que diseñé para la documentación, como está marcado, hay una serie de 6 pasos que se deben seguir para dar de alta un medio de pago y generar el token que se usará cada vez que se realice una transacción con dicho método.

A continuación, describo cada uno de los pasos:

1. La aplicación cliente (web o móvil) pide un token de autenticación al servidor de Hybris.
2. El servidor de Hybris genera un token basado en el SDK de Braintree y lo regresa como respuesta a la aplicación cliente.
3. La aplicación cliente hace una petición al servidor de Braintree solicitando el *nonce* y envía el token obtenido de Hybris.
4. El servidor Braintree valida el token enviado por la aplicación cliente y genera el nonce correspondiente que regresa a la aplicación cliente.

5. La aplicación cliente envía al servidor de Hybris una solicitud para realizar una transacción de pago enviando el monto de la operación y datos de pago incluyendo en *nonce.*
6. El servidor de Hybris envía una petición al servidor Braintree primero para registrar un método de pago con los datos de pago y después para realizar la transacción de pago en ambas solicitudes envía el *nonce.*

Reflejé este flujo en los componentes del sistema tomando en cuenta la aplicación nativa del cliente Android y IOS y la implementación Braintree en la plataforma Hybris.

A continuación, muestro un diagrama de componentes que hice para exponer mi implementación:

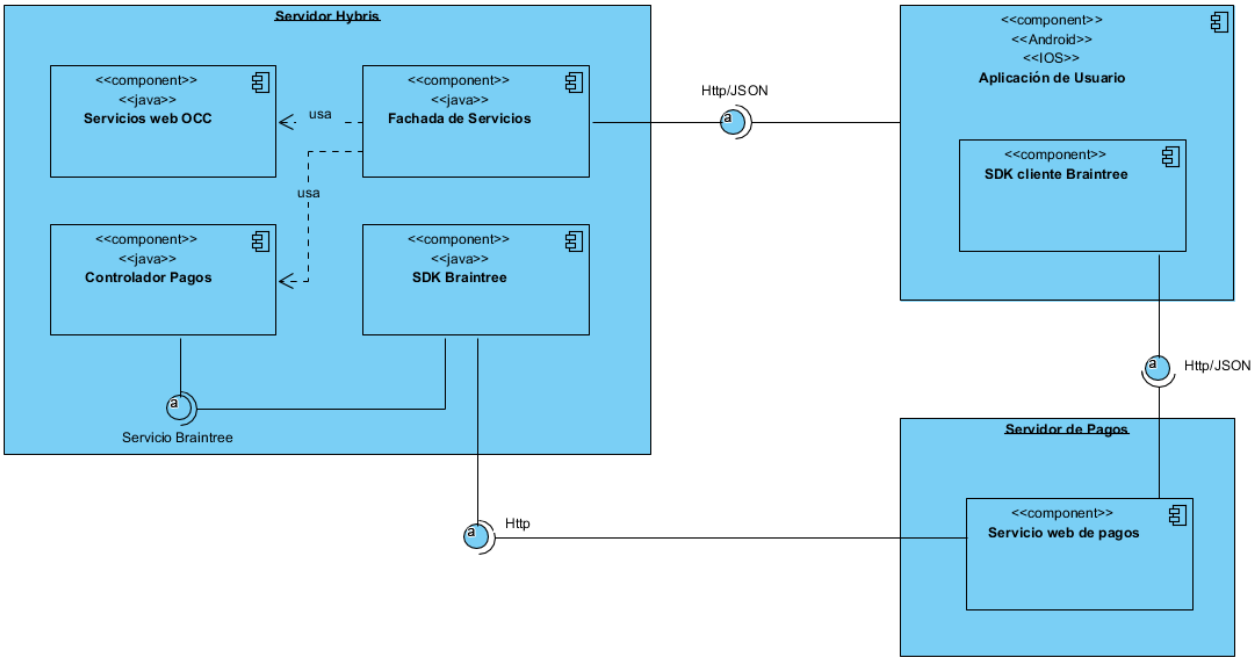


Figura 21.- Diagrama de componentes Braintree con Hybris

En el diagrama se puede apreciar que las interfaces de integración son servicios web tanto para el servidor de Braintree, así como para Hybris, así mismo podemos ver el componente controlador de pago que en términos sencillos es una implementación de java del patrón de diseño Modelo-Vista-Controlador (MVC por sus siglas en ingles).

Una vez que implementé esta integración fue posible para el sistema recibir pagos por medio de *paypal*, tarjetas bancarias, *google pay* y *apple pay* cumpliendo así con uno de los requerimientos funcionales imprescindibles en la solución.

Braintree cumple con el estándar de seguridad de datos de la industria de tarjetas de pago (PCI DSS) lo que garantiza la seguridad de la información de los métodos de pago, y asegura que la información de facturación y pagos se almacene en *Braintree Vault* con un alto nivel de cifrado.

En caso de interrupción del servicio de Braintree determiné que las transacciones fueran encoladas en la plataforma Hybris hasta que se reestablezca el servicio, usando el servicio de colas que ofrece como procesos estándar la plataforma.

Sistema de Notificación

Sistema 1: Sistema de notificación.

Systema 2: Hybris.

Nombre: COMAPI-INBOX

Datos de integración: Ofertas, nuevos lanzamientos y promociones.

Funcionalidad: Notifica acerca de nuevos productos u ofertas disponibles en el sistema.

Tipo de Integración: SDK, asíncrono, integración bidireccional.

