UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMA DE RECEPCIÓN DE INFORMACIÓN (SIRI)

INFORME DE TRABAJO PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA

CARLOS GONZÁLEZ ZARCO

Asesor: **M.I. María Del Socorro Guevara Rodríguez**

Ciudad Universitaria, México, D.F. Noviembre del 2015

ÍNDICE

Tabla de Ilustraciones

INTRODUCCIÓN

En el siguiente informe se presentan las actividades que realicé como desarrollador de sistemas en la empresa GETechnologies. Explicaré a detalle mi participación en el proyecto Sistema De Recepción De Información (SIRI) para la empresa PROCESAR.

Durante mi trayectoria profesional he colaborado con varias empresas, la mayoría de las veces desarrollando proyectos para dependencias gubernamentales tales como el Instituto Mexicano del Seguro Social (IMSS), Servicio de Administración Tributaria (SAT), Sistema de Ahorro para el Retiro (SAR), entre otros. Mi enfoque en general ha sido hacia la capa de negocios, en este proyecto mi participación estuvo limitada al desarrollo de componentes y sus respectivas pruebas usando como base la plataforma Java Enterprise Edition (JEE) 5.

En los siguientes capítulos daré una breve explicación de lo que hace la empresa GETechnologies y su relación con la empresa PROCESAR. Explicaré de qué se trata el proyecto SIRI y los beneficios que ha aportado a las dependencias de gobierno que tienen a sus empleados afiliados al Instituto de Seguridad y Servicios Sociales de los Trabajadores del Estado (ISSSTE).

Explicaré a detalle las tecnologías utilizadas durante el proyecto y las ventajas y desventajas que se presentaron en su implementación, así como las mejores prácticas y metodologías que utilizamos.

Finalmente presentaré los resultados de la puesta en producción del sistema. Cabe destacar que es sólo una fase y que por la sensibilidad de la información tratada, no se verán detalles de la arquitectura ni de la seguridad del sistema.

CAPÍTULO I: Descripción de la empresa

GETechnologies

GETechnologies es una joven empresa que toma como eje las habilidades comprobadas de sus socios fundadores para aprovechar la creciente necesidad de desarrollo de proyectos de Tecnologías de la Información (TI). Sus integrantes aportan al negocio una experiencia de más de dos décadas de dirección de proyectos exitosos, en los que el común denominador ha sido el ejercicio de una gestión proactiva, la que no se limita a la administración de las actividades y la documentación de las desviaciones, sino, por el contrario pone el énfasis en la mitigación de los riesgos y en la solución de conflictos.

La empresa ha desarrollado cuatro líneas de acción que permiten integrarnos como un agente de solución en cada fase del ciclo de vida de las aplicaciones:

Arquitectura de Negocio. Ofreciendo a nuestros clientes asesoría en la identificación y expresión de sus requerimientos, emprender iniciativas de reingeniería o refinación a sus procesos, esto orientado al correcto uso de las tecnologías lo que garantiza que las definiciones cuenten con elementos suficientes para el proceso de construcción.

Servicios:

- Evaluación y refinamiento de procesos operativos
- Identificación de proyectos y elaboración de planes para su ejecución
- Análisis de Fuerzas, Oportunidades, Debilidades y Amenazas (FODA) para las áreas de TI

Arquitectura de software. En la que se brinda un marco tecnológico a las necesidades de negocio, se acompaña en el proceso de implementación de arquitecturas y la sistematización del ciclo de gobierno de las aplicaciones.

Servicios:

- Definición del entorno para la automatización de procesos
- Diseño técnico con estándares del Lenguaje Unificado de Modelado (UML)
- Implementación de arquitectura orientada a servicios
- Sistematización del ciclo de vida de las aplicaciones
- Control y aseguramiento de calidad

Arquitectura de datos. Entendemos el valor que tienen los datos para la organización y por eso orientamos los esfuerzos en establecer los esquemas de modelado y diseño físico que garanticen el máximo de integridad con el mayor performance.

Servicios:

- Modelado de bases de datos
- Implementación de bases de datos Oracle
- Estrategia de respaldos y Plan de Recuperación ante Desastres (DRP)
- Migración, limpieza y data profiling

Visión

Convertirnos en proveedores de confianza de clientes claves en el sector financiero y de seguridad social. Ser líderes en la formación de capital humano que nos diferencie de nuestros competidores por conformar equipos de talento que ofrezcan alternativas reales de solución.

Misión

Generar los vínculos que permitan el tránsito eficiente de las necesidades de nuestros clientes a bienes informáticos de alta calidad.

Objetivos

Realizar funciones que aseguren el éxito de los proyectos de TI en las empresas en las que es crítico el uso de la tecnología.

Dentro de estos servicios encontramos la Arquitectura de Aplicaciones, la Arquitectura de Información, la Gestión de Proyectos Informáticos y la Arquitectura Orientada a Servicios (SOA).

Se pretende ofrecer los servicios como proyectos predecesores de los esfuerzos de implementación o bien como proyectos paralelos e integrando a los equipos de implementación, de planeación y usuarios finales.

Cuando durante el proyecto se detecten necesidades adicionales pero que son factores críticos para el cumplimiento del alcance, se cuenta con las relaciones de negocio que permiten integrar socios que cubren las áreas descubiertas.

PROCESAR

Es la empresa operadora de la Base de Datos Nacional SAR. Dicha base, está conformada por la información procedente de los sistemas de ahorro para el retiro. Contiene la información individual de cada trabajador y el nombre de la administradora o institución de crédito en que cada uno de éstos se encuentra afiliado, llevando una certificación de los registros y del control de los procesos de recaudación y traspasos del sistema. La prestación de este servicio público se lleva a cabo por Procesar, S.A. de C.V., quien goza de la concesión que le fuera otorgada por el gobierno federal a través de la Secretaría de Hacienda y Crédito Público.

Otra de las particularidades del sistema mexicano de pensiones, radica en el proceso de recaudación centralizada, diseñado de acuerdo con los avances tecnológicos que posee el sistema financiero mexicano. Una de sus principales contribuciones ha sido la creación del Sistema Único de Autodeterminación (SUA) el cual ha permitido procesar la información de la recaudación con bajos costos, así como garantizar la seguridad de la misma.

El sistema centralizado de recaudación opera a través de las instituciones bancarias autorizadas como entidades receptoras, que actúan por cuenta y orden de los institutos de seguridad social. Gracias a la utilización de la poderosa plataforma de procesamiento electrónico de datos con los que cuenta Procesar, el proceso de recaudación, dispersión y liquidación de la información hacia las Afore se realiza en muy corto tiempo y las aportaciones ganan intereses a partir del día siguiente de su depósito en las entidades receptoras.

La tecnología empleada en el proceso de recaudación, constituye una innovación del sistema mexicano de pensiones, y ha contribuido a la optimización del procesamiento de datos, lo que le ha hecho merecer elogios a nivel mundial

CAPÍTULO II: Descripción de mí puesto de trabajo

Mi puesto en este proyecto fue el de Desarrollador Java Enterprise Edition Senior. El periodo en el que participé fue de Febrero del 2011 a Febrero del 2012. Mis actividades principales fueron:

- Revisar la documentación de los analistas de negocio y seguir las políticas de arquitectura para la construcción del software
- Documentar en código el propósito de las clases y archivos de configuración
- Ejecutar pruebas funcionales y de integración en la fase de desarrollo y Pruebas de Calidad
- Monitorear el comportamiento del sistema en pruebas con el cliente
- Corregir errores en producción
- Revisar la calidad del código continuamente para cumplir con los estándares acordados en el contrato
- Atender las reuniones de seguimiento con los equipos de bases de datos y arquitectura
- Monitorear el rendimiento de la base de datos para optimización de consultas y creación de nuevos índices

Mi entrada al proyecto fue en una etapa intermedia de su ciclo de vida. La arquitectura ya había sido definida y probada y restaba agregar los servicios de negocio faltantes así como validar su integración con el sistema existente.

Las pruebas con el cliente sirvieron para refinar algunos requerimientos y realizar ajustes al sistema que no habían sido considerados en un inicio. Dichos cambios exigieron un mayor esfuerzo del equipo ya que las fechas de contrato en proyectos gubernamentales suelen ser muy exigentes.

El mayor reto fue la integración final del sistema con el resto de las aplicaciones que el cliente había desarrollado o adquirido. Para esta integración se requirió de una selección de ingenieros de las diferentes plataformas que pudieran dar su diagnóstico experto cuando algo fallaba. El flujo de información a través de estos sistemas hacía especialmente difícil el rastreo de errores en los procesos. A pesar de que muchos de estos procesos tenían la capacidad de reiniciarse automáticamente, algunos requerían una intervención manual y de forma coordinada. Cabe mencionar que estos procesos son muy estrictos debido a que se trata de transferencias bancarias en lotes que de no ocurrir en una fecha determinada pueden ser causa de intereses y recargos.

CAPÍTULO III: Descripción de mi participación en la empresa

SIRI

Sistema de Recepción de Información

ANTECEDENTES

La preocupación del Estado Mexicano por la seguridad social y los derechos laborales, ha estado presente en las políticas y acciones gubernamentales desde 1917, cuando se consignaron estos derechos como normas supremas a nuestra Constitución Política.

En 1943 se creó el Instituto Mexicano del Seguro Social (IMSS), con el objetivo de ofrecer servicios de salud y de seguridad social a los trabajadores de nuestro país. Al año siguiente este Instituto implementó el seguro de Invalidez, Vejez, Cesantía en Edad Avanzada y Muerte (IVCM), esquema que estaba diseñado para operar como un sistema de pensiones de beneficios definidos, invirtiéndose de acuerdo a la Ley las reservas que se generaban año con año, para que llegado el retiro de los trabajadores, se pudiera cumplir con el pago de sus pensiones.

El seguro de IVCM estaba basado en un esquema de reparto, por lo que las cuotas de los trabajadores y patrones debían servir para otorgar beneficios directos a los jubilados, a través de las pensiones. El excedente de los recursos se utilizó para beneficiar a la población asegurada; así como para subsidiar otros seguros, como el de enfermedades y maternidad, que presentaban un déficit en su operación.

El siguiente gran avance se dio en el año de 1992, cuando se puso en práctica el Sistema de Ahorro para el Retiro, como seguro complementario al de IVCM. Bajo este esquema el patrón realizaba la apertura, a cada trabajador, de una cuenta individual en una institución de crédito; el empleador depositaba el 2% del salario base de cotización del trabajador, y los recursos que se acumulaban se entregaban en una sola exhibición al trabajador al momento de su retiro. Con este esquema quedaron perfectamente definidos los derechos de propiedad de estos recursos.

Conforme avanzaba el antiguo sistema, varios problemas se hicieron evidentes, reflejándose claramente en un desequilibrio financiero para el Instituto, y en la inviabilidad de mantener este esquema en el mediano y largo plazo. Asimismo, era claro que el Instituto tenía mermada su capacidad de inversión para seguir ampliando su infraestructura, cobertura y servicios, así como dificultades cada vez mayores para operar eficazmente algunos de sus ramos de seguro. Es por ello, que se decidió emprender un proceso profundo de revisión del IMSS. De este análisis surgió un debate muy amplio, con las más diversas opiniones de los sectores obrero, gubernamental y patronal, que constituyeron una alianza para el fortalecimiento y modernización del IMSS, elaborándose un diagnóstico sereno y objetivo promovido por el propio Instituto.

La nueva Ley del Seguro Social, fue publicada el 21 de diciembre de 1995, y entró en vigor el 1 de julio de 1997, se marca el inicio de una de las reformas más trascendentales en la seguridad social mexicana. A través de esta legislación, los cuatro ramos de aseguramiento que administraba el IMSS: Enfermedades y Maternidad; Riesgos de Trabajo; Guarderías para hijos de aseguradas; e Invalidez, Vejez, Cesantía en Edad Avanzada y Muerte, sufren cambios para dar lugar a cinco ramos de aseguramiento en la actualidad, cuyas principales adecuaciones consistieron en dividir al ramo de IVCM en dos: Invalidez y Vida (IV) y, Retiro, Cesantía en Edad Avanzada y Vejez (RCV), con base en un esquema de ahorro y capitalización individual, además de ampliar la prestación emanada del ramo de Guarderías e incorporarle a éste, Prestaciones Sociales.

En el nuevo Sistema de Pensiones cada trabajador tiene una cuenta individual abierta de manera personal en instituciones financieras de giro exclusivo denominadas Administradoras de Fondos para el Retiro (Afores), donde se acumulan sus aportaciones junto con las de sus patrones y el Gobierno Federal.

La cuenta individual, además de recibir las aportaciones del seguro de RCV, tiene dos subcuentas más: la de Vivienda y la de Aportaciones Voluntarias. La primera, registra las aportaciones que realiza el patrón, equivalente al 5 por ciento del Salario Base de Cotización y son administradas por el Instituto del Fondo Nacional de la Vivienda para los Trabajadores (INFONAVIT), por lo que las Afores solo toman nota e informan a sus afiliados de los movimientos de dicha subcuenta y de los intereses que el Instituto determine. La subcuenta de Aportaciones Voluntarias se conforma con los depósitos que los trabajadores y en su caso, los patrones realizan de manera adicional a los obligados por la Ley.

Los recursos correspondientes a las subcuentas de RCV y Aportaciones Voluntarias son invertidos a través de las Sociedades de Inversión Especializadas de Fondos para el Retiro (Siefores) que administran las Afores, las que buscan obtener, bajo las más estrictas condiciones de seguridad, los mejores rendimientos para este importante ahorro.

CÓMO FUNCIONA

Ilustración 1 Funcionamiento del ciclo de fondos

DEFINICIÓN DEL PROYECTO

El sistema tiene como objetivo, en su fase inicial, dar soporte a la operación del SAR para ISSSTE. Sin embargo, debe ser lo suficientemente flexible para poderse escalar y modificar en un futuro próximo y así atender las necesidades de IMSS, Instituto de Salud del Estado de México (ISEM), etc.

Además de la necesidad de adaptación, el sistema debe soportar gran número de transacciones concurrentes y volúmenes de datos muy grandes. Aunado a esto, algunos de sus procesos de negocio requieren información de otros sistemas, tales como el Registro Nacional de Población (RENAPO), bancos, aseguradoras, Nacional Financiera (NAFIN), etc.

La información que ingresará al sistema, se hará por distintas vías. Mucha de esta información forma parte de otras bases de datos y será migrada en su mayoría. Existen también procesos, que debido al gran volumen de datos, envían sus datos en lote dentro de un archivo de texto plano. La vía de entrada más importante es el portal SIRI. A través de él los usuarios podrán realizar operaciones tales como:

- Altas, Bajas y Modificaciones de datos de trabajadores: Domicilios, Clave Única de Registro de Población (CURP), Sueldos, etc.
- Altas, Bajas y Modificaciones de centros de pago
- Altas, Bajas y Modificaciones de catálogos
- Solicitudes de líneas de captura
- Notificación de pago de líneas de captura
- Consulta de movimientos
- Generación de reportes estadísticos
- Entre otras…

Algunas operaciones requieren respuesta inmediata (síncrona) y algunas no (asíncrona). Por lo tanto, se requieren mecanismos de ejecución de tareas en segundo plano. También se deben implementar mecanismos para ejecución de tareas por lotes (batch) y manejo de caché debido a la alta demanda en transacciones a la base de datos.

Por todo esto, la arquitectura, metodologías, tecnologías y herramientas empleadas deben elegirse con mucho cuidado. A continuación se describen cada una de las elecciones que se hicieron para este proyecto y su relevancia en el mismo.

MEJORES PRÁCTICAS EN EL DESARROLLO DE SOFTWARE

A través de mi experiencia en diferentes proyectos de desarrollo de sistemas puedo enumerar y describir las ventajas de las herramientas utilizadas durante este proyecto, a pesar de haber ingresado en una etapa avanzada del mismo y por lo tanto no haber sido parte en la toma de decisiones.

La historia reciente en la industria de software nos ha enseñado que la única constante es el cambio. Por lo que hay que ser conscientes de que los sistemas que construimos irán evolucionando y cambiando. Este ciclo de vida de los sistemas se ve afectado también por el dinamismo en las personas responsables del desarrollo y mantenimiento. Cada vez es más frecuente ver equipos distribuidos en diferentes zonas geográficas colaborando, lo que agrega un grado más de complejidad en los proyectos modernos.