Mecanismo: REST, servicios web o *Webhooks*.

Rendimiento esperado: dos mil notificaciones al día.

El Sistema diseñado debía ser capaz de enviar notificaciones a sus clientes cuando nuevas ofertas o productos tuvieran lugar en la plataforma, tomando en cuenta dicho requerimiento funcional, me di a la tarea de evaluar diferentes sistemas de notificación que cumpliera con los requerimientos que el sistema demandaba como lo muestro en el análisis previo.

COMAPI-INBOX usa dos mecanismos para enviar notificaciones documentados en la guía oficial:

1.- Usando el sitio web <<control de INBOX>> que se usa para enviar notificaciones basados en campañas o por medio de análisis de datos, este sitio web ofrece herramientas de análisis de datos que ayudan a crear reglas automatizadas de notificaciones, se requiere la interacción de un usuario, ya que la creación de estas reglas es manual.

2.- Usando INBOX-API se pueden enviar notificaciones consumiendo una interfaz de servicios web proporcionada por la plataforma de notificaciones, este método requiere codificación para integrar estos servicios con la plataforma deseada.

Puntos claves a considerar en la implementación:

1.- El Sistema necesita enviar notificaciones a un grupo específico de usuarios ya sea usando el <<control de INBOX>> o usando la INBOX-API la especificación de la documentación menciona dos opciones a utilizar, las describo a continuación:

Enviar notificaciones a todos los usuarios (*AllUsers*), con esta opción se enviará una notificación específica a todos los usuarios registrados en la plataforma de notificaciones ya sea a sus dispositivos móviles o sitio web.

Para poder enviar notificaciones a grupos específicos de usuarios es requerido por la plataforma proveer un grupo específico de id's de usuarios a los cuales se les notificará sobre alguna promoción, para dicho propósito diseñe una tabla de la base de datos encargada de persistir los datos de grupos específicos de usuarios suscritos a promociones a alguna promoción en específico.

2.- Gestionar grupos de usuarios: De acuerdo a los requerimientos funcionales las notificaciones deben enviarse a todos los usuarios que se suscriban a las actualizaciones de una marca en una categoría de productos y cada vez que un nuevo producto se agregue a dicha marca se les enviará una notificación a los grupos de usuarios correspondientes, para realizarlo diseñe un flujo de notificación que sirvió para decidir cuándo enviar una notificación desde el servidor de Hybris ya que los productos están alojados en su base de datos.

3.- Para borrar o añadir usuarios en un grupo de usuarios diseñé una clase de control que se encarga de actualizar la tabla de la base de datos que persiste la información de grupos de usuarios, dicho diseño lo incluí en el anexo del documento de la arquitectura para el equipo de desarrollo.

4.- El proceso de mostrar las notificaciones en la aplicación móvil y en el sitio web requiere la integración del SDK de COMAPI-INBOX para realizarlo realicé una guía de pasos basados en la documentación oficial de COMAPI.

Para llevar a cabo la implementación de este canal realicé una investigación exhaustiva de la documentación oficial que también añadí en un anexo del documento de arquitectura presentado al cliente final.

Las pruebas de concepto realizadas también las agregue como parte de la documentación adicional para el equipo de desarrollo.

A continuación, detallo el proceso de notificaciones que diseñé para enviar y recibir notificaciones desde la plataforma de Hybris, en dicho flujo se puede ver el proceso que sigue una notificación desde que un producto es exportado en Mirakl y pasa por la plataforma de Hybris en dicha plataforma se hace la validación de productos nuevos y la revisión de los usuarios que pertenecen al grupo que se va a modificar.

Una vez validado en Hybris a que usuarios se notificará indico los pasos que se deben seguir en la plataforma COMAPI para hacer uso de su sistema de notificaciones y como paso final él envió al dispositivo final.