Es por esto que la elección de herramientas y tecnologías debe seguir estándares globales y mantener procesos de comunicación que no se vean severamente afectados por los diferentes rasgos culturales de los individuos que colaboran en el proyecto.

UML es un lenguaje de modelado ampliamente aceptado en todo el mundo debido a su simpleza y expresividad para modelar sistemas complejos. Gracias a sus diferentes tipos de diagramas se puede analizar un sistema desde diferentes perspectivas y visualizar como los diferentes componentes interactúan y contribuyen al resultado final.

Realizar un diseño previo, así sea una versión no exhaustiva, puede reducir en gran medida el riesgo en los malos entendidos que ocurren en proyectos de software. Estos diseños se convierten en acuerdos que los diferentes equipos involucrados siguen para la construcción del sistema.

UML es además agnóstico del lenguaje de programación aunque su enfoque es orientado a objetos. En el desarrollo de sistemas empresariales podemos claramente nombrar a las 2 grandes plataformas de desarrollo: Java y .Net.

Java sin embargo, entendido como plataforma y no solo como lenguaje de programación, goza de una mayor comunidad que se ha beneficiado de la apertura y flexibilidad que no tiene .Net debido a antiguas políticas proteccionistas de Microsoft.

En general seguir el paradigma de programación orientado a objetos beneficia a largo plazo al desarrollo de sistemas. Estos lenguajes obligan al programador a

desarrollar pensando a largo plazo y no solo en la solución inmediata al problema. Se favorece la reutilización, modularidad y claridad de los programas, lo que se traduce en una mayor facilidad para la extensión o modificación del sistema sin afectar gravemente la totalidad del mismo.

Hay que tener en cuenta también que los sistemas empresariales llegan a componerse de miles de archivos y millones de líneas de código. Analizar y entender un sistema como estos no es tarea sencilla sin las herramientas adecuadas. Los ambientes de desarrollo integrados (IDE por sus siglas en inglés) son una herramienta que puede mejorar la productividad de los programadores ya que contienen todas las utilidades que normalmente se necesitan.

Eclipse es tal vez el IDE más utilizado para Java, su licencia gratuita y su flexibilidad para la integración de nuevos plugins (gratuitos también en su mayoría) lo hacen muy atractivo.

Cuando hablamos de sistemas complejos que involucran mucho personal colaborando al mismo tiempo, debemos tener en cuenta que ocurrirán muchos conflictos para integrar el trabajo de cada uno. Otra cosa que ocurrirá con frecuencia es la revisión del historial de trabajo y en ocasiones, desechar parte de lo hecho en favor de una mejor versión o idea.

Herramientas de Control de Versión de Sistemas son indispensables para mantener ese orden en los proyectos. Subversion es un sistema que ha madurado y demostrado su beneficio en los equipos de desarrollo. El dominio de uso por parte de la gran mayoría de programadores lo hace una buena apuesta cuando se elige un CVS.

El dinamismo de los proyectos obliga a buscar siempre herramientas que mejoren la productividad de los equipos. Es por eso que la automatización de tareas repetitivas es fundamental para reducir el tiempo invertido en actividades que no contribuyen directamente en el avance del proyecto.

Maven es una herramienta abierta y fácilmente extensible que permite organizar el código de un proyecto y automatizar las tareas cotidianas en la construcción de sistemas. Tareas como compilar, documentar, empaquetar, probar, instalar, entre otras, son fácilmente automatizables gracias a Maven.

Errores en la compilación de un módulo del proyecto puede impactar gravemente la totalidad del sistema y comprometer el avance y la calidad del entregable final. Es por esto que una muy buena práctica de desarrollo es la integración continua de los cambios y la validación frecuente de que el sistema es estable.

Los errores en los sistemas son más costosos entre más avanzado está el proyecto. Lo mejor es detectar y corregir estos errores en las etapas más tempranas. Una buena estrategia es implementar pruebas unitarias exhaustivas en cada línea de código que se genere. JUnit es el framework de pruebas más poderoso y confiable que existe para Java actualmente.

Conforme se desarrollan más y más componentes, es fácil darse cuenta de que hay características comunes a muchos sistemas que podrían reutilizarse y evitar reinventar la rueda. Los EJBs nacen con la idea de abstraer funcionalidades comunes en sistemas empresariales como cuestiones de seguridad, monitoreo, escalabilidad, etc.

Utilizar las funcionalidades ya implementadas en el contenedor de las aplicaciones permite que los equipos de desarrollo se enfoquen en construir componentes específicos del negocio y no desperdicien tiempo rehaciendo cosas una y otra vez.

En ocasiones la reutilización no es tan simple, por ejemplo, cuando una empresa compra a otra empresa, y por ende a sus sistemas. La empresa comprada puede tener sus sistemas en una tecnología completamente distinta y hacer su integración casi imposible con nuestras políticas.

Los WebServices son una opción muy utilizada para integrar sistemas construidos en plataformas de desarrollo distintas y que permiten construir encima de estas plataformas nuevos procesos de negocio.

Esta integración es algo para lo que tal vez no estaban preparados los sistemas originalmente y en su diseño inicial se decidió utilizar un repositorio de datos muy simple. Conforme los sistemas van creciendo y teniendo más demanda, es necesario escalar las bases de datos para seguir garantizando un nivel de servicio aceptable para los clientes. Oracle es una de las mejores bases de datos en el mercado y nos brinda las herramientas para escalar nuestra información con seguridad.

ANÁLISIS Y DISEÑO

La fase de análisis consiste en la definición del alcance del proyecto, así como la estimación de tiempos y costos. En esta fase se definen también los requerimientos funcionales y no funcionales del sistema.

El objetivo de la primera fase del SIRI es dar soporte a las operaciones del SAR para el sistema ISSSTE. Sin embargo se debe tener en cuenta que este sistema funcionará como sistema piloto que posteriormente se adaptará al modelo de negocio de otros sistemas como el IMSS. Esto nos obliga a analizar las operaciones comunes a estos sistemas y hacer un modelo básico que pueda reutilizarse y extenderse sin mucho esfuerzo.

Requerimientos Funcionales

Los requerimientos funcionales son todas aquellas funciones que el sistema debe hacer como parte de las exigencias del negocio. Se definen las entradas, comportamientos y salidas del sistema. Estos requerimientos normalmente provienen de las reglas de negocio del cliente.

Requerimientos No Funcionales

También conocidos como requerimientos de calidad, son aquellos procesos por los que se evalúan las operaciones del sistema. Entre estos procesos se encuentran la seguridad, rendimiento y confiabilidad del sistema.

Si bien los requerimientos funcionales del SIRI son importantes, no lo son menos los no funcionales. La base de datos de trabajadores y pensionados incrementará año con año. La legislación de la Comisión Nacional del Sistema de Ahorro para el Retiro (CONSAR) está en constante evolución y con ello las reglas de negocio que el sistema debe cumplir.

La información de sueldos, aportaciones, fondos de vivienda, retiro, etc. Es de una sensibilidad tal que el sistema debe garantizar la seguridad e integridad de dicha información.

Se debe considerar también que existen días de mayor demanda. El sistema no puede estar fuera de servicio en esos momentos porque esto puede significar retrasos en las operaciones de los clientes y con ello, generación de intereses por retrasos en pagos.

Modelo del dominio

Es un modelo conceptual de las entidades que estarán involucradas en el sistema. En este modelo se definen sus atributos, comportamientos, restricciones y relaciones entre las entidades. Lo más importante de este modelo es que nos permite definir el alcance del proyecto y crear una representación abstracta de lo que el analista entiende, pero que a su vez pueda ser entendida y validada por el cliente.

El modelo del dominio provee una vista estática del sistema, el cual puede ser complementado con una vista dinámica llamada Casos de Uso.

UML

Lenguaje Unificado de Modelado *(Unified Modeling Language)* es un lenguaje gráfico para el modelado de sistemas. Nos sirve para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Desde su aparición a finales de los 90, se ha convertido en el estándar para desarrollo de sistemas orientados a objetos. Su creciente aceptación ha contribuido al desarrollo de herramientas que facilitan su uso y permiten la generación de código. Esta funcionalidad es muy importante ya que rara vez el diseño inicial permanece inmutable durante el proyecto, por lo que necesitamos de herramientas que nos permitan tener una sincronía entre los cambios que se hacen en código contra lo que se encuentra en el diseño.

La importancia de UML en la fase de análisis y diseño radica en la conexión que crea entre los expertos en el negocio y los desarrolladores. Los diferentes diagramas de UML nos permiten visualizar el sistema desde diferentes perspectivas. Al tener un lenguaje común, podemos garantizar el cumplimiento de las expectativas del cliente.

Diagramas UML

Así como la construcción de un edificio requiere de diversos planos (instalación eléctrica, hidráulica, etc.) Los sistemas informáticos también requieren de diferentes diagramas que nos permitan analizar el problema desde distintas perspectivas.

A continuación se describen los diagramas más utilizados durante el proyecto.

Diagrama de clases

Es un diagrama estático que nos permite conocer la estructura de una clase y su relación con otras clases.

Una clase es una categoría que contiene atributos y comportamientos similares.

Se representa con un rectángulo dividido en tres partes. La primera parte contiene el nombre de la clase, la parte central contiene los atributos y la parte final contiene las acciones o comportamientos.

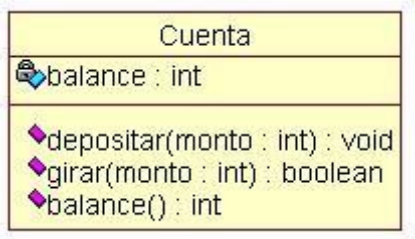


Ilustración 2 Diagrama de clase

Asociación

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si. Se denota con una línea solida entre dos clases

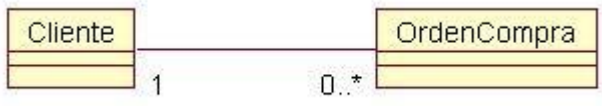


Ilustración 3 Diagrama de asociación

Cardinalidad

En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

- Uno o muchos: 1..* (1..n)
- 0 o muchos: 0..* (0..n)
- Número fijo: m (m denota el número).

Agregación y Composición

Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos con otros objetos, se tienen:

Agregación es una relación que implica que una clase contiene a otra clase independiente. Si la clase contenedora deja de existir, la clase contenida puede seguir existiendo

Composición, a diferencia de la agregación, si la clase contenedora deja de existir, la clase contenida no tiene razón para seguir existiendo.

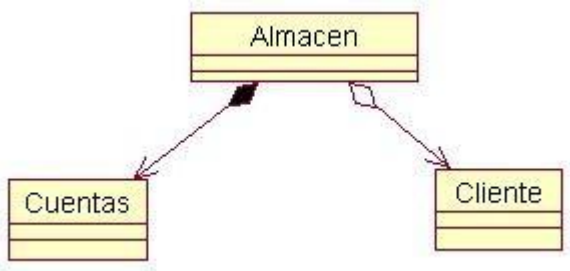
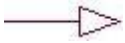


Ilustración 4 Diagrama de Composición

La relación entre Almacén y Cliente es una agregación, mientras que la relación entre Almacén y Cuentas es una composición.

Herencia (Especialización/Generalización)

Indica que una subclase hereda los métodos y atributos especificados por una Súper Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Súper Clase

Se denota por una flecha con un triángulo cerrado vacío del lado de la clase padre. 

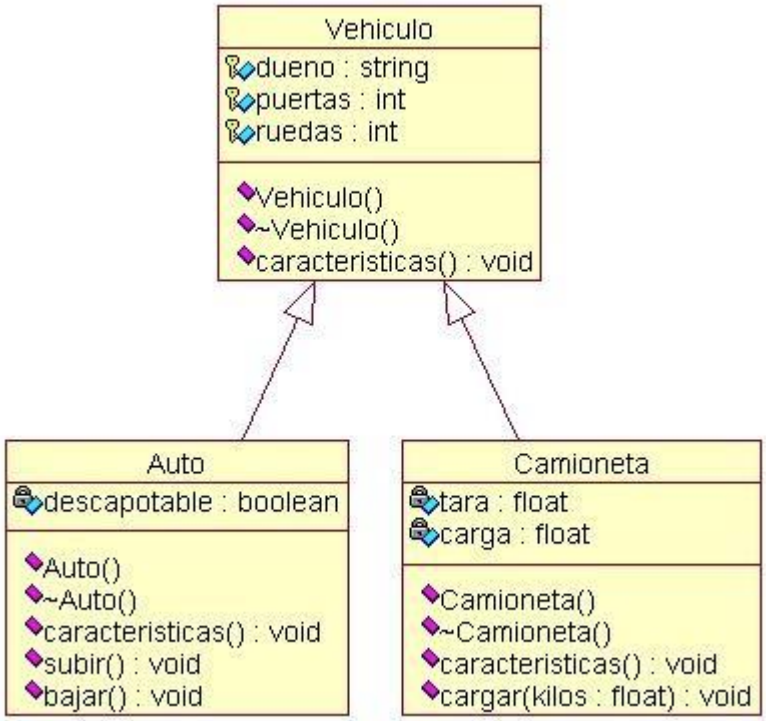


Ilustración 5 Diagrama de herencia

Diagrama de componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

Debido a que los diagramas de componentes son más parecidos a los diagramas de casos de usos, éstos son utilizados para modelar la vista estática y dinámica de

un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Cada diagrama describe un apartado del sistema.

En él se situarán librerías, tablas, archivos, ejecutables y documentos que formen parte del sistema.

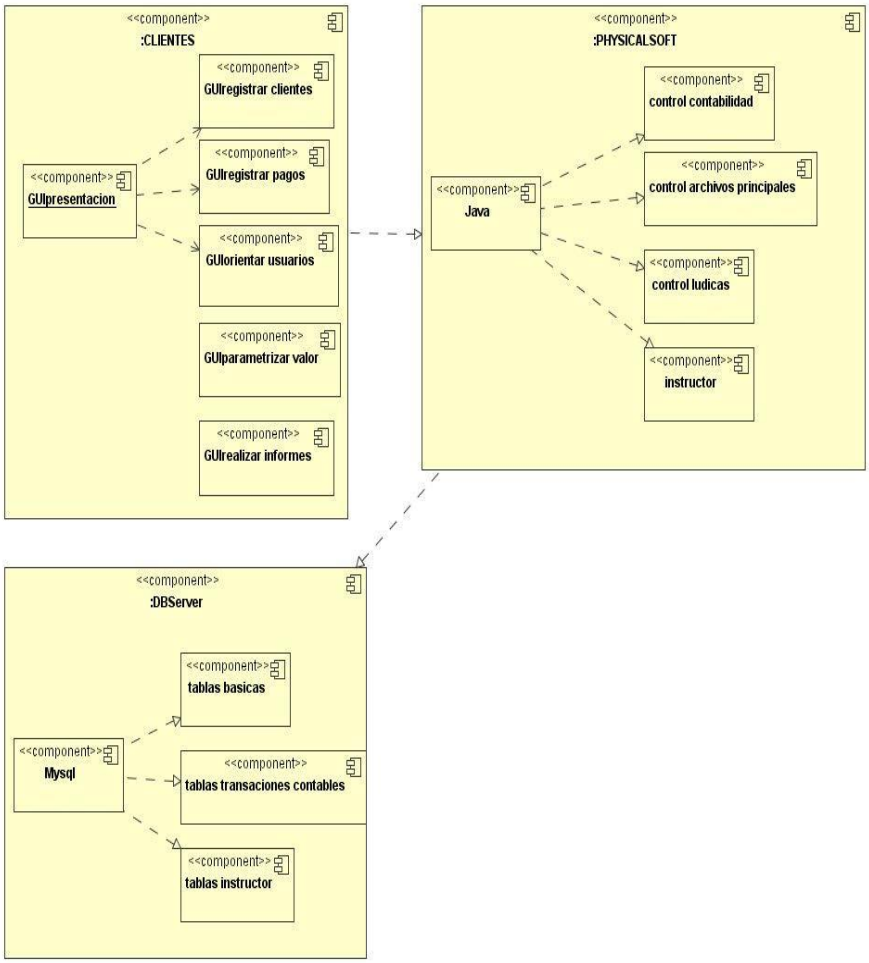


Ilustración 6 Diagrama de Componentes

Diagrama de Caso de Uso

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participarán en un caso de uso se denominan actores.

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

Los requerimientos funcionales se representan mediante estos diagramas.