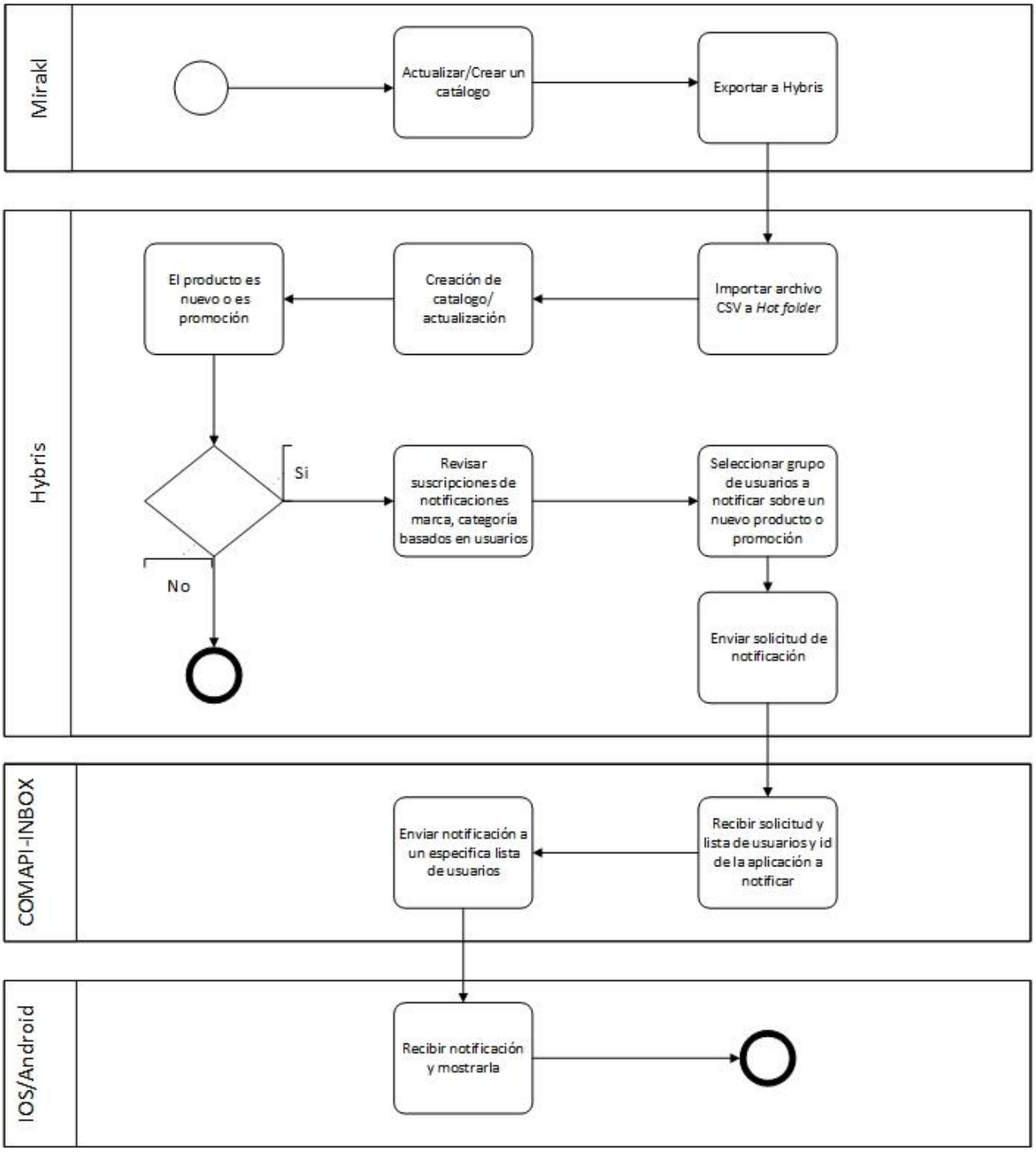


Figura 22.- Descripción del flujo de notificación

Una vez definidos y detallados los canales de integración en las vistas previas conforme el documento de arquitectura para proponer la solución final.

El documento final de arquitectura la presenté en una serie de 3 reuniones en las que recibí retroalimentación del equipo del cliente final, conforme a dichos comentarios realicé cambios menores en el diseño y volví a presentar el documento con los cambios integrados.

El proyecto y diseño fue aprobado y comenzó la implementación de acuerdo con el diseño que propuse, durante la implementación mi participación fue principalmente como consultor de dudas técnicas y asegurando que la implementación cumpliera de tal forma que no se perdieran de vista los requerimientos funcionales y no funcionales.

Implementación

El equipo de desarrollo fue el principal protagonista durante la implementación, como lo mencioné antes en el marco teórico el arquitecto baja considerablemente en carga de trabajo durante la implementación sin embargo si participé en tareas cruciales de calidad y procesos de desarrollo que detallaré a continuación.

Aporté explicando detalladamente las pruebas de concepto que realicé y explicando los procesos de debía cumplirse.

Realicé un documento de estándares de programación y buenas prácticas en las que declaré las reglas generales que debían seguirse por los desarrolladores para darle calidad al código, esta serie de reglas fueron cruciales para llevar a cabo una implementación capaz de ser entregada al cliente.

Para el aseguramiento de calidad del código usé la herramienta *Sonar Qube* para una inspección continua del código, esta herramienta permite la detección en tiempo real de errores en la validación de reglas definidas y ofrece integración con las herramientas de programación más usuales para el desarrollo como *Eclipse* e *IntelliJ.*

Definí las reglas basado en el marco de trabajo *Sonar Qube*, en términos sencillos cada regla la definí en un documento XML y el marco de trabajo de *Sonar Qube* se encarga de inspeccionar el código desarrollado en tiempo real y anunciar al desarrollador sobre la violación de una regla, a continuación, muestro un ejemplo de una regla violada en la herramienta para programación *IntelliJ*:

Figura 23.- Ejemplo de inspección continua de código con *Sonar Qube*

Además de participar en el aseguramiento de calidad del código, di soporte a cualquier duda técnica del equipo de desarrollo respecto a la implementación de las diferentes capas y componentes de Las plataformas descritas anteriormente.

La metodología que seguimos en la implementación fue Scrum que es una metodología ágil explicada a detalle en la sección de marco teórico de este documento.

En cada iteración participe en el refinamiento de los requerimientos a nivel técnico para darles una correcta guía a los desarrolladores encargados de implementar cada característica del sistema sin embargo la documentación técnica de cada clase modificada o creada corrió a cargo de cada desarrollador.

Por mi parte participe en la definición del flujo para una correcta implementación de Integración continua (CI por sus siglas en ingles *Continuos Integration*), que consistió principalmente en establecer un flujo de validaciones de código para realizar una correcta integración de versiones en la herramienta de repositorio de código *GitHub* y generar el aplicativo entregable, es decir, en términos técnicos, el archivo ejecutable que deber ser desplegado en un servidor de aplicaciones.