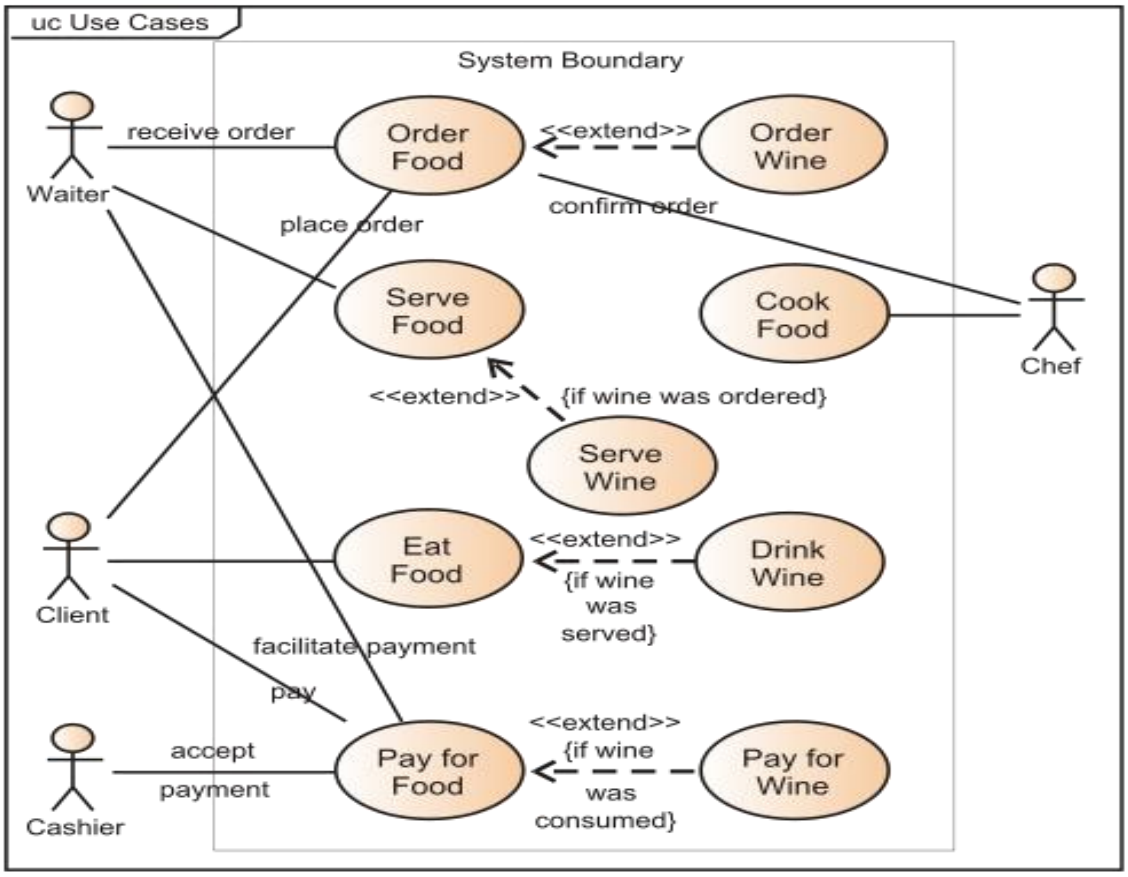


Ilustración 7 Diagrama de Caso de Uso

Diagrama de secuencia

Muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista abstracta del escenario, el diagrama de

secuencia contiene detalles de implementación, incluyendo los objetos y clases que se usan para implementarlo, así como los mensajes intercambiados entre los objetos.

Típicamente se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se dispone de la descripción de cada caso de uso como una secuencia de varios pasos, entonces se puede "caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos. Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales.

Existen dos tipos de mensajes: sincrónicos y asincrónicos. Los mensajes sincrónicos se corresponden con llamadas a métodos del objeto que recibe el mensaje. El objeto que envía el mensaje queda bloqueado hasta que termina la llamada. Este tipo de mensajes se representan con flechas con la cabeza llena. Los mensajes asincrónicos terminan inmediatamente, y crean un nuevo hilo de ejecución dentro de la secuencia. Se representan con flechas con la cabeza abierta.

También se representa la respuesta a un mensaje con una flecha discontinua.

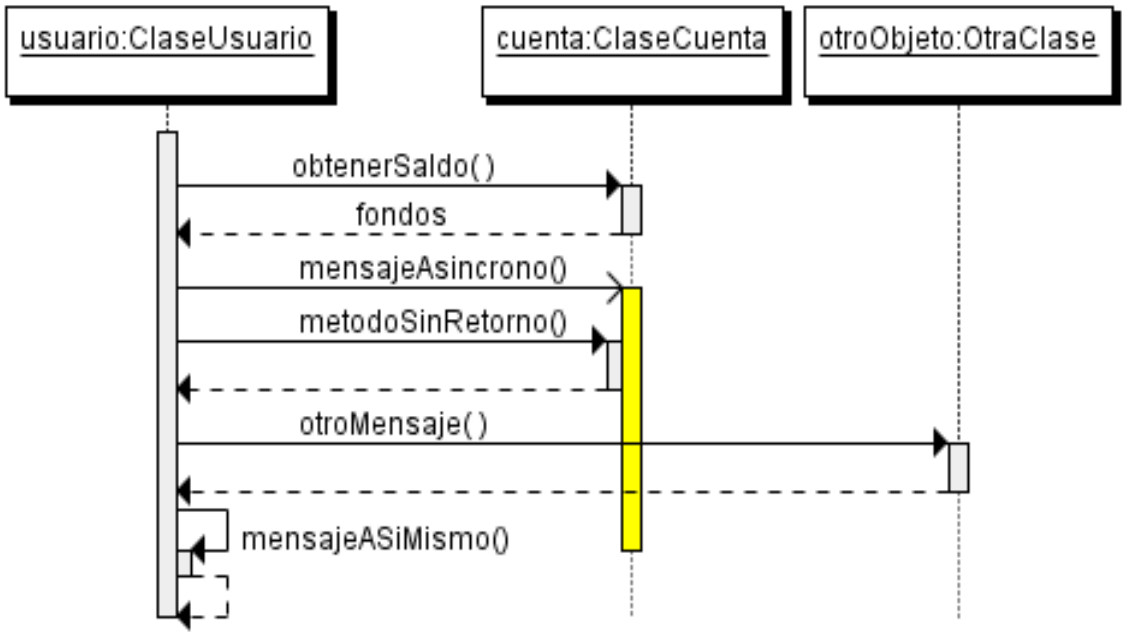


Ilustración 8 Diagrama de secuencia

Diagrama de despliegue de la aplicación

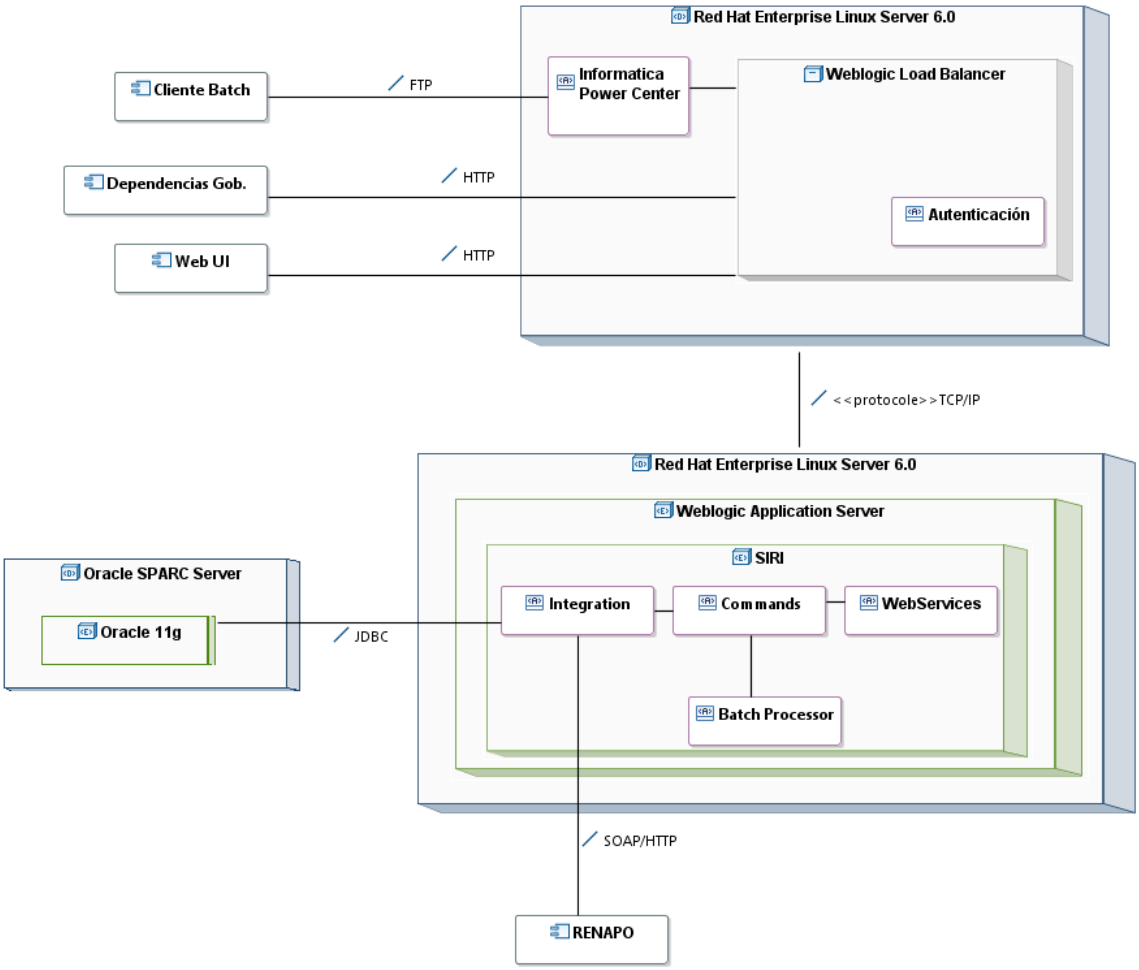


Ilustración 9 Diagrama de despliegue

ENTERPRISE ARCHITECT

Durante la fase de diseño se utilizó la herramienta Enterprise Architect. Esta herramienta nos permite crear todo tipo de diagramas UML e incluso agregar fragmentos de código y comentarios. La funcionalidad de ingeniería inversa nos permite crear o modificar diseños desde el código, es decir, una vez escrito el código, se puede extraer su diseño.

Esta herramienta no solo es útil para la fase de diseño. Sus características permiten llevar un registro desde los requerimientos hasta la fase pruebas y mantenimiento. La creación de reportes y documentación también se puede hacer desde el Enterprise Architect.

Su sistema se basa en un ambiente colaborativo. Los diseños pueden ser almacenados y compartidos a través de un repositorio.

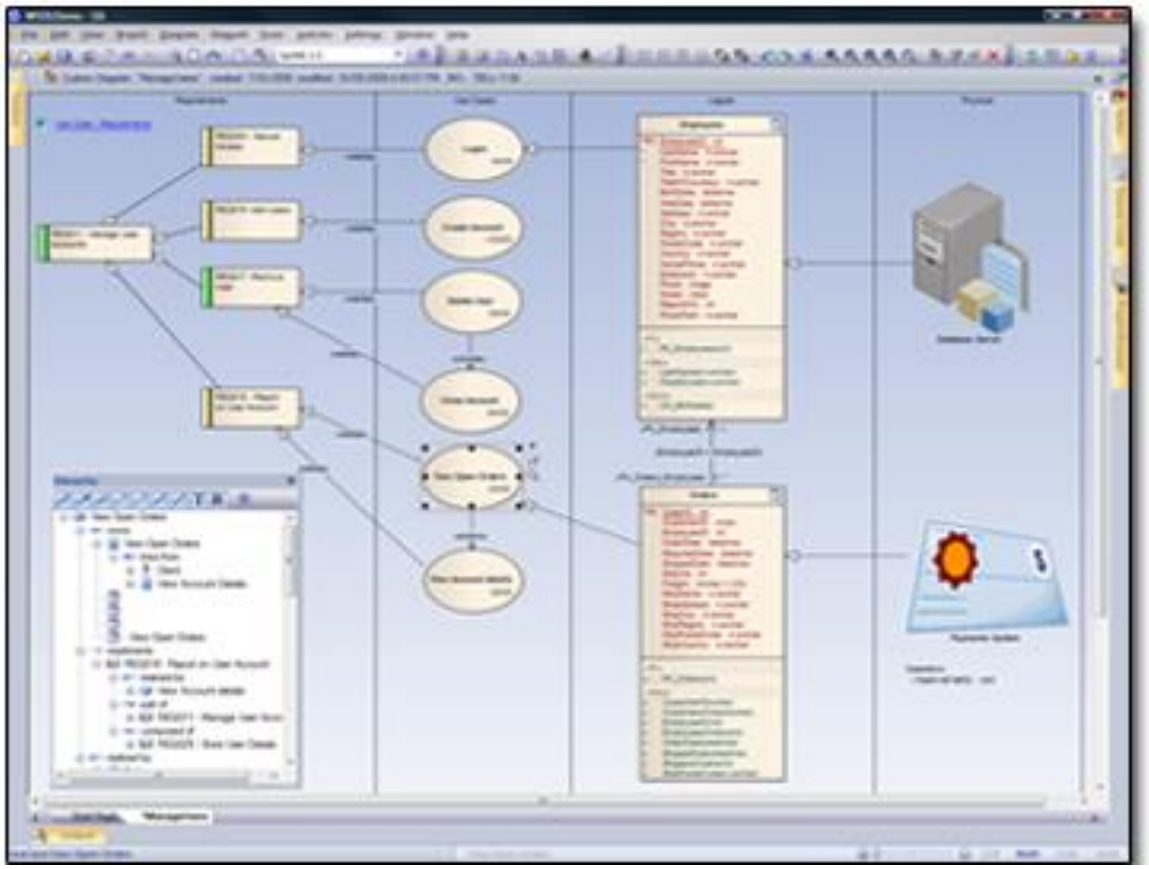


Ilustración 10 Enterprise Architect

DESARROLLO

La fase de desarrollo consiste en la creación de los componentes de software que cumplan con los requerimientos de cliente. Previamente durante la fase de diseño se define la arquitectura, estándares, metodologías y herramientas que se utilizarán.

Las complicaciones que se presentan durante esta etapa están normalmente ligadas a errores en el diseño o a modificaciones que el cliente hace una vez iniciado el proyecto. El lenguaje de programación y la falta de ingenieros calificados en las herramientas que son utilizadas, puede ser otro factor para que un proyecto de software se retrase. En mi experiencia he conocido proyectos donde la rotación de personal también juega un papel importante. La curva de aprendizaje y la adaptación de los nuevos recursos en el proyecto hacen que los planes de trabajo se modifiquen constantemente.

Actualmente existen muchas herramientas de administración de proyectos que ayudan a llevar un mejor control y a prever estos riesgos comunes. Es muy importante automatizar aquellos procesos que afecten de manera indirecta al equipo de desarrollo. Algunos de los problemas que he encontrado en diferentes empresas son:

- Integración de nuevos recursos (programadores) al proyecto.
- Buena adopción de los estándares de arquitectura
- Falta de revisión o monitoreo constante de la calidad del software
- Falta de documentación de las soluciones empleadas durante el proceso de desarrollo. Muchas de estas soluciones podrían ser reutilizadas por otros miembros del equipo.
- Ausencia de pruebas unitarias exhaustivas
- Integración de tecnologías que no estaban planeadas.
- Ausencia de pruebas de rendimiento

A continuación explicaré las tecnologías que se utilizaron para la construcción del proyecto.

JAVA

La empresa PROCESAR tiene como estándar de desarrollo el uso de Java Enterprise Edition (JEE), esta es una política que se sigue en todos los nuevos proyectos de software por lo que su valoración no es exclusiva del equipo que trabajó en SIRI. Una de las principales ventajas de Java es su robustez y nivel de madurez. Al ser una plataforma abierta permite que su comunidad de usuarios aporte mejoras y participe activamente en su evolución. Si bien es una plataforma que nace en la empresa SUN, otras empresas como IBM, Oracle y SAP participan activamente en la definición de nuevos estándares.

El respaldo de estas compañías internacionales hace que JEE sea atractivo para proyectos públicos y privados. En nuestro país es ampliamente usado en proyectos gubernamentales y en PROCESAR constituyen el 90% de los sistemas internos.

La principal desventaja actual es la falta de ingenieros especializados en nuestro país. Esto provoca que los proyectos sean más costosos debido a la poca oferta de en el mercado.

Al igual que .NET, la plataforma Java hace uso del paradigma de programación orientado a objetos que ha demostrado ser muy seguro en el desarrollo de sistemas de alta complejidad y que requieren mantenimiento constante a través de muchos años, lo cual es muy común en proyectos empresariales de la magnitud del SIRI

Características de la Programación Orientada a Objetos (POO)

Las características siguientes son las más importantes:

Abstracción: denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las

características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.

Modularidad: Se denomina Modularidad a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la Modularidad de diversas formas.

Principio de ocultación: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

Polimorfismo: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Herencia: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el

comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común.

Recolección de basura: la recolección de basura o *garbage collector* es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando.

Portabilidad

Sin embargo, Java no es el único lenguaje de programación que se crea bajo esta filosofía. La adopción de los lenguajes de programación orientados a objetos revolucionó la forma de desarrollar sistemas. Una buena adopción del paradigma por sí mismo, garantiza que los sistemas estarán preparados en un futuro para algo que es una constante en el mundo de las tecnologías de la información: el cambio.

Java ha crecido considerablemente durante estos años, y aún más las librerías asociadas. Un punto clave en su evolución ha sido la confianza que los usuarios tienen con cada nuevo lanzamiento de que sus sistemas seguirán funcionando. Esto es, compatibilidad con versiones anteriores.

La independencia de la plataforma, permitió que las empresas redujeran sus dolores de cabeza al decidir migrar sus desarrollos a otro sistema operativo. Esto también significó una reducción en el tiempo de programación porque no tenían que rescribirse instrucciones específicas de cada sistema operativo. La portabilidad ha sido una de las mayores ventajas frente a otras plataformas de desarrollo.

"Write once, run anywhere" es el *slogan* que SUN creó para ilustrar las ventajas de usar Java. Significa que se puede escribir un programa en cualquier ambiente, compilarlo, y que el *bytecode* generado se ejecutará de la misma forma en cualquier Máquina Virtual de Java (JVM).

Entornos de funcionamiento

En dispositivos móviles y sistemas empotrados

Desde la creación de la especificación J2ME (Java 2 Platform, Micro Edition), una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el mercado de dispositivos electrónicos de consumo se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos.

El modelo de desarrollo de estas aplicaciones es muy semejante a las applets de los navegadores salvo que en este caso se denominan MIDlets.

Android es otro ejemplo de implementación de la plataforma Java para dispositivos con menores recursos que una PC.

En el navegador web

Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (plug-in) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

El éxito de este tipo de aplicaciones no fue realmente el esperado debido a diversos factores, siendo quizás el más importante la lentitud y el reducido ancho de banda de las comunicaciones en aquel entonces que limitaba el tamaño de las applets que se incrustaban en el navegador. La aparición posterior de otras alternativas (aplicaciones web dinámicas de servidor) dejó un reducido ámbito de uso para esta tecnología, quedando hoy relegada fundamentalmente a componentes específicos para la intermediación desde una aplicación web dinámica de servidor con dispositivos ubicados en la máquina cliente donde se ejecuta el navegador.

Las applets Java no son las únicas tecnologías (aunque sí las primeras) de componentes complejos incrustados en el navegador. Otras tecnologías similares pueden ser: ActiveX de Microsoft, Flash, Java Web Start, etc.

En sistemas de servidor

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages).

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes *Common Gateway Interface* (CGI) y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga computacional o de memoria y propensión a errores por su interpretación dinámica).

Los Servlets y las JSPs supusieron un importante avance ya que:

- El API de programación es muy sencilla, flexible y extensible.
- Los Servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.
- Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.

A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con frameworks (Struts, Webwork, Velocity, etc.) que se sobreponen sobre los servlets y las JSPs para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que la especialización de roles sea posible y se facilite la reutilización y robustez de código. A pesar de todo ello, las tecnologías que subyacen (Servlets y JSPs) son substancialmente las mismas.

Este modelo de trabajo se ha convertido en uno de los estándar de-facto para el desarrollo de aplicaciones web dinámicas de servidor.

En aplicaciones de escritorio

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en las computadoras de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier computadora.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico. Desde la aparición de la biblioteca Swing la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

JavaFX es una familia de productos y tecnologías para la creación de *Rich Internet Applications* (RIAs), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas. Las tecnologías incluidas bajo la denominación JavaFX son JavaFX Script y JavaFX Mobile.

Las aplicaciones JavaFX pueden ser ejecutadas en una amplia variedad de dispositivos. Permite crear aplicaciones de escritorio, para celulares, la Web, TV, consolas de videojuegos, reproductores Blu-ray, entre otras plataformas planeadas.

En octubre de 2011 fue lanzada la versión 2.0. Para el desarrollo de aplicaciones JavaFX un lenguaje declarativo, tipado llamado JavaFX Script, además puede integrarse código Java en programas JavaFX. JavaFX es compilado a código Java, por lo que las aplicaciones JavaFX pueden ser ejecutadas en computadores con la máquina virtual de Java instalada (JRE), o celulares corriendo Java ME.

JAVA ENTERPRISE EDITION 5

Es una plataforma de programación para aplicaciones web y empresariales. La versión 5 representa una notable mejoría. En esta versión se simplifican muchos de los procedimientos que anteriormente tomaban mucho tiempo y constituían tareas repetitivas y complejas. Gracias a la integración de metodologías y estándares de frameworks abiertos, JEE 5 logra unificar en sus estándares las mejores prácticas en la industria.

La introducción de anotaciones simplifica muchas de las tareas rutinarias que se hacían a través de descriptores de despliegue o archivos de configuración. El concepto de *convention over configuration* ampliamente usado, proporciona al desarrollador una base confiable, pero suficientemente flexible para ser modificada, para que se preocupe en tareas que requieren más atención: solucionar los problemas de negocio.

La decisión de que version de Java utilizar es algo muy importante en un proyecto. No siempre es recomendable ir con la versión más reciente a pesar de todas las ventajas con las que se anuncie, se tiene que revisar la compatibilidad con los sistemas actuales en la empresa además de valorar el dominio que tiene el equipo de desarrollo en las nuevas tecnologías. No tomar en cuenta estos riesgos conlleva a un desajuste en las estimaciones de la duración del proyecto.

En el caso del proyecto SIRI se decidió implementar el sistema enteramente en JEE 5 y exponer WebServices para su integración con otros sistemas. Es importante recalcar que no todas las tecnologías o APIs se usan explícitamente durante el proyecto, ya que muchas de ellas están implícitas en la versión del framework correspondiente, es decir, se puede entender que al usar Spring 3.0 en adelante se está usando JEE 5

APIs y Especificaciones JEE 5

Web Services Technologies

Implementing Enterprise Web Services (JSR 109)
Java API for XML-Based Web Services (JAX-WS) 2.0 (JSR 224)
Java API for XML-Based RPC (JAX-RPC) 1.1 (JSR 101)
Java Architecture for XML Binding (JAXB) 2.0 (JSR 222)
SOAP with Attachments API for Java (SAAJ) (JSR 67)
Streaming API for XML (JSR 173)
Web Service Metadata for the Java Platform (JSR 181)

Web Application Technologies

Java Servlet 2.5 (JSR 154)
JavaServer Faces 1.2 (JSR 252)
JavaServer Pages 2.1 (JSR 245)
JavaServer Pages Standard Tag Library (JSR 52)

Enterprise Application Technologies

Enterprise JavaBeans 3.0 (JSR 220)
JEE Connector Architecture 1.5 (JSR 112)
Common Annotations for the Java Platform (JSR 250)
Java Message Service API (JSR 914)
Java Persistence API (JSR 220)
Java Transaction API (JTA) (JSR 907)
JavaBeans Activation Framework (JAF) 1.1 (JSR 925)
JavaMail (JSR 919)

Management and Security Technologies

JEE Application Deployment (JSR 88)
JEE Management (JSR 77)
Java Authorization Contract for Containers (JSR 115)

En continua revisión y expansión

Ilustración 11 APIs y Especificaciones JEE 5

ECLIPSE

Entorno de Desarrollo Integrado para JEE

Eclipse fue desarrollado originalmente por IBM Canadá como el sucesor de su familia de herramientas para VisualAge. Actualmente es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de Código abierto y un conjunto de productos complementarios, capacidades y servicios. En noviembre del 2001, se formó un consorcio para el desarrollo futuro de Eclipse como Código abierto. En 2003, la fundación independiente de IBM fue creada.

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse)

Si bien las funciones de Eclipse son más bien de carácter general, las características del programa se pueden ampliar y mejorar mediante el uso de plugins. Asimismo, a través de estos "plugins" libremente disponibles es posible añadir un sistema de control de versiones a través de Subversion y a la vez lograr una integración mediante Hibernate.

El 28 de junio de 2005 fue liberada la versión 3.1 de Eclipse, la cual incluye mejoras en el rendimiento, el soporte de Java 5.0, mejor integración con Ant (incluido debugger) y un Sistema de Control de Versiones (CVS).

Principales Ventajas en la utilización de Eclipse

- El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plugin) para proporcionar toda su funcionalidad al frente de la Plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.
- Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y sistema de gestión de base de datos.
- La arquitectura plug-in permite escribir cualquier extensión deseada en el ambiente, como asistentes de configuración, conectores a base de datos, editores gráficos, etc.

- La definición que da el proyecto Eclipse acerca de su Software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular". Esto permite que los ingenieros de software se familiaricen rápidamente con las herramientas de desarrollo y no tengan que estar cambiando entre varias ventanas para completar alguna tarea.
- Al ser una herramienta gratuita, permite reducir considerablemente los costos de un proyecto gracias al ahorro en licencias.
- El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código.
- El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadatos en un espacio para archivos, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

Estas ventajas significan mucho en proyectos gubernamentales donde los presupuestos normalmente tratan de ajustarse al mínimo.

Estas ventajas significan mucho en proyectos gubernamentales donde los presupuestos normalmente tratan de ajustarse al mínimo.

El uso de un IDE como Eclipse constituye más del 75% del tiempo de un desarrollador, el resto del tiempo se distribuye en correo electrónico, revisión de documentos y uso del navegador web.

En el día a día del proyecto se vuelve costumbre descargar actualizaciones del proyecto desde el repositorio de Subversion, luego se construye usando algun script de Maven o Ant, estos scripts pueden tener activadas las pruebas unitarias y de integración. Al hacer esto validamos que los cambios que hacemos o que otros han agregado al código base, no impacten negativamente el resto de la aplicación.

SUBVERSION

Herramienta para control de versiones

El repositorio SVN / Subversion es un sistema de control de revisiones que permite a los programadores mantener versiones actualizadas e históricas de archivos. Estos archivos pueden ser, por ejemplo, de código fuente, de páginas web o documentación.

A primera vista subversion parece una implementación mejor realizada que su antecesor cvs en ciertos detalles que dan mayor juego y son más cómodos de realizar.

- **El Repositorio**: un detalle importante es que el repositorio en el servidor no son archivos en formato rcs, es un Berkeley DB por lo que la administración del repositorio se ha de realizar mediante herramientas proporcionadas por subversion.
- **Sistemas de Acceso**: el desarrollo de subversion tiene un marcado efecto de Apache 2, ya que incluye la posibilidad mediante un módulo DAV de trabajar con el repositorio mediante el protocolo HTTP y utilizando un sistema de autentificación similar al utilizado en Apache. Resumiendo tenemos los siguientes métodos de acceso:
    - **file://** acceso al sistema local.
    - **http://** y **https://** haciendo uso del módulo de DAV en Apache 2.
    - **svn://** por svnserve y **svn+ssh://** para utilizar ssh.
- **Mejor trabajar desconectado**: los módulos sobre los que trabajan los desarrolladores están pensados para necesitar lo menos posible la conexión con el repositorio general. Podemos hacer diff de versiones anteriores, añadir archivos y directorios, comprobar logs. Todo sin tener que estar conectados con el servidor.
- **Soporte para binarios**: otra de las ventajas de subversion es la posibilidad de manejar binarios en nuestro repositorio, opción poco recomendable con cvs.
- **Facilidad para añadir/eliminar/mover archivos y directorios**: con cvs la creación de directorios y archivos y sobretodo su eliminación son tremendamente cansinos. Con subversion podemos hacer uso de comandos tales como *svn rm* <directorio>
- **Branches y Tags más sencillos**: subversion maneja el repositorio de forma absoluta sobre su estado, es decir, no mantiene versiones por cada archivo si no que toma "instantáneas" sobre todo el repositorio. Esta peculiaridad permite que la creación de braches y tags sea muy sencilla.

Desde Eclipse se puede descargar el código de los repositorios de Subversion gracias al plugin Subclipse o Subversive que se encuentran disponibles de manera gratuita en el marketplace (Menú Ayuda -> Marketplace)

Una vez descargado (checkout) se pueden hacer las modificaciones necesarias y posteriormente subir (commit) los cambios al servidor. Es recomendable hacer actualizaciones periódicas del proyecto para evitar tener conflictos con los cambios que otros desarrolladores puedan hacer a los mismos archivos en los que estamos trabajando.

MAVEN

Herramienta para automatización de tareas en la construcción de proyectos de software

Maven es una herramienta open source para administrar proyectos de software. Por administrar, se refiere a gestionar el ciclo de vida desde la creación de un proyecto en un lenguaje dado, hasta la generación de un binario que pueda distribuirse con el proyecto.

Maven nació dentro de la fundación Apache para complementar a Ant, la herramienta de compilación más usada en el mundo Java. Es similar a Make para C. Ant permite crear scripts (usando XML) que indican cómo compilar un proyecto Java y generar un binario. Maven complementa esta tarea brindándonos una estructura consistente de proyectos (todos los proyectos Maven tienen por default los mismos directorios) y herramientas necesarias actualmente para la complejidad de los proyectos de software: gestión avanzada de dependencias, informes sobre pruebas automáticas y extensibilidad vía plugins. Por detrás, Maven sigue confiando en Ant para manejar la compilación e incluso se puede usar las tareas de Ant dentro de Maven. Así que no es en sí un sustituto de esta herramienta, sino un complemento.

POM son las siglas de "Project Object Model" (Modelo de Objetos de Proyecto. Un POM no es más que la abstracción usada por Maven para definir proyectos, como tal contiene los atributos de estos y las instrucciones para construirlo. Un Proyecto en Maven se define mediante un archivo POM, que es un archivo llamado pom.xml con una etiqueta inicial <*project*>. En dicho archivo se definen cosas como las instrucciones para compilar el proyecto, las librerías de las que se depende, etc.

En Maven, la ejecución de un archivo POM siempre genera un "artefacto". Este **artefacto** puede ser cualquier cosa: un archivo jar, un swf de flash, un archivo zip o el mismo archivo pom.

Los **arquetipos** son artefactos especiales de Maven que sirven como plantillas para crear proyectos. Maven cuenta con algunos predefinidos y terceros han hecho los suyos para crear proyectos con tecnologías específicas, como proyectos web con Spring o proyectos con Adobe Flex e incluso el equipo de AppFuse ha ido más allá y tiene arquetipos para crear proyectos 100% funcionales usando una mezcla de tecnologías y generación de código.

Ejemplo de un archivo POM:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>mx.unam.ejemplo</groupId>
  <artifactId>EjemploMaven</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Example</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.13</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

groupId: Es similar al *paquete* de proyecto. Típicamente aquí se pone el nombre de la empresa u organización, ya que conceptualmente todos los proyectos con ese groupId pertenecen a una sola empresa

artifactId: Es el nombre del proyecto.

version: Número de versión del proyecto.

package: Paquete base donde irá el código fuente

dependencies: Se enlistan las dependencias con sus versiones para que funcione correctamente este proyecto

Para el proyecto SIRI se realizaron las siguientes tareas de **maven**:

- Generar un .jar con los fuentes
- Generar en formato web una documentación similar al javadoc
- Un análisis de métricas de calidad de código para cada modulo
- Un análisis de cobertura de los test, indicando qué líneas de código se han ejecutado o no en los test. Este análisis reporta a SONAR para llevar una estadística de la calidad del software a través del tiempo
- Ejecutar desde *maven* tareas de *ant*
- Preparar una distribución etiquetando todo el código fuente en svn.
- Crear un proyecto maven compuesto de varios sub proyectos
- Crear un gran jar/war/ear que tenga todos los .class, tanto los nuestros como los de los jar dependientes.
- Agregar configuraciones parametrizadas en descriptores de despliegue XML para cada ambiente: desarrollo, QA, Producción
- Copiar el componente instalable a carpetas de auto despliegue para el servidor de aplicaciones

JUNIT

Framework de pruebas unitarias

JUnit es un “framework” para automatizar las pruebas unitarias de aplicaciones Java. Se utiliza en la fase de desarrollo, su utilización por parte de los desarrolladores permite la creación de software de mayor calidad.