Participe también en la definición del flujo para un correcto despliegue de una nueva versión de la aplicación en los diferentes ambientes del sistema (desarrollo, pruebas, preproducción y producción), a este proceso se le llama Despliegue continuo (CD por sus siglas en ingles Continuos Deployment) el flujo en cuestión valida que el archivo ejecutable haya pasado los lineamientos de seguridad en la etapa de Integración Continua y si es así procede al despliegue en el ambiente seleccionado, si es necesario reiniciar el servidor o ejecutar alguna rutina en específico para preparar el ambiente para el despliegue entonces la ejecuta automáticamente con el propósito que el despliegue sea ágil y transparente para el usuario.

Para realizar la automatización de los procesos mencionados arriba (integración continua y despliegue continuo) utilicé la herramienta *Jenkins* que es bien conocida y prácticamente un estándar en la implementación de CI/CD, *Jenkins* define los procesos tanto de despliegue como de integración con un concepto al que llama *pipelines* estos son nada más que un conjunto de tareas secuenciales que deben ejecutarse paso a paso para asegurar ya sea la correcta implementación o correcto despliegue de un aplicativo, cada tarea en un pipeline puede ser desde la ejecución de una rutina a nivel *Shell (*consola de comandos*)* hasta la ejecución de validación de pruebas de código de *SonarQube*.

Por cuestiones de confidencialidad no puedo compartir el pipeline de mi proyecto sin embargo a manera ilustrativa se puede ver en la imagen a continuación un ejemplo de un pipeline en Jenkins.

Figura 24.- *Ejemplo de Pipeline en Jenkins*

Una vez que quedaron definidos los flujos de integración y despliegue el equipo de desarrollo fue capaz de continuar la implementación del sistema.

Diseño de la Infraestructura y despliegue en producción.

La definición de la infraestructura es otra tarea en la que tuve mayor injerencia, me encargué de hacer el diseño y asegurar una correcta implementación de dicho diseño en los diferentes ambientes.

La infraestructura que seleccioné por requerimiento no funcional especifico del cliente de <<en la nube>> debido a su atractiva oferta y su garantía de servicios el proveedor AWS (*Amazon Web Services*) fue el que seleccioné basándome primeramente en cumplir las expectativas de alta disponibilidad, rendimiento, costo y compatibilidad entre plataformas.

A continuación, ilustro el diagrama de arquitectura de la infraestructura que definí explicando a detalle los componentes involucrados y su respectiva justificación:

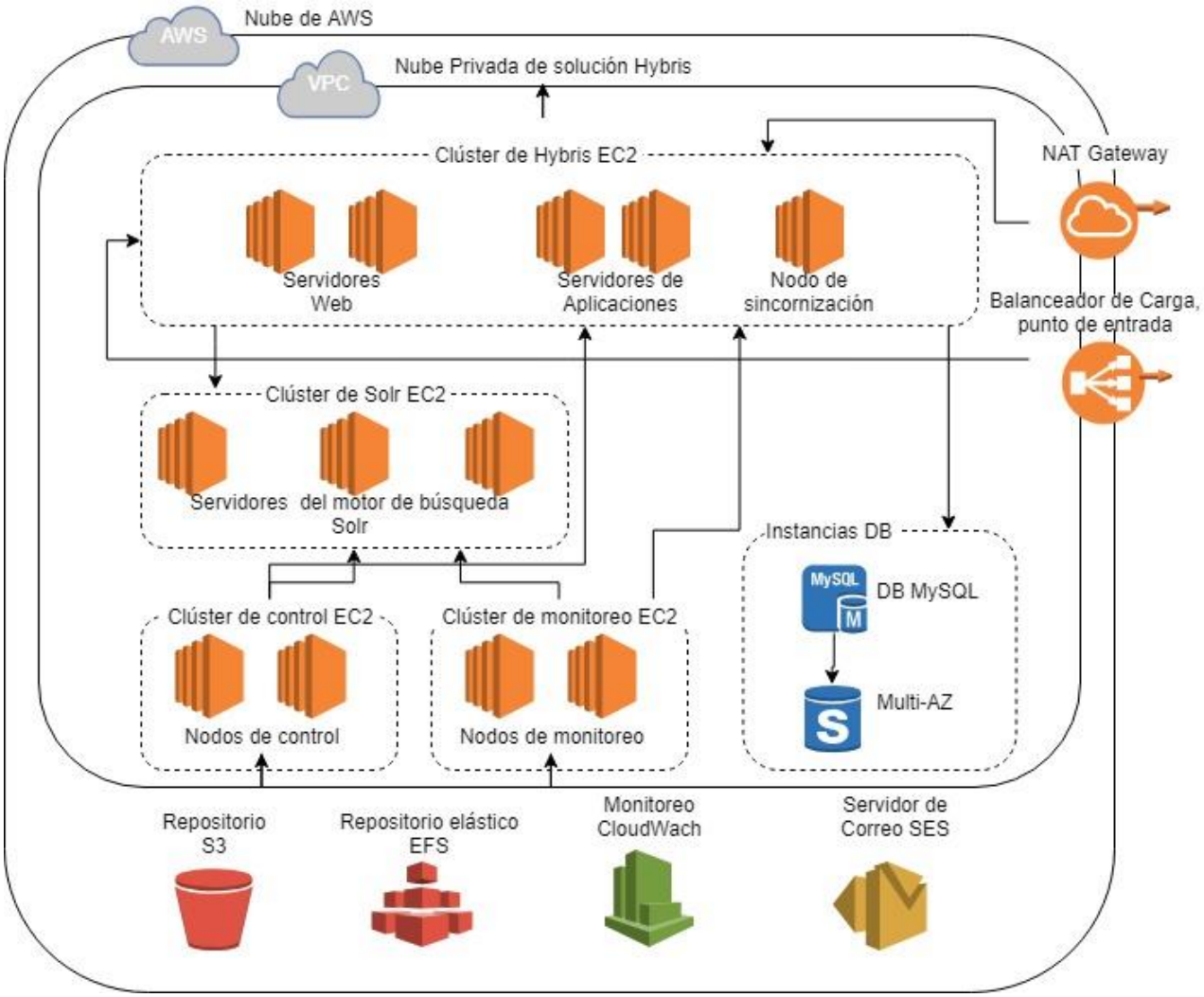


Figura 25.- Infraestructura de la solución en AWS.

Clúster de Hybris EC2:

En este componente decidí que se alojarían los servidores tanto para la aplicación web como para los procesos de negocio, llamados comúnmente en el argot de desarrollo como *front-end* y *back-end* respectivamente decidí usar instancias EC2 (*Elastic Compute Cloud*) que son componentes genéricos que ofrece AWS para alojar servidores web o de aplicaciones, dichos componentes son flexibles y se pueden escalar fácilmente mediante la administración de la nube privada en AWS.

En dicho clúster definí dos instancias para la capa web y dos instancias para la capa de procesos de negocio teniendo así redundancia haciendo el sistema tolerante a fallos (*failover*) y mejorando el rendimiento, aunado a eso agregué un nodo de sincronización para ejecutar procesos de replicaciones de sesiones de forma desacoplada para no perder ningún dato de ninguna transacción en caso de algún fallo en alguna de las instancias.

Clúster de Solr EC2:

Utilizando el mismo componente EC2 de AWS descrito en el punto anterior definí el clúster del motor de búsqueda Solr, este es el motor de búsqueda incluido con la plataforma Hybris. Con el propósito de maximizar el rendimiento de las búsquedas basadas en índices configuré tres instancias en este clúster que replicaría los datos cada 15 minutos de manera automática y cada vez que un nuevo producto se añadiera al índice de búsqueda. La configuración de dichas instancias se puede llevar a cabo tomando como guía la documentación oficial de la plataforma Hybris.

Clúster de control EC2:

Definí este clúster con el propósito de tener dos instancias encargadas de realizar los procesos de mantenimiento y control regular de la plataforma, estos procesos comprenden limpieza de índices, replicación de datos de nodos, tareas programadas de mantenimiento, borrado de logs, entre otras tareas específicas para llevar un control correcto de la plataforma Hybris. Utilicé instancias EC2 debido a que estos procesos de control se alojan directamente en sistema operativo en este caso UNIX para dichas instancias.

Clúster de monitoreo EC2:

El clúster de monitoreo lo definí usando dos instancias EC2 siguiendo con la misma lógica de tener un rendimiento alto y tolerancia a fallos, al igual que en el clúster de control, el de monitoreo lo definí para tareas específicas de monitoreo de procesos de negocio, es decir para monitoreo de cada proceso interno de Hybris, de logs, de auditorías, de integridad de datos y de tareas programadas, este clúster monitorea procesos internos de la plataforma, no monitorea componentes de infraestructura hablando de hardware, para ese propósito definí otro componente que explicaré más adelante.

Instancias DB:

En este componente definí las base de datos de la plataforma que se usó para la plataforma de primera instancia elegí MySQL empresarial como base de datos relacional debido a su compatibilidad probada con Hybris además para proveer alta disponibilidad y redundancia de datos definí una instancia de *base de datos multi-AZ* dicha instancia funciona de tal forma que delega a AWS la creación automática de una instancia de base de datos principal y otra instancia de reserva, en la que se replican sincrónicamente los datos, en una zona de disponibilidad diferente, de esta manera aseguramos que la disponibilidad de la plataforma prevalezca aun si hay fallo en una de las instancias de bases de datos manejada por AWS.

Repositorio S3:

Utilicé este componente como un sistema de archivos (*file system*) para datos que no sean accedidos por la plataforma de forma constante, hablo de logs de procesos, reportes diarios, reportes de auditoria, etc. Este tipo de datos no se accede de forma constante por la plataforma por lo cual la instancia S3 funciona a la perfección y optimiza costos.

Repositorio elástico EFS:

Este repositorio lo definí con el propósito de almacenar los archivos multimedia que se acceden de forma constante en la plataforma, es decir los archivos multimedia (imágenes y videos de los productos), estos archivos son accedidos en alta demanda por la plataforma por lo cual requerí usar este componente de archivos que proporciona un escalamiento dinámico cuando la demanda de accesos aumenta, el repositorio EFS (*Elastic File System* por sus siglas en inglés) es manejado automáticamente por AWS con base a la configuración que hice en la consola de administración de la nube privada, de esta manera anticipe que el costo de la replicación de instancias sobrepasará el limite definido en la solución que diseñé.

Monitoreo *Cloud Watch*:

Definí este componente en la solución para hacer un monitoreo activo del hardware de la solución, es decir de las instancias virtuales en la nube. Este componente de AWS hace un monitoreo de la infraestructura a nivel hardware de toda la solución, es decir verifica el rendimiento del CPU de cada instancia declarada y envía alertas a los correos configurados en caso de detectar alguna irregularidad o alto consumo de recursos de las instancias en la nube privada de la solución.