JUnit es un conjunto de clases que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado, si la clase cumple la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba. En caso de que el valor esperado sea diferente al retornado por el método durante la ejecución, JUnit devolverá un error en el método correspondiente.

JUnit es también un medio para controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple los requisitos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. Esto es de gran utilidad ya que en el proyecto SIRI participaron más de 30 personas de manera concurrente y cada cambio lleva el riesgo de impactar desarrollo previamente hecho por otros programadores.

JUnit es también un medio para controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple los requisitos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. Esto es de gran utilidad ya que en el proyecto SIRI participaron más de 30 personas de manera concurrente y cada cambio lleva el riesgo de impactar desarrollo previamente hecho por otros programadores.

Ejemplo:

```
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestJunit {


   String message = "Titulacion por Experiencia Profesional";

   MessageUtil messageUtil = new MessageUtil(message);
```

```
   @Test
   public void testPrintMessage() {
      assertEquals(message,messageUtil.printMessage());
   }
}
```

Métodos assertxxx() de JUnit.

Algunos métodos para hacer comprobaciones son:

Método assertxxx() de JUnit	**Qué comprueba**
assertTrue(expresión)	comprueba que expresión evalúe a true
assertFalse(expresión)	comprueba que expresión evalúe a false
assertEquals(esperado,real)	comprueba que esperado sea igual a real
assertNull(objeto)	comprueba que objeto sea null
assertNotNull(objeto)	comprueba que objeto no sea null
assertSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado y objeto_real sean el mismo objeto
assertNotSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado no sea el mismo objeto que objeto_real
fail()	hace que el test termine con fallo

EJB 3

Estándar para la creación de componentes de negocio distribuidos

Un EJB (Enterprise Java Bean) es un componente que debe ejecutarse de un contenedor de EJBs. Entre otras tareas, el contenedor gestionará el ciclo de vida de cada componente.

El objetivo de estos objetos es principalmente mantener la lógica de negocio separada del resto de la aplicación. Los clientes que necesiten usar estos servicios tendrán que hacerlo a través de la interfaz pública que ofrezca cada EJB

Poco a poco los estándares van madurando y aunque durante muchos años se ha usado Spring, Struts, Hibernate como frameworks de referencia, poco a poco los estándares van evolucionando y las organizaciones involucradas en las especificaciones de la plataforma java van integrando las mejores ideas en los APIs oficiales.

El contenedor de proporciona una capa de servicios añadidos, los más importantes son los siguientes:

- Manejo de transacciones: apertura y cierre de transacciones asociadas a las llamadas a los métodos del bean.
- Seguridad: comprobación de permisos de acceso a los métodos del bean.
- Concurrencia: llamada simultánea a un mismo bean desde múltiples clientes.
- Servicios de red: comunicación entre el cliente y el bean en JVMs distintas.
- Gestión de recursos: gestión automática de múltiples recursos, como colas de mensajes, bases de datos o fuentes de datos en aplicaciones heredadas que no han sido traducidas a nuevos lenguajes/entornos y siguen usándose en la empresa.
- Persistencia: sincronización entre los datos del bean y tablas de una base de datos.
- Gestión de mensajes: manejo de Java Message Service (JMS).
- Escalabilidad: posibilidad de constituir clusters de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.
- Adaptación en tiempo de despliegue: posibilidad de modificación de todas estas características en el momento del despliegue del bean.

Desde la versión 3.0, EJB no impone ninguna restricción ni obligación a nuestros objetos de negocio de implementar una API en concreto. Dicho de otro modo, podemos escribir dichos objetos de negocio usando Plain Old Java Objects (POJOs), facilitando entre otras cosas la reutilización de componentes y la tarea de probarlos

Por otro lado, el uso de POJOs para encapsular nuestra lógica de negocio nos proporciona un modelo simple que es altamente. Se debe tener en cuenta que un POJO no actuará como un componente EJB hasta que haya sido empaquetado, desplegado en un contenedor EJB y accedido por dicho contenedor (por ejemplo a petición de un usuario). Una vez que un POJO definido como EJB haya sido desplegado en el contenedor, se convertirá en uno de los tres siguientes componentes (dependiendo del como lo hayamos definido):

- Session Bean
- Message-Driven Bean
- Entity Bean

Session Beans

Los componentes Session Beans son los componentes que contienen la lógica de negocio que requieren los clientes de nuestra aplicación. Son accedidos a través de un *proxy* tras realizar una solicitud al contenedor. Tras dicha solicitud, el cliente obtiene una referencia del Session Bean, pero no el Session Bean real. Esto permite al contenedor realizar ciertas operaciones sobre el Session Bean real de forma transparente para el cliente (como gestionar su ciclo de vida, solicitar una instancia a otro contenedor trabajando en paralelo, etc.)

Los componentes Session Bean pueden ser de tres tipos:

- Stateless Session Beans
- Stateful Session Beans
- Singletons

Los componentes Stateless Session Beans (SLSB) son componentes que no requieren mantener un estado entre diferentes invocaciones. Un cliente debe asumir que diferentes solicitudes al contenedor de un mismo SLSB pueden devolver vistas a objetos diferentes. Dicho de otra manera, un SLSB puede ser compartido (y probablemente lo será) entre varios clientes. Por todo esto, los SLSB son creados y destruidos a discreción del contenedor, y puesto que no mantienen estado son muy eficientes a nivel de uso de memoria y recursos en el servidor. Además de que los hace altamente escalables en ambientes de clúster.

Los componentes Stateful Session Beans (SFSB), al contrario que SLSB, sí mantienen estado entre distintas invocaciones realizadas por el mismo cliente. Esto permite crear un estado conversacional (como el carrito de la compra en una tienda online, que mantiene los objetos que hemos añadido mientras navegamos por las diferentes páginas), de manera que acciones llevadas a cabo en invocaciones anteriores son tenidas en cuenta en acciones posteriores. Un SFSB es creado justo antes de la primera invocación de un cliente, mantenido ligado a ese cliente, y destruido cuando el cliente invoque un método en el SFSB que esté marcado como finalizador (también puede ser destruido por timeout de sesión). Son menos eficientes a nivel de uso de memoria y recursos en el servidor que los SLSB.

Los componentes **Singleton** son un nuevo tipo de Session Bean introducido en EJB 3.1. Un Singleton es un componente que puede ser compartido por muchos clientes, de manera que una y solo una instancia es creada. A nivel de eficiencia en uso de memoria y recursos son indiscutiblemente los mejores, aunque su uso está restringido a resolver ciertos problemas muy específicos.

Message-Driven Beans

Los componentes Message-Driven Beans (MDB) son componentes de tipo *listener* que actúan de forma asíncrona. Su misión es la de consumir mensajes (por ejemplo eventos que se producen en la aplicación), los cuales pueden gestionar directamente o enviar (derivar) a otro componente. Los MDB actúan sobre un proveedor de mensajería, por ejemplo Java Messaging System (JMS es además soportado de forma implícita por la especificación EJB).

Al igual que los Stateless Session Beans, los Message-Driven Beans no mantienen estado entre invocaciones.

Entity Beans

Los componentes Entity Beans (Beans de Entidad, a partir de ahora EB) son representaciones de información (en forma de POJO's) que es almacenada en una base de datos. El encargado de gestionar los EB es *EntityManager*, un servicio que es suministrado por el contenedor y que está incluido en la especificación Java Persistence API (JPA - API de Persistencia en Java).

JPA es parte de EJB desde la versión 3.0. Al contrario que los Session Beans y los Message-Driven Beans, los Entity Beans no son componentes del lado del

servidor. En otras palabras, no trabajamos con una vista del componente, si no con el componente real.

Ejemplo de un SLSB

```
@Stateless

public class CalculatorBean implements Calculator {

public double calculate (int start, int end,  double growthrate, double saving) {

   double tmp = Math.pow(1. + growthrate / 1.2,1.2 * (end - start) + 1);

   return saving * 12. * (tmp - 1) / growthrate;

  }

}
```

Invocación del EJB desde un MDB

```
public class CalculatorMDB implements MessageListener {

@EJB Calculator cal;

 // Use the cal variable

  // cal.calculate(1,10, 3.9, 4.5);

}
```

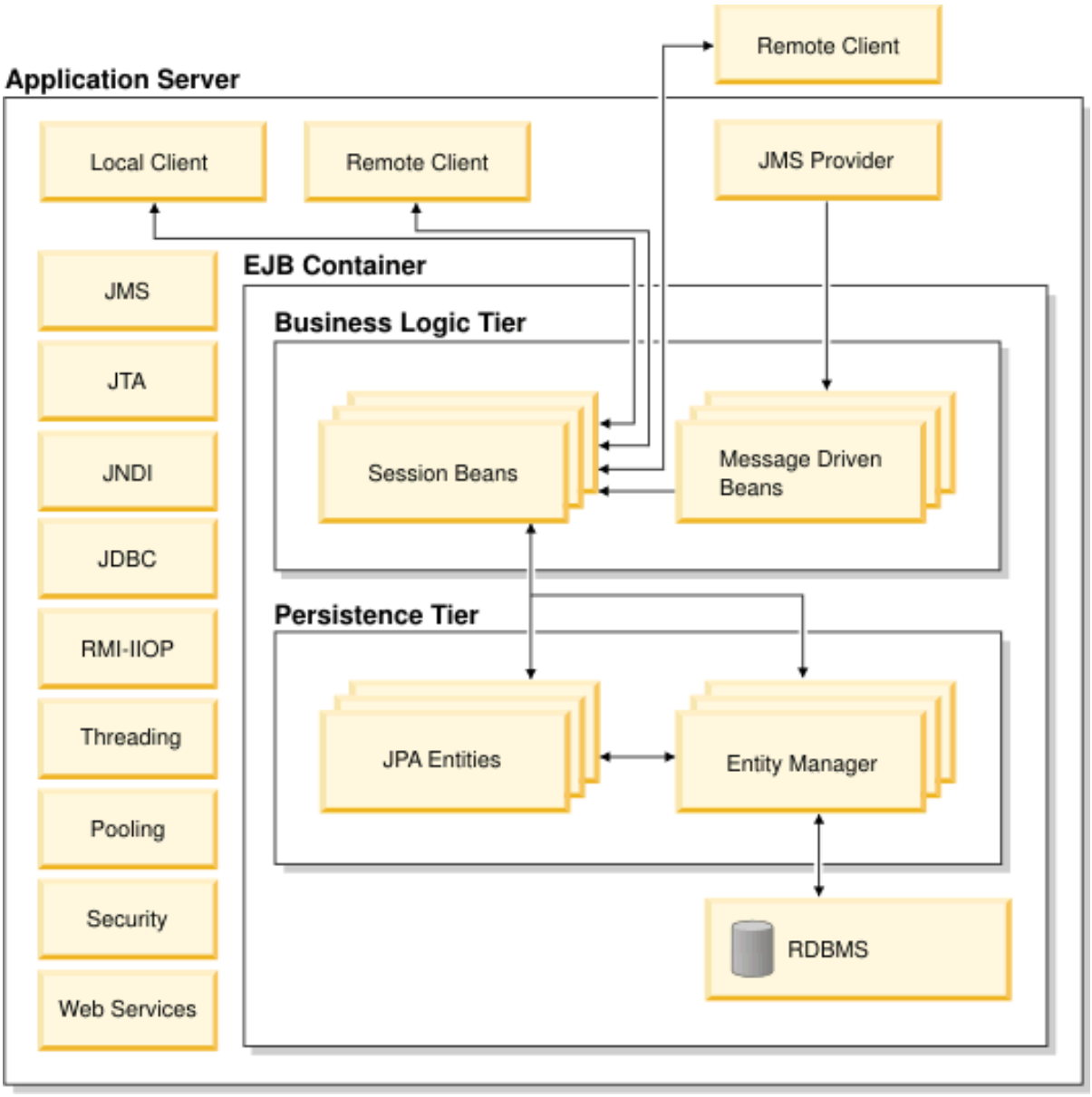


Ilustración 12 Arquitectura de un Servidor de Aplicaciones

La imagen muestra el ambiente de ejecución de un servidor de aplicaciones. Dicho servidor provee un contenedor de EJBs que permite la ejecución y administración de los servicios adicionales que forman parte del estándar.

En el proyecto SIRI se utilizaron ampliamente los SLSB como implementación del patrón de diseño Business Facade. Estos EJBs exponen un WebService que otros sistemas utilizan para invocar las diferentes operaciones del SIRI. Las ventajas de utilizar esta implementación radica en que se simplifica el acceso a la aplicación en una interfaz más simple y al ser los SLSB altamente escalables se tiene la

garantía de que el sistema puede incrementar su capacidad sin modificar el código. Esta interfaz común además permite utilizar funciones adicionales de auditoría, seguridad y balanceo de carga.

La implementación interna delega la ejecución de las distintas tareas a otros componentes de negocio. Estos componentes eran principalmente objetos de Spring que eran los encargados de ejecutar la lógica de negocio e invocar a la capa de persistencia cuando se requería acceder a la base de datos.

WEB SERVICES (JAX-WS)

Estándar para la creación de servicios web basados en XML y SOAP

API Java para XML Web Services (JAX-WS), JSR 224, es una parte importante de Java EE 5. Un seguimiento a la liberación del API Java para RPC basado en XML 1.1 (JAX-RPC), JAX-WS simplifica la tarea de desarrollar servicios web utilizando la tecnología Java. Proporciona soporte para múltiples protocolos como SOAP 1.1, SOAP 1.2, XML sobre HTTP. JAX-WS utiliza JAXB 2.0 para el enlace de datos y es compatible con personalizaciones para el control de las interfaces generadas en los *endpoints*. Con su soporte para anotaciones, JAX-WS simplifica el desarrollo de servicios web y reduce el tamaño de los archivos JAR en tiempo de ejecución.

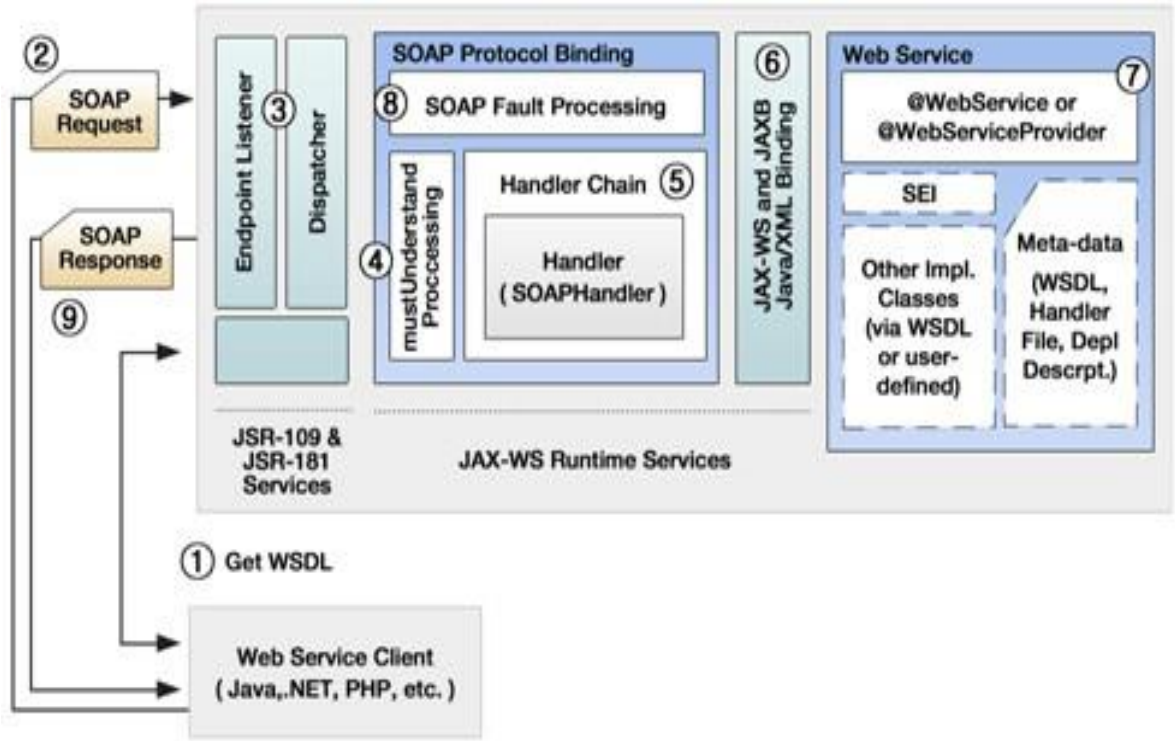


Ilustración 13 Invocación y ejecución de un Web Service

Los servicios Web Java (construidos con JAX-WS o con JAX-RPC) suelen desplegarse dentro de un servidor de aplicaciones Java EE. Normalmente se desplegarán como un Servlet que será quien gestione y redirecciones las peticiones HTTP que contengan los mensajes SOAP dirigidos al servicio Web

En ese caso la clase compilada que implementa el servicio Web, junto con las clases auxiliares generadas con *wsgen* y el correspondiente archivo *WSDL* se empaquetarán en un archivo .WAR (un .jar que contiene componentes Web).