Servidor de correo SES:

Definí este componente para enviar correos desde la plataforma (SES por sus siglas en inglés *Simple Email Service*), lo administré y configuré por medio de la consola de AWS de nube privada, su integración fue fácil de realizar y solo tuve que seguir la guía proporcionada por Amazon.

NAT Gateway:

Utilicé este componente de AWS para dar acceso a internet a las instancias EC2 del clúster de Hybris esto fue necesario para integrar Hybris con las plataformas de pagos y *Marketplace* sin embargo dicho componente previene el acceso de trafico de internet hacia la nube privada, es decir funciona en un solo sentido, de adentro (nube privada) hacia afuera (Internet).

Balanceador de carga, punto de entrada:

Este componente lo definí y configuré para proveer el acceso de los usuarios finales a la plataforma Hybris, es un balanceador de cargas que divide la cantidad de peticiones de manera equitativa entre las instancias de servidores web que definí en el clúster de Hybris.

Finalización y despliegue en producción

Una vez que definí y configuré el ambiente para producción sólo resto hacer el despliegue de cada iteración de acuerdo con la metodología en dicho ambiente utilizando el flujo de integración continua y despliegue continuo (CI y CD respectivamente) que ayudé a definí previamente en la implementación.

En cada liberación a producción di apoyo y soporte para asegurar un correcto despliegue además de ayudar a la categorización y diagnóstico de incidencias en el ambiente productivo cuando el equipo de desarrollo necesitó ayuda.

Con esto terminé mi participación en el proyecto.

Capítulo 4: Resultados

Resultados del sistema

El proyecto mencionado en el capítulo 3 se ejecutó de manera exitosa recibiendo una excelente retroalimentación del cliente, aunque en un principio los canales de comunicación fueron difíciles de abrir a medida que fue avanzando el proyecto se afino la comunicación y la información fluyo constantemente.

En cuanto a los resultados finales a continuación enlisto los resultados de la solución en producción en el primer mes de uso:

I. El sistema fue capaz de completar un flujo de compra de productos de principio a fin sin fallos
II. El sistema se integró exitosamente con las plataformas de pago y Marketplace.
III. Fue posible para los usuarios vendedores crear tiendas virtuales mediante la plataforma Mirakl y publicar sus productos en el portal de comercio electrónico de una forma sencilla y con una integración transparente.
IV. Fue posible para los usuarios compradores ver todos los productos publicados por los usuarios compradores tanto desde su dispositivo móvil como desde el portal web.
V. Los usuarios vendedores pudieron publicar sus ofertas de forma exitosa y añadirlas al catálogo de productos mostrándose de manera remarcada en resultados de búsquedas.
VI. Los usuarios compradores vieron reflejado en sus dispositivos móviles las notificaciones de las promociones publicadas por los usuarios vendedores de manera exitosa.
VII. Los usuarios compradores pudieron realizar búsquedas a través de todos los productos publicados por los vendedores de manera exitosa y con un tiempo de respuesta aceptado.
VIII. Los usuarios compradores fueron capaces de agregar productos a la funcionalidad <<*my store*>> para persistirlos en su cuenta a manera de lista de deseos en la plataforma.
IX. La plataforma presentó una interfaz amigable y fácil de usar tanto en el portal web como en la aplicación móvil.
X. La plataforma fue capaz de aceptar pagos tanto con tarjeta de crédito tradicional, *paypal*, *google pay* y *apple pay*.
XI. El usuario vendedor pudo realizar las tareas administrativas esperadas en la plataforma, tanto ver cuántos usuarios han comprado sus productos, así como un análisis de sus productos más vendidos y los usuarios que más visitaron sus productos.
XII. En términos de infraestructura el sistema presentó buenos resultados en las pruebas de estrés de carga pudiendo soportar hasta 2400 peticiones por día de manera simulada y hasta 100 peticiones por hora.

XIII. Aunque la plataforma se desarrolló y cumplió con todos los requerimientos funcionales y no funcionales, la implementación se retrasó en 4 meses a lo estimado, esto se debió en gran medida a peticiones de cambio del cliente que afectaron los tiempos de desarrollo.

XIV. Hubo definiciones que el cliente cambio después de haberse desarrollado lo que implicó una doble implementación y una negociación tanto de tiempos como de presupuesto para llegar que el proyecto siguiera siendo rentable para ambas partes.

XV. El sistema presentó algunos incidentes de severidad media a baja que debieron ser resueltos por el equipo de desarrollo, estos incidentes son parte del ciclo de desarrollo y no generaron un gran impacto en la funcionalidad general del sistema.

XVI. El proyecto se consideró exitoso por ambas partes y el cliente final se mostró satisfecho con miras a expandir el alcance y añadir robustez a la solución después del primer año de uso.

En términos generales y específicos la implementación de la solución fue exitosa y con muy buenos comentarios de todas las partes, hubo retrasos en la implementación y retrabajo por redefiniciones del cliente sin embargo el margen de ganancia no se vio afectado sustancialmente y el cliente se llevó una grata impresión con la solución propuesta e implementada.

Conclusiones

El comercio electrónico en México ha cobrado una gran relevancia hoy en día, a medida que la tecnología avanza las formas de comunicación y los canales se expanden pudieron entregar su mensaje a un público más variado y de todas las latitudes posibles.

Entre las principales ventajas del comercio electrónico encuentro las siguientes:

- Es posible llegar a una audiencia más amplia que con el modelo de ventas tradicional
- El costo de mantenimiento disminuye notablemente después del primer año de operación
- La inversión inicial es la única significativa, después de ello la inversión es mínima.
- El negocio puede cobrar relevancia internacional con las adecuaciones pertinentes a la plataforma de comercio.
- Se tiene un negocio funcionando 24/7 todo el año con el mínimo de personal operativo.
- Es posible realizar todos los procesos tradicionales de comercio en la plataforma virtual gracias al avance y madurez del comercio electrónico hoy en día.
- Se puede tener integración con las redes sociales y dar alcance a un público aún más amplio
- Se puede usar las herramientas de análisis de datos para seleccionar específicamente el público al que se quiere llegar con dicho comercio.

La evolución del desarrollo de software permite hoy en día un aumento en la rapidez de desarrollo y entrega de productos constante con las metodologías agiles, metodologías como Scrum permiten a un proyecto de desarrollo obtener una retroalimentación constante de parte del cliente final y en esa medida realizar ajustes a las tareas programadas.

Las metodologías agiles han llevado al desarrollo de software a una nueva etapa en la cual el nivel de exigencia aumenta y hay menos margen de error y aunque seguir una metodología al pie de la letra puede resultar tortuoso en un proyecto real, lo cierto es que tener una guía a la cual seguir siempre ayudará al proyecto a llegar a buen término.

Para poder realizar un desarrollo de calidad es necesario conocer en gran medida los requerimientos específicos del cliente, esto sólo es alcanzable teniendo una interacción directa y constante con sus preocupaciones y expectativas, en la medida en la que estos requerimientos sean obtenidos de la forma correcta se podrá lleva a cabo el desarrollo con los resultados esperados.

Obtener los requerimientos del cliente puede llegar a ser una tarea compleja si no se siguen lineamientos o guías para llevar a cabo las preguntas correctas, metodologías como la propuesta por el SEI son grandes aliadas en esta compleja tarea.

Es importante llevar un control de calidad exhaustivo en el desarrollo del software, este control es lo que dictaminara que el producto sea de calidad, las diversas herramientas que hoy en día se ofrecen como proyectos de uso libre son una excelente opción para tener dicho control, definir lineamientos de código y buenas prácticas también son puntos importantes para considerar en el aseguramiento de calidad del código. Así mismo, realizar una fase de pruebas apropiada descartara que los productos entregados al cliente contengan errores de ejecución.

La arquitectura de software es un rol de suma importancia en la solución de un sistema, llevar a cabo una correcta etapa de diseño será crucial para ejecutar una correcta implementación.

El rol de arquitecto en un proyecto involucra muchas responsabilidades entre las cuales destacan el diseño entero de la solución, investigación de tecnologías, asegurar la interacción de estas y verificar la compatibilidad de las integraciones cruciales. Además, el arquitecto en un proyecto está involucrado en todas las fases del desarrollo del software no limitándose únicamente a la parte de diseño, este rol está presente y participando desde que el proyecto se concibe hasta su puesta en producción asegurando a sí un correcto despliegue y un cumplimiento de todos los requerimientos tanto funcionales como no funcionales del sistema.

En específico en el proyecto planteado concluyo que la plataforma desarrollada cumplió las expectativas y se puede considerar un proyecto exitoso en lo general y particular, es importante mencionar que la elección de las plataformas correctas y la visión global del sistema ayudaron en gran medida para la implementación y diseño correctos.

La plataforma Hybris cumplió las expectativas con todos los procesos de comercio electrónico integrándose a su vez de manera correcta con las plataformas de *Marketplace,* de pagos y de notificaciones.

El proyecto fue exitoso en gran medida a la mezcla y ejecución correcta de las metodologías descritas en el capítulo 3. La metodología del SEI fue la columna vertebral de las tareas de diseño y arquitectura, ayudándome así a lograr una solución integral y viable en el tiempo especificado para dicho propósito.

La metodología del SEI también me ayudo a identificar claramente los diferentes componentes del sistema y aún más importante dicha metodología fue mi guía para analizarlos, darles solución y documentarlos de una forma correcta para el entendimiento de los diferentes involucrados de la solución.

Es importante mencionar también que también debido a la experiencia implementando la metodología SEI en este proyecto puedo concluir que en cada proyecto la metodología puede adaptarse a las circunstancias específicas sin perder su punto de vista y objetivo, es decir, en este proyecto pude usar todas las tareas descritas en dicha metodología, sin embargo, por mi experiencia en proyectos pasados puedo decir que no siempre hay tiempo, dinero o disponibilidad del cliente para

pasar por todas las etapas que propone el SEI para la definición de un diseño y arquitectura, por lo que concluyo que la metodología es la que se adapta al proyecto en cada ocasión y no viceversa.

De la implementación del proyecto puedo concluir que las metodologías agiles ayudan a tener una retroalimentación del cliente constante durante todo el ciclo de desarrollo, este factor afecta directamente la ejecución de tareas y prioridades durante la implementación repercutiendo de forma positiva si se prioriza de manera coherente dicha retroalimentación y se lleva un control de cambios adecuado y también puede repercutir de forma negativa si no se lleva una negociación adecuada con el cliente y con el alcance comprometido en el contrato inicial del proyecto.

Los atrasos sufridos durante este proyecto se debieron a una mala negociación de parte del cliente y cambios en rubros previamente establecidos una vez terminado el desarrollo de estos.