Ese archivo WAR se desplegará conforme al esquema que define el servidor de aplicaciones Java EE (archivos web.xml, etc.)

También es posible que un Enterprise Java Bean (objeto remoto de Java EE) exponga sus métodos en forma de Servicios Web (en ese caso se empaquetaría en un archivo .EAR)

La mayoría de IDEs, como Eclipse o Netbeans, incluyen asistentes y/o automatizan las tareas de importación/exportación de WSDL, generación de ''artefactos'', compilación, empaquetado y despliegue de aplicaciones.

Ejemplo de un EJB sin estado expuesto como Web Service

```
import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless
@WebService(
        portName = "CalculatorPort",
        serviceName = "CalculatorService",
        targetNamespace = "http://unam.mx/wsdl",
        endpointInterface = "mx.unam.calculator.ws.CalculatorWs")
public class Calculator implements CalculatorWs {

    public int sum(int add1, int add2) {
        return add1 + add2;
    }

    public int multiply(int mul1, int mul2) {
        return mul1 * mul2;
    }
}
```

Durante el proyecto fue necesario exponer diferentes WebServices además de algunos clientes de WebService, para validar datos de un CURP con RENAPO por ejemplo. Hasta antes de la aparición de JAX-WS era una tarea muy tediosa trabajar con WebServices en java. Ahora es tan simple como agregar anotaciones en una clase como se ve en ejemplo anterior. Java también cuenta con herramientas para generar automáticamente las clases necesarias para consumir un WebService a partir de un contrato WSDL

SPRING FRAMEWORK

Framework para el desarrollo de aplicaciones empresariales con Java

Spring Framework es una plataforma que nos proporciona una infraestructura que actúa de soporte para desarrollar aplicaciones Java. Spring maneja toda la infraestructura y así es posible enfocarse en las particularidades de cada aplicación. Es decir, Spring es el "conector" que une todos los componentes de la aplicación, maneja su ciclo de vida y la interacción entre ellos.

Spring Framework es un contenedor ligero ("lightweight container") en contraposición a un servidor de aplicaciones JEE. En el caso de una aplicación web, basta con un contenedor de servlets como Tomcat o Jetty. Pero Spring no solo se puede usar para crear aplicaciones web, se podría usar para cualquier aplicación java, aunque su uso habitual sea en entornos web, nada impide utilizarlo para cualquier tipo de aplicación.

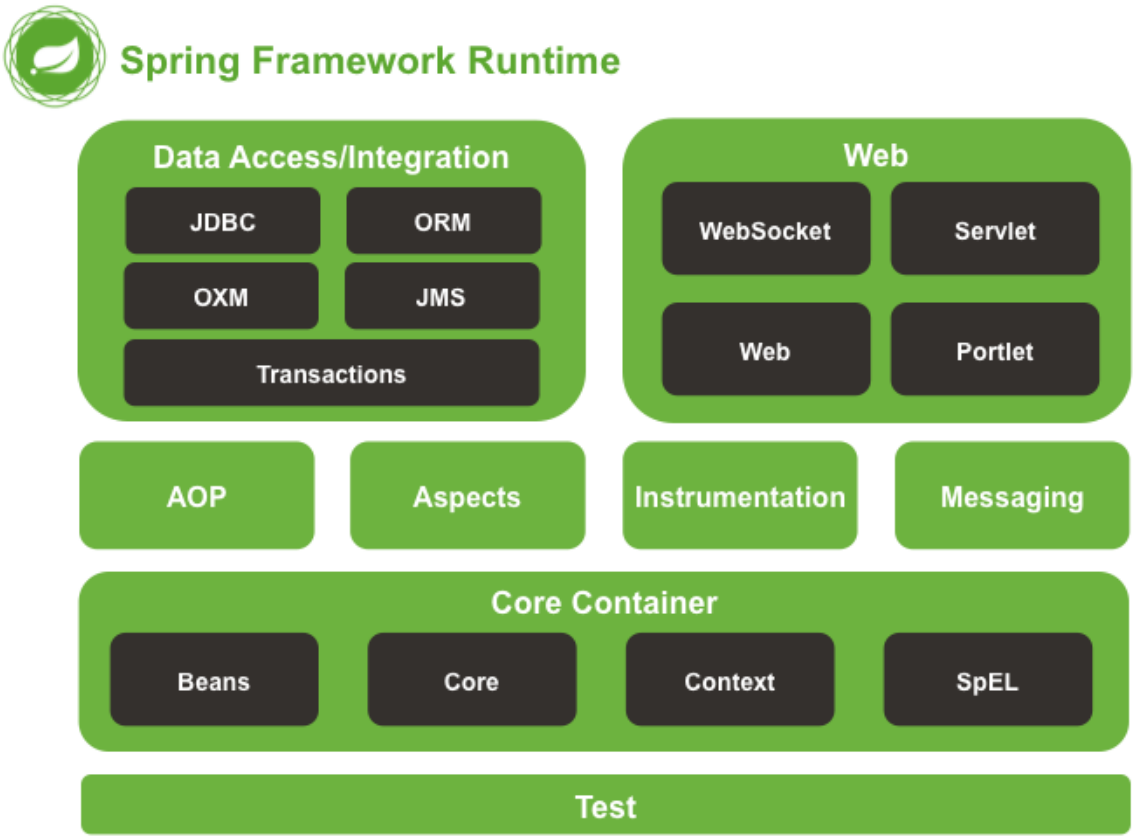


Ilustración 14 Componentes del Framework de Spring

En los inicios de las aplicaciones JEE, EJB era muy complejo y tedioso, tanto la versión 1 como la 2. La primera versión de Spring se lanzó en junio de 2003, aunque el gran lanzamiento se hizo en Marzo de 2004, con la versión 1.0. Meses más tarde Rod Johnson, creador de Spring, publicó el libro: “J2EE development without EJB“.

Las ideas "innovadoras" que en su día popularizó Spring se han incorporado en la actualidad a las tecnologías y herramientas estándar. Así, ahora mismo no hay una gran diferencia entre el desarrollo con Spring y el desarrollo JavaEE "estándar", o al menos no tanta como hubo en su día. No obstante, Spring ha logrado aglutinar una importante comunidad de desarrolladores en torno a sus tecnologías y hoy por hoy sigue constituyendo una importante alternativa al estándar que merece la pena conocer. En la actualidad, las aportaciones más novedosas de Spring se centran en los campos de Big Data/NoSQL, HTML5/móviles y aplicaciones sociales.

Básicamente, la mayor diferencia práctica que podemos encontrar hoy en día entre desarrollar con Spring y con JEE estándar es la posibilidad de usar un servidor web convencional al estilo Tomcat para desplegar la aplicación. Las tecnologías JavaEE más sofisticadas requieren del uso de un servidor de aplicaciones, ya que los APIs los implementa el propio servidor, mientras que Spring no es más que un conjunto de librerías portables entre servidores. En otras palabras, usando JavaEE estándar, nos atamos al servidor de aplicaciones y usando Spring nos atamos a sus APIs. Eso sí, los desarrolladores de Spring se han preocupado bastante de armonizar con el estándar en la medida de lo posible, por ejemplo dando la posibilidad de usar anotaciones estándar aun con implementaciones propias por debajo. La idea es obstaculizar lo menos posible una posible portabilidad a JavaEE, idea que es de agradecer en un mundo en que todos los fabricantes intentan de una forma u otra mantener un público cautivo.

Desde un punto de vista genérico, Spring se puede ver como un soporte que proporciona tres elementos básicos:

- **Servicios Enterprise**: podemos hacer de manera sencilla que un objeto sea transaccional, o que su acceso esté restringido a ciertos roles, o que sea accesible de manera remota y transparente para el desarrollador, o acceder a otros muchos servicios más, sin tener que escribir el código de manera manual. En la mayoría de los casos solo es necesario anotar el objeto.

- **Estereotipos** configurables para los objetos de nuestra aplicación: podemos anotar nuestras clases indicando por ejemplo que pertenecen a la capa de negocio o de acceso a datos. Se dice que son configurables porque podemos definir nuestros propios estereotipos "a medida": por ejemplo podríamos definir un nuevo estereotipo que indicara un objeto de negocio que además sería cacheable automáticamente y con acceso restringido a usuarios con determinado rol.
- **Inyección de dependencias**: La inyección de dependencias nos permite solucionar de forma sencilla y elegante cómo proporcionar a un objeto cliente acceso a un objeto que da un servicio que este necesita. Por ejemplo, que un objeto de la capa de presentación se pueda comunicar con uno de negocio. En Spring las dependencias se pueden definir con anotaciones o con XML.

Cuando se diseña una aplicación en Java se dispone de muchos objetos que se relacionan entre sí mediante composición. Para enlazar dos objetos se tiene que inyectar a uno de ellos una instancia del otro. Esto lo realiza Spring automáticamente, por eso se llama Inversión de control, porque es Spring quien se encarga de estas dependencias, instancia los objetos y los inyecta por *reflexión*. A grandes rasgos, se declara en un XML los componentes de la aplicación y sus dependencias. Spring lee este XML, llamado *Application Context*, crea los componentes y sus relaciones entre ellos. Las últimas versiones de Spring, ya permiten anotaciones, y se puede anotar una propiedad en una clase mediante @Autowired para que Spring busque la clase correspondiente, la instancie y la inyecte, ahorrándonos bastante código XML.

La “Dependency Injection” ya no es un concepto propio de Spring, otros frameworks lo imitaron. Desde la version 6 de JEE existe la anotación *@Inject* para hacer exactamente lo mismo

La parte fundamental de Spring, es su módulo Core, el cual se encarga de la Inyección de dependencias. El concepto de este módulo es el *BeanFactory*, que implementa el patrón de diseño *Factory*. Arriba del Core, encontramos el módulo Context (no está en el diagrama), que se encarga de brindar las herramientas para acceder a los beans.

Data: Provee una capa de abstracción sobre JDBC, abstrae el código de acceso a datos de una manera simple. Cuenta con una capa de excepciones sobre los mensajes de error provistos por cada servidor específico de base de datos.

AOP: En simples palabras esta capa desacopla el código de una manera limpia, implementando funcionalidad que por lógica y claridad debería estar separada. Usando metadatos a nivel de código fuente se pueden incorporar diversos tipos de información y comportamiento al código

ORM: Provee capas de integración para APIs de mapeo objeto - relacional, incluyendo, JDO, Hibernate e iBatis. Usando el paquete ORM es posible usar esos mapeadores en conjunto con otras características que Spring ofrece, como la administración de transacciones.

WEB: provee características básicas de integración orientadas a la web, como funcionalidad multipartes (para realizar la carga de archivos), inicialización de contextos mediante *Servlet Listeners* y un contexto de aplicación orientado a web.

BATCH: permite definir procesos que realicen lecturas a archivos, ejecución de consultas a base de datos, procesadores, y escrituras de resultados a archivos y base de datos; todo ello, de una forma sencilla y, además, ocultando operaciones de infraestructura. Es decir, con Spring podemos definir y desarrollar procesos batch de una forma sencilla y declarativa.

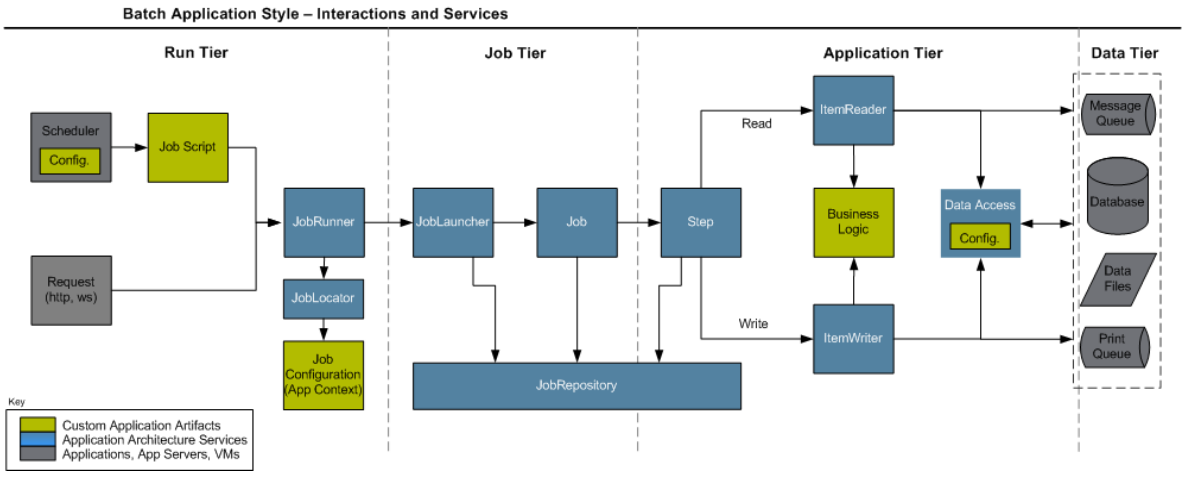


Ilustración 15 Flujo de una aplicación con Spring Batch

El uso del framework de Spring se ha vuelto un estándar en la comunidad de programadores Java. La base del éxito de Spring es su comunidad de expertos que constantemente está implementando mejores prácticas de desarrollo y generando arquitecturas reutilizables en los diferentes módulos de su framework.

Una parte fundamental del sistema es el procesamiento por lotes. El sistema recibe un pico de carga al final de cada cierre del bimestre en el SAR. Las dependencias gubernamentales tienen la posibilidad de enviar el reporte de aportaciones patronales en un archivo de texto plano que puede contener cientos de miles de registros. Cada uno de estos registros puede significar diferentes tipos de movimientos como: aportaciones a subcuentas de vivienda, modificación de datos personales, alta de nuevos empleados, etc.

Con el uso de Spring Batch se logra resolver este problema con una arquitectura robusta y flexible que divide el total de registros en fragmentos más pequeños y posteriormente ejecuta en paralelo las tareas (jobs) necesarias para la lectura, procesamiento y escritura de resultados correspondientes.

PATRONES DE DISEÑO

Se define un patrón como una solución recurrente para un problema en un contexto

Un contexto es el entorno, situación, o condiciones interrelacionadas dentro de las cuales existe algo. Un problema es una cuestión insatisfecha, algo que se necesita investigar y resolver. Un problema se puede especificar mediante un conjunto de causas y efectos. Normalmente un problema está restringido al contexto en el que ocurre. Finalmente, la solución se refiere a la respuesta al problema dentro de un contexto que ayuda a resolver las dificultades.

Los patrones deberían comunicar soluciones de diseño a los desarrolladores y arquitectos que los leen y los utilizan.

Un patrón describe, con algún nivel de abstracción, una solución experta a un problema. Normalmente, un patrón está documentado en forma de una plantilla. Aunque es una práctica estándar documentar los patrones en un formato de plantilla especializado, esto no significa que sea la única forma de hacerlo. Además, hay tantos formatos de plantillas como autores de patrones, esto permite la creatividad en la documentación de patrones.

Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que describen soluciones para todo, desde el análisis hasta el diseño y desde la arquitectura hasta la implementación

Java tiene su propio catálogo de patrones que han sido probados por la industria e implementados en el mismo desarrollo de las librerías de java. Estos patrones se reúnen en la siguiente ilustración.

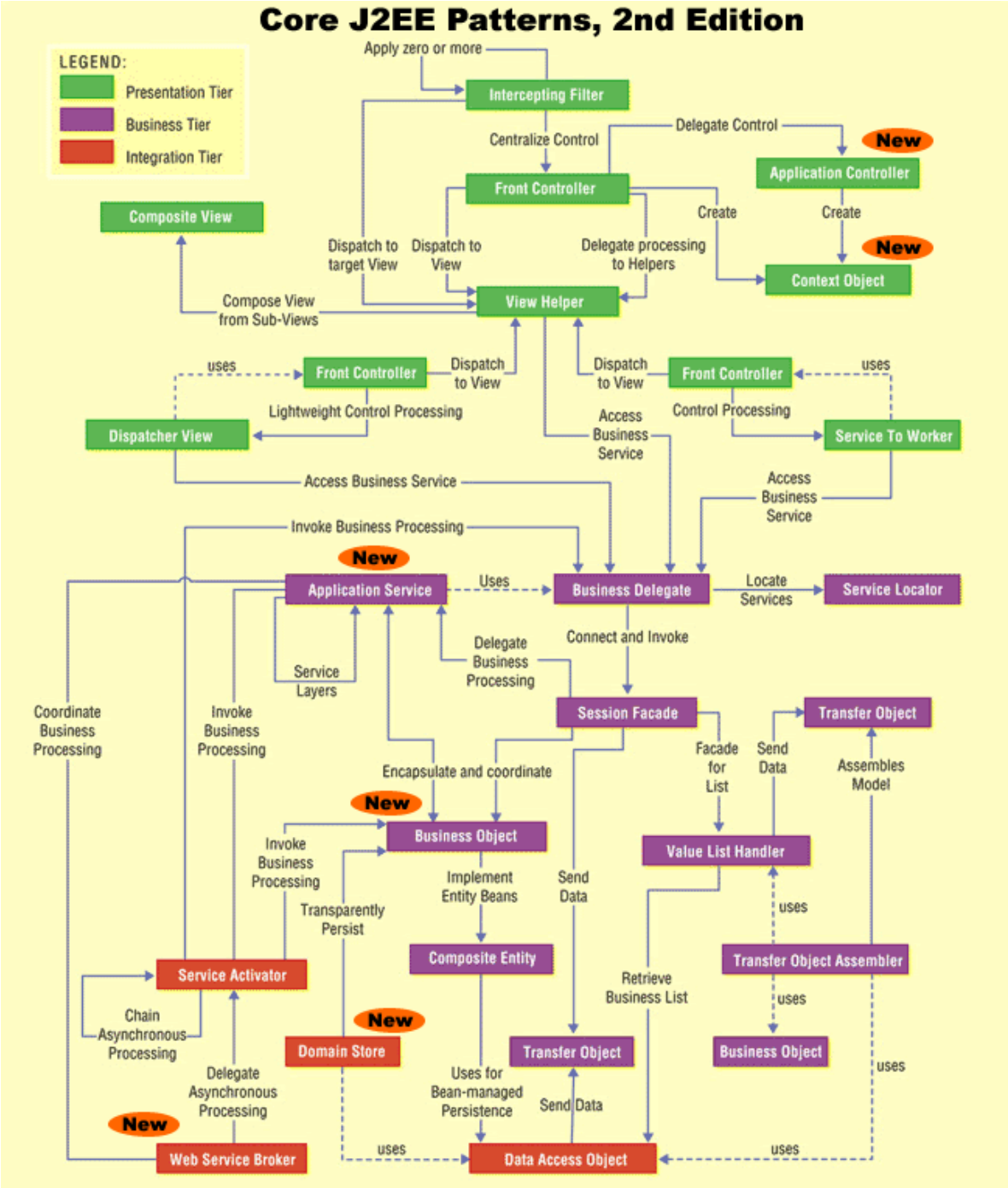


Ilustración 16 Core J2EE Patterns

PATRONES DE DISEÑO GOF (The Gang of Four)

El término de patrón fue dado por primera vez en el año 1977 por el arquitecto Christopher Alexander, quien dio en su libro *A Pattern Language* la siguiente definición: "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma". En la ingeniería del software, un patrón constituye el apoyo para la solución a los problemas más comunes que se presentan durante las diferentes etapas del ciclo de vida del software. Ahora bien, los patrones de diseño según The Gang of Four -GOF- "describen soluciones simples y elegantes a problemas específicos en el diseño de software orientado a objetos" (Gamma et al, 1994), Eric Braude define los patrones de diseño como "combinaciones de componentes, casi siempre clases y objetos que por experiencia se sabe que resuelven ciertos problemas de diseño comunes". En términos generales es posible decir que un patrón de diseño es una solución a un problema recurrente en el diseño de software.

La línea base en el tema de patrones de diseño la impone el catálogo *"Design Patterns: Elements of Reusable Object-Oriented Software"*, en este libro se presenta un conjunto de 23 patrones de diseño identificados a partir del estudio y la experiencia del grupo GOF, quienes se dedicaron a analizar los problemas recurrentes en el desarrollo de software y realizaron una clasificación y agrupación a partir de dos criterios, su propósito y alcance, las categorías definidas son: creacionales, estructurales y de comportamiento

i) **Patrones creacionales**: Se ocupan del proceso de creación de clases y objetos, son los encargados de "abstraer el proceso de instanciación o creación de objetos, ayudan a que el sistema sea independiente de cómo sus objetos son creados, integrados y representados". Los patrones que hacen parte de esta categoría son cinco: Factory Method, Abstract Factory, Builder, Prototype y Singleton

ii) **Patrones estructurales**: Tratan de la composición de clases y objetos, "se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes". Los patrones en esta categoría se encargan de lograr que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos (Braude, 2003). Los patrones que hacen parte de esta categoría son siete: Adapter, Bridge, Composite, Decorator, Façade,Flyweight y Proxy

iii) **Patrones de comportamiento**: Caracterizan las formas en que las clases o los objetos interactúan y distribuyen la responsabilidad. "Son los

encargados de las opciones de comportamiento de la aplicación, permitiendo que el comportamiento varíe en tiempo de ejecución, sin estos patrones cada comportamiento tendría que diseñarse e implementarse por separado". Los patrones que se agrupan en esta categoría son once: Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor.

El empleo de patrones de diseño obedece a las necesidades y características de cada proyecto. No debe usarse como una obligación ya que cada problema es distinto. A veces los programadores inexpertos consideran que si no usan todos los patrones que dicta un libro estarán haciendo algo mal. En muchos sistemas el uso excesivo de patrones tiene como consecuencia una baja en el rendimiento de la aplicación, debido principalmente a la agregación de capas e intermediarios. Además usar patrones poco comunes hace que el código se vuelva menos entendible y que su mantenimiento futuro sea más complicado

Uno de los patrones principales durante el proyecto fue el patrón Command. El objetivo era construir un motor de reglas flexible que permitiera a los usuarios generar nuevas reglas de negocio o hacer modificaciones a las existentes sin necesidad de pasar por todo el ciclo de desarrollo. La arquitectura que resultó se basa en 3 componentes principales: pre-requisitos, acciones y post-acciones.

Cada uno de estos componentes implementa el patrón de diseño Command cuyo único método execute se encarga de ejecutar un pequeño fragmento de código. Así por ejemplo puede existir una regla de negocio compleja que se pueda dividir en 3 pre-requisitos, 2 acciones y 1 post-acción que al encadenarse generen el resultado deseado. Esta cadena de comandos resulta ser bastante flexible a la hora de generar nuevas reglas. Una vez compilados los distintos comandos se pueden publicar en el servidor y desde la base de datos construir nuevas reglas de negocio que reutilicen los comandos existentes para la construcción de nuevas reglas de negocio.

ORACLE 11g

Base de Datos

Una de las ventajas de utilizar Oracle Database 11G es que puede ayudar a crear aplicaciones de bases de datos personalizadas. En lugar de utilizar un programa de bases de datos estandarizado, se puede obtener un programa personalizado para que se ajuste a las necesidades del proyecto. Esto hace que sea posible crear una base de datos que sólo tenga las funciones indispensables. También dará la oportunidad de eliminar cualquier función no necesaria en la aplicación de bases de datos del negocio

Minería de datos

Oracle Database 11G también ayuda con la minería de datos. Esta aplicación de bases de datos hace que sea posible encontrar las relaciones en los datos que de otra forma serían difíciles de encontrar de forma manual. Si se está trabajando con grandes cantidades de datos regularmente, esto puede ahorrar tiempo. También tiene la habilidad de descubrir patrones en los datos a través del tiempo.

Costo

Una de las desventajas de utilizar Oracle Database 11G es el costo. Cuando se invierte en este tipo de software de administración de bases de datos, se debe pagar una licencia por su uso. Las licencias de Oracle suelen tener una tarifa bastante elevada además de que los administradores de bases de datos que dominen esta tecnología no son fáciles de encontrar y por lo tanto su sueldo es mayor.

Curva de aprendizaje

Otra desventaja de usar Oracle Database 11G es que hay una curva de aprendizaje relacionada. Esto no es algo que se pueda comprender en un periodo corto, sobre todo si no se tiene experiencia previa con algún otro manejador como MySQL o Postgres.

En la siguiente ilustración se muestra la complejidad de una base de datos de Oracle 11g

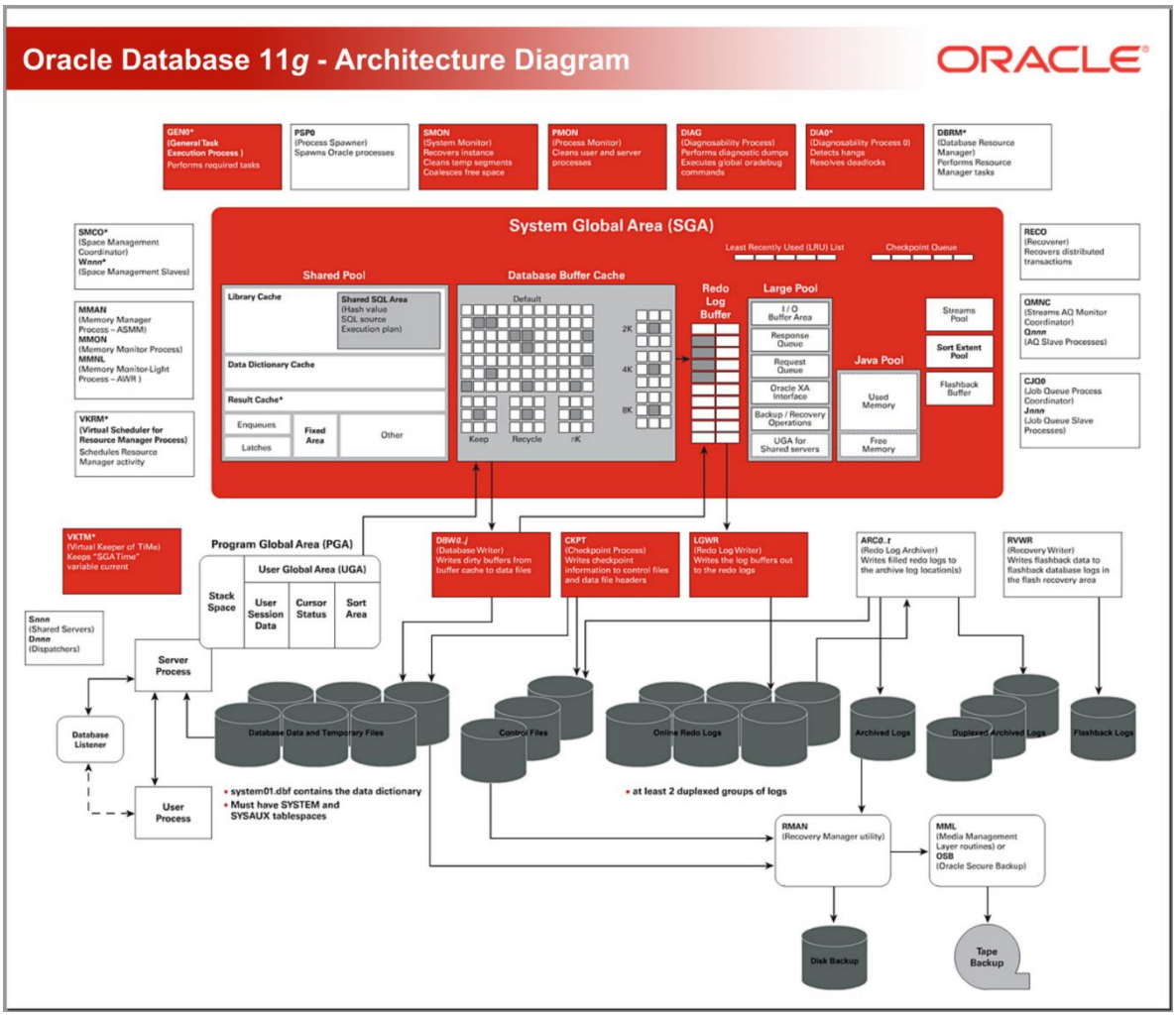


Ilustración 17 Arquitectura de Oracle 11g

SOAP UI

Herramienta para probar Web Services

SoapUI es una aplicación muy versátil que nos permite probar, simular y generar código de servicios web de forma ágil, partiendo del contrato de los mismos en formato WSDL y con vínculo SOAP sobre HTTP. SoapUI tiene dos distribuciones: soapUI freeware y soapUIPro (comercial), en versión de escritorio, online y plugin para varios IDEs.

La mayoría de las ocasiones usamos SoapUI para validar el comportamiento de la invocación de un WebService tipo Soap. En la siguiente ilustración se muestra el contenido XML de una solicitud al servicio:

http://www.webservicex.net/globalweather.asmx

Del cual previamente se obtuvo su WSDL para la generación del proyecto

Ilustración 18 Ejemplo de una solicitud a un WebService

La solicitud de la operación *GetCitiesByCountry* recibe el parámetro *CountryName*. Una vez llenados los parámetros del método se ejecuta la llamada con el botón ejecutar en la esquina superior izquierda de la ventana.