Tomando en cuenta lo anterior concluyo que una metodología ágil puede jugar en tu favor o en contra dependiendo de las capacidades de negociación de los administradores del proyecto y del compromiso del equipo de desarrollo en terminar sus tareas de forma adecuada y oportunamente. Es importante llevar un monitoreo constante del avance de las tareas en cada iteración y escalar cualquier tipo de bloqueo que se tenga de inmediato para evitar atrasos en el desarrollo.

Este proyecto me ayudó a pulir mis habilidades como arquitecto de software a la vez que fue parteaguas para consolidarme como arquitecto de solución pudiendo integrar diferentes plataformas con diferentes tecnologías para lograr un objetivo común, una plataforma de comercio electrónico con estándares de primer nivel.

Glosario

ADLs
Architecture Description Languages
Apache
Proveedor de servicios y marcos de trabajo de uso libre basados en el lenguaje java
API
Application Program Interface
Apple pay
Método de pago provisto por la compañía Apple
ASR's
Architecture Significant Requirements
B2B
Business to Business
B2C
Business to Commerce
Back-End
Desarrollo alojado en servidor de aplicación, no visible al usuario final
BPM
Business Process Management
CD
Continuos Delivery
Change Management
Gestión de cambios de ITIL
CI
Continuos Integration
COBOL
Lenguaje de Desarrollo legado
Command
Patrón de diseño de la programación orientada a objetos
CRM
Customer Relationship Management
Cronjob
Tarea programada
CSS
Cascading Style Sheets
CVS
Comma-Separated Values
CXF
Marco de trabajo de Apache para desarrollar servicios web
DAO
Data Access Object, patrón de diseño de software
Db2
Base de datos relacional de IBM

Documentum
Base de datos no relacional para almacenar archivos
Eclipse
Ambiente de Desarrollo integrado de uso libre
ERP
Enterprise Resource Planning
ESB
Enterprise Service Bus
Facade
Fachada, patrón de diseño de software
Factory pattern
Fabrica, patrón de diseño de software
Failover
Conmutación por fallo, tolerancia a fallos de un sistema
Framework
Marco de trabajo de software o de una metodología
Front-end
Desarrollo alojado en servidor web, visible al usuario final
FTP
File Transfer Protocol
Google pay
Método de pago de Google
Hibernate
Marco de trabajo de software para acceso a datos
Incident Management
Gestión de incidentes de ITIL
IntelliJ
Ambiente de Desarrollo integrado de uso libre
ITIL
Information Technology Infrastructure Library
Java
Lenguaje de desarrollo
Javascript
Lenguaje de Desarrollo interpretado
Jenkins
Herramienta de integración de código y despliegue
JMS
Java Message Service
JSP
Java Server Page
Maven
Herramienta de construcción de software
Middleware
Capa intermedia en una arquitectura de software
MVC
Model, View Controller, patrón de diseño de software

NET
Marco de trabajo de software por Microsoft
OOTB
Out Of The Box, refiere a una característica incluida en la plataforma
ORM
Object-relational mapping
PaaS
Platform as a Service
Paypal
Método de pago electrónico
Pipeline
Flujo de ejecución de una tarea en Jenkins
PL/SQL
Lenguaje estructurado de bases de datos
Portlets
Componente basado en java para front-end
Problem Management
Gestión de problemas de ITIL
QAW
Quality Attribute Workshop
REST
Representational State Transfer
RM
Resource Manager
SDK
Software development kit
SFTP
Secure File Transfer Protocol
Shell
Línea de comandos de un servidor
SLA
Service Level Agreement
SOA
Service Oriented Architecture
SOAP
Simple Object Access Protocol
Solr
Servidor motor de búsqueda de Apache
Sonar Qube
Herramienta de calidad de código
Spring
Marco de trabajo de java para integración
SQL
Simple Query Language
Strategy
Patrón de diseño de software

Tibco
Proveedor de servicios de internet empresarial
Tuxedo
Servidor de integración middleware de Oracle
UML
Unified Modeling language
Webhooks
Mecanismo de transferencia de mensajes en aplicaciones web
Webphere
Servidor de aplicaciones de IBM

Referencias

Bass, L., Clements, P. and Kazman, R. (2012). *Software architecture in practice*. 3rd ed. United States: Addison-Wesley.

Clements, P. (2014). *Documenting software architectures*. 2nd ed. Upper Saddle River, NJ: Addison-Wesley.

Sutherland, J. (2015). *Scrum: The art of doing twice the work in half of time*. 1st ed. London: Random House Business Books.

Pcisecuritystandards.org. (2018). *Official PCI Security Standards Council Site*. [online] Disponible en: https://www.pcisecuritystandards.org/pci_security/ [Acceso 15 oct. 2018].

Introducing Jenkins X. (2018). *CI/CD solution for modern cloud applications*. [online] Available at: https://jenkins.io/blog/2018/03/19/introducing-jenkins-x/ [Acceso 20 Oct. 2018].

braintreepayments.com. (2018). *Braintree Support Articles*. [online] Available at: https://articles.braintreepayments.com/ [Acceso 27 Oct. 2018].

Mirakl, the Marketplace Company. (2018). *SaaS Marketplace Platform for B2C E-Commerce - Mirakl*. [online] Available at: https://www.mirakl.com/b2c/ [Acceso 29 Oct. 2018].

User, S. (2018). *Asociación de Internet Inicio*. [online] Asociaciondeinternet.mx. Available at: https://www.asociaciondeinternet.mx/es/ [Acceso 30 Oct. 2018].