Tras unos instantes, recibimos el mensaje de respuesta SOAP que se mostrará a la derecha de la ventana anterior:

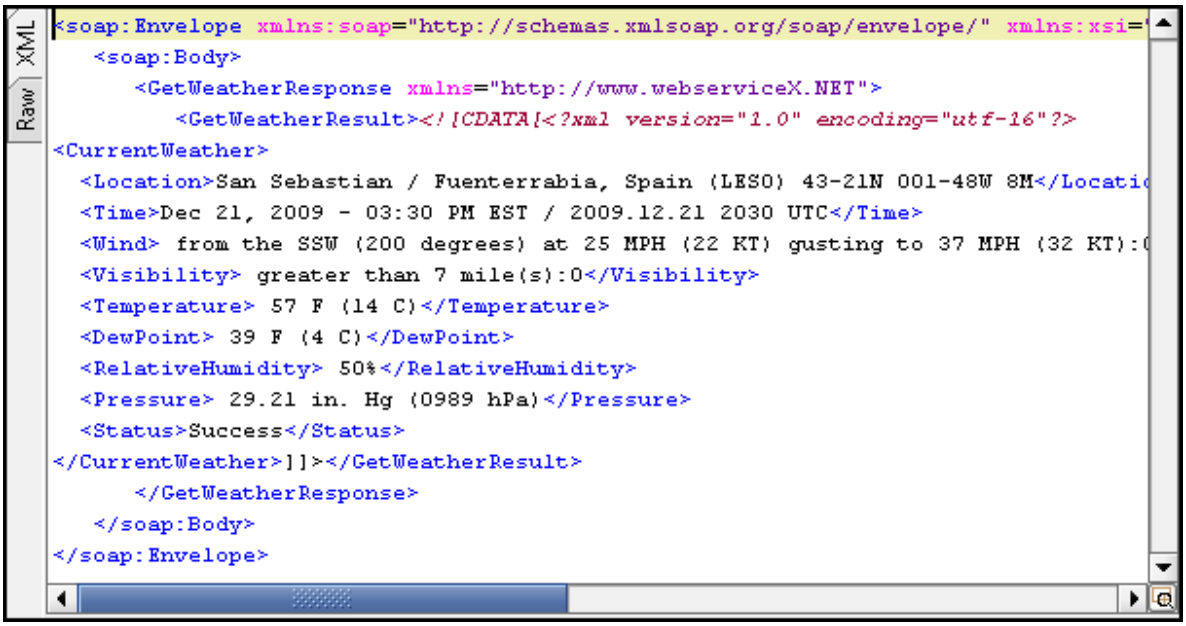


Ilustración 19 Ejemplo de respuesta de un WebService

En caso de que la invocación sea incorrecta, ya sea por falta de parámetros, error en el formato de los datos o incluso error en el servidor, veríamos una salida como:

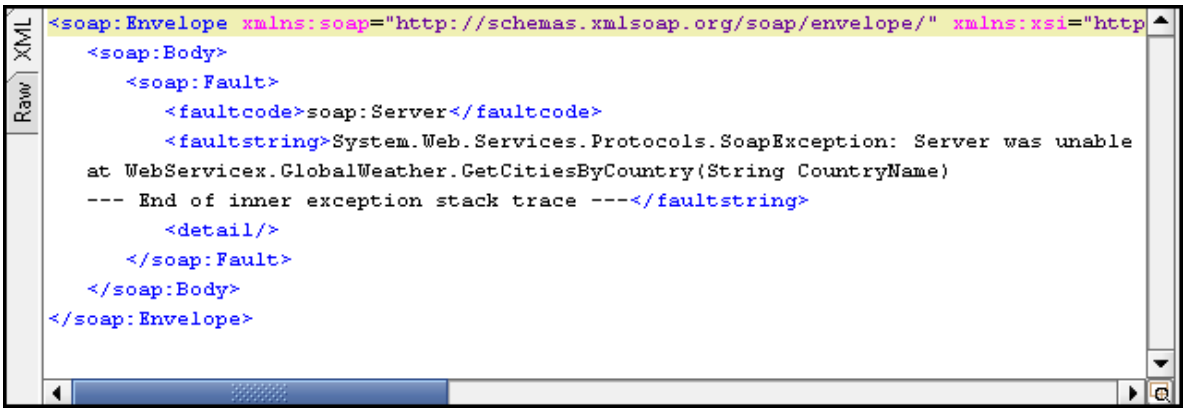


Ilustración 20 Ejemplo de respuesta errónea de un WebService

SONAR

Herramienta para controlar la calidad del código

Sonar es una plataforma que permite gestionar la calidad del código controlando los 7 ejes principales:

- Arquitectura y diseño
- Duplicaciones
- Pruebas unitarias
- Complejidad
- Errores potenciales
- Reglas de codificación
- Comentarios

Sonar realiza varios análisis de nuestro código a través de otras herramientas (Checkstyle, PMD, Cobertura, etc.) y presenta de manera unificada a través de su interfaz la información generada por ellas en forma de métricas.
A través de la interfaz de Sonar podemos ver de forma detallada los puntos débiles de nuestro proyecto como errores potenciales en el código, escasez de comentarios, clases demasiado complejas, escasez de cobertura de las pruebas unitarias, y más.

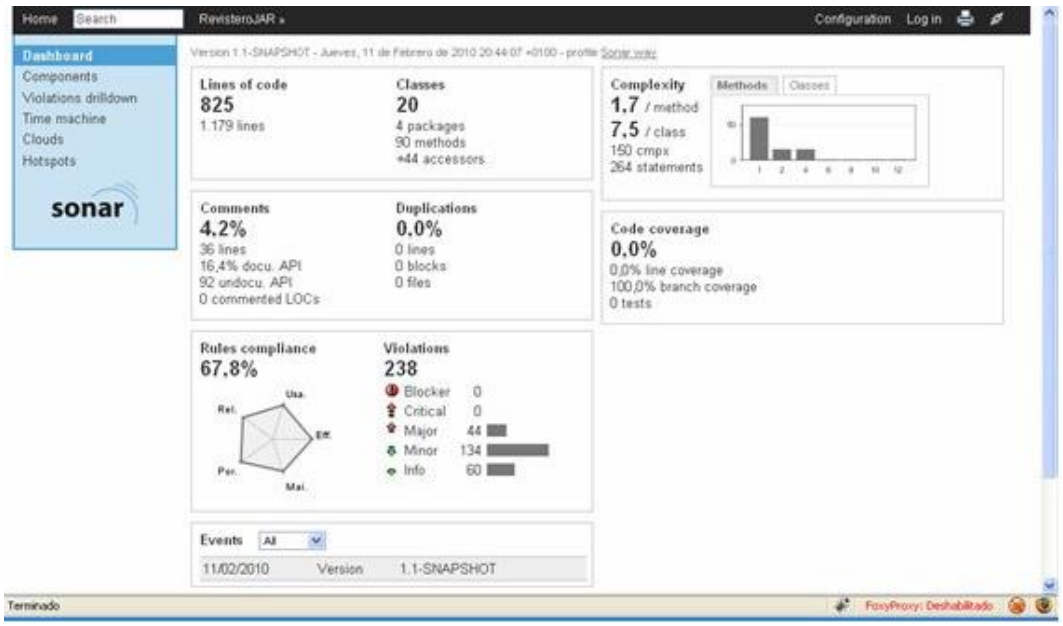


Ilustración 21 Reporte de un proyecto en SONAR

En una interfaz compacta nos muestra una cantidad importante de información sobre el código del proyecto como:

- Número de líneas de código
- Complejidad de los componentes
- Cantidad de comentarios introducidos
- Duplicidad del código
- Cobertura del código
- Conformidad respecto a ciertas reglas de codificación
- Violaciones del código

Para ampliar la información, podemos, por ejemplo, acceder a las violaciones tipo *major* de las reglas de codificación:

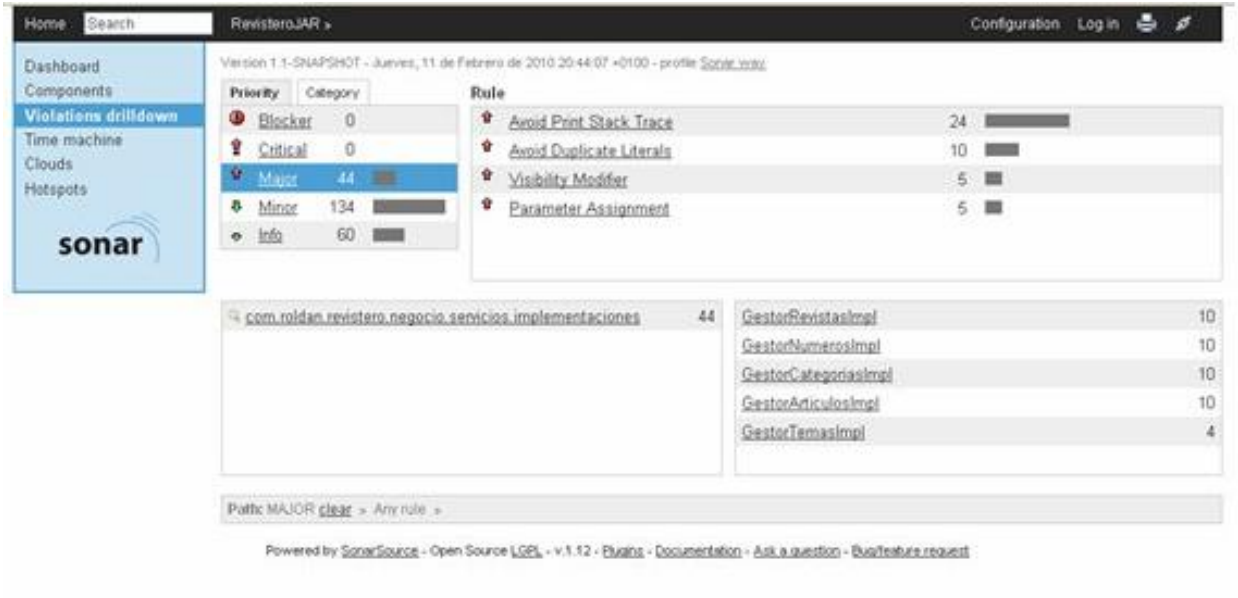


Ilustración 22 Violaciones mayores en el código

Sonar nos muestra información sobre las reglas que se están incumpliendo, y los paquetes y clases en que esto ocurre. Y, si damos clic sobre una de las clases Sonar muestra cuál es el código que está incumpliendo estas reglas y da indicaciones para corregirlo:

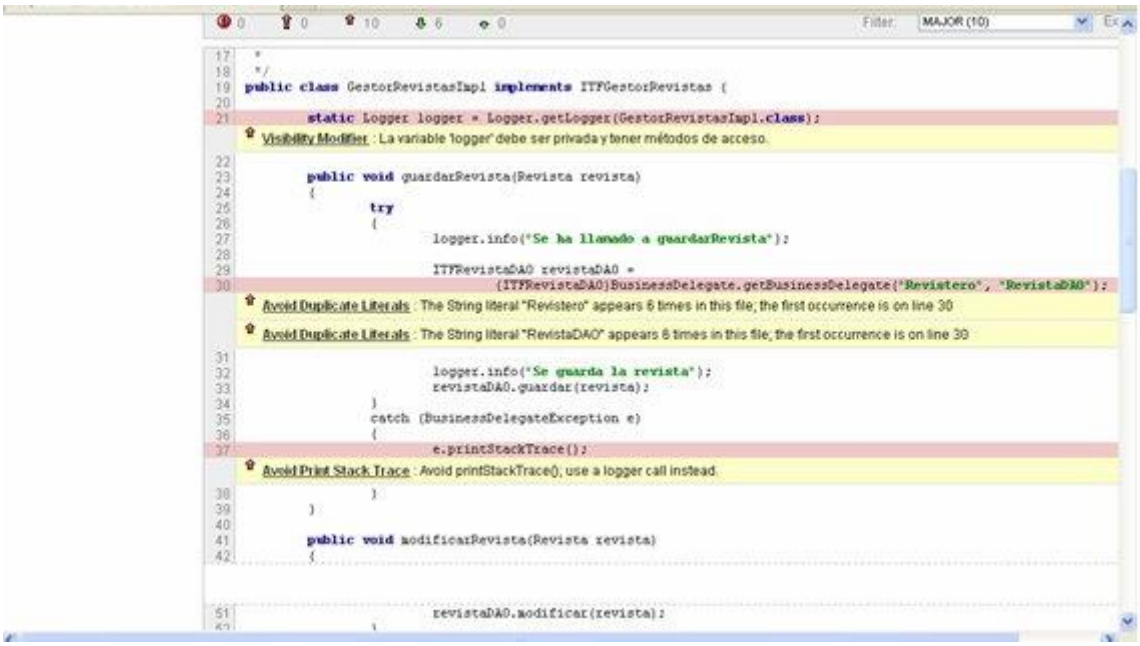


Ilustración 23 Detección de errores de calidad en código

Cada proyecto establece el nivel de calidad mínimo permitido para liberar sistemas a producción. No es lo mismo tener 95% de calidad para un proyecto de banca que para una red social. El nivel de exigencia suele determinarse por contrato y es tarea de los administradores de proyecto y de los líderes técnicos validar que se esté cumpliendo.

No es tarea fácil cumplir con estas métricas. Elevar el porcentaje de cobertura de pruebas, por ejemplo, puede elevar al doble el tiempo que un programador se tarda en construir un componente, aunque puede reducir el riesgo de tener bugs en sistemas productivos.

METODOLOGÍA DE DESARROLLO ÁGIL: SCRUM

Scrum es un framework para desarrollo ágil que facilita el trabajo colaborativo y reduce la complejidad de los ciclos de desarrollo.

Las actividades comunes en un equipo de Scrum son:

1. Un Scrum Owner crea una lista de tareas y las ordena por prioridad
2. En una reunión de planeación el equipo decide cuánto durará su iteración y que tareas cree poder terminar en ese periodo
3. Una vez iniciado el sprint, se hace una reunión diaria donde se revisan avances, obstáculos y siguientes actividades para el día.
4. Existe un rol de Scrum Master que tiene el objetivo de facilitar a los desarrolladores los insumos y contactos necesarios para continuar su trabajo.
5. Al final del sprint se realiza una reunión de retrospectiva para analizar que se hizo bien, que se puede mejorar y analizar qué cambios se pueden introducir en la siguiente iteración para mejorar la productividad.

Ilustración 24 Scrum

JIRA

Herramienta para la gestión de incidentes del proyecto

Jira es un sistema, basado en estándares JEE, para la administración de proyectos. Su principal objetivo es facilitar el trabajo en equipo y promover un ambiente ágil y centralizado del estado de las tareas de un proyecto.

Jira es una aplicación extremadamente flexible que permitirá comenzar a coordinar y controlar procesos semi-estructurados. Una vez que el equipo de trabajo esté familiarizado con el sistema y a medida que vaya definiendo procesos de trabajo, Jira puede transformase en un motor de procesos modelable de acuerdo a las necesidades. Es decir, Jira permite comenzar con una solución simple y flexible, para luego evolucionar a un sistema de procesos modelable

Beneficios de usar JIRA

- Simple, poderoso y amigable.
- Administración de actividades: tareas, trámites, defectos, procesos, requerimientos, ideas, etc.
- Permite adjuntar documentos a las actividades
- Poderoso sistema de búsqueda en lenguaje natural de actividades.
- Poderoso sistema de reportes (filtros).
- Notificaciones vía email.
- Compatible con casi todas las bases de datos.
- Fácil extensión e integración con otros sistemas gracias a sus plugins

Jira permite trabajar con tableros Kanban y Scrum. Cada proyecto puede adaptarse a una metodología u otra. En proyectos de creación de software por ejemplo, es más común utilizar Scrum debido a la planeación de iteraciones de tiempo fijo. Sin embargo, en proyectos de soporte donde las tareas llegan sin un comportamiento fijo, es más recomendable Kanban.

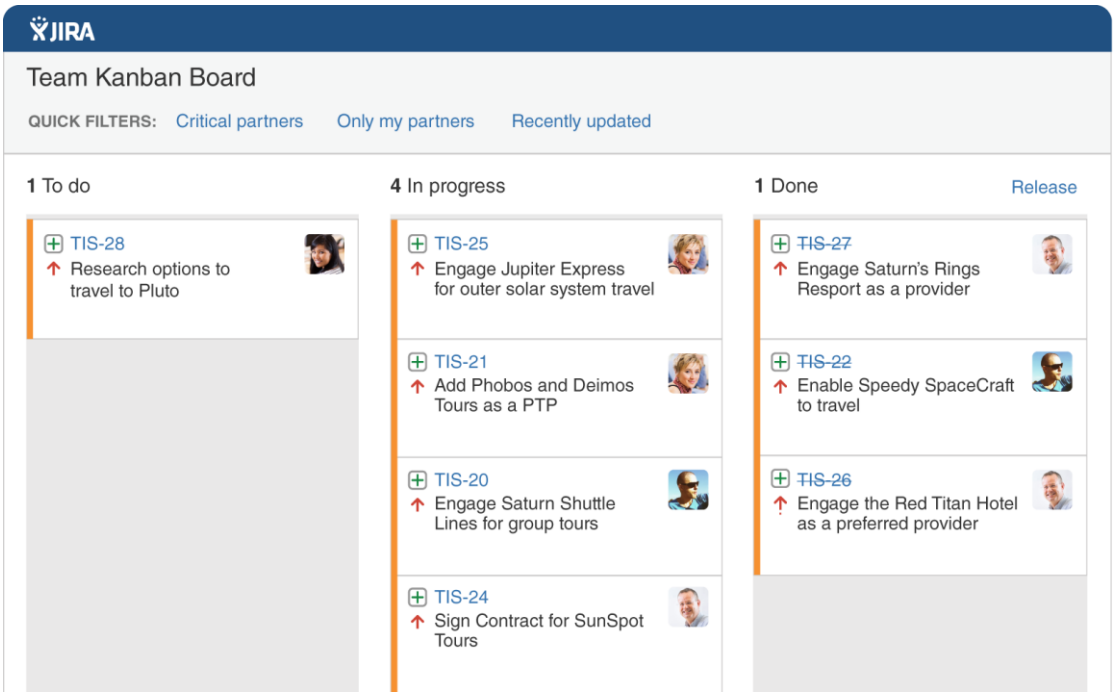


Ilustración 25 Ejemplo de un tablero Kanban en Jira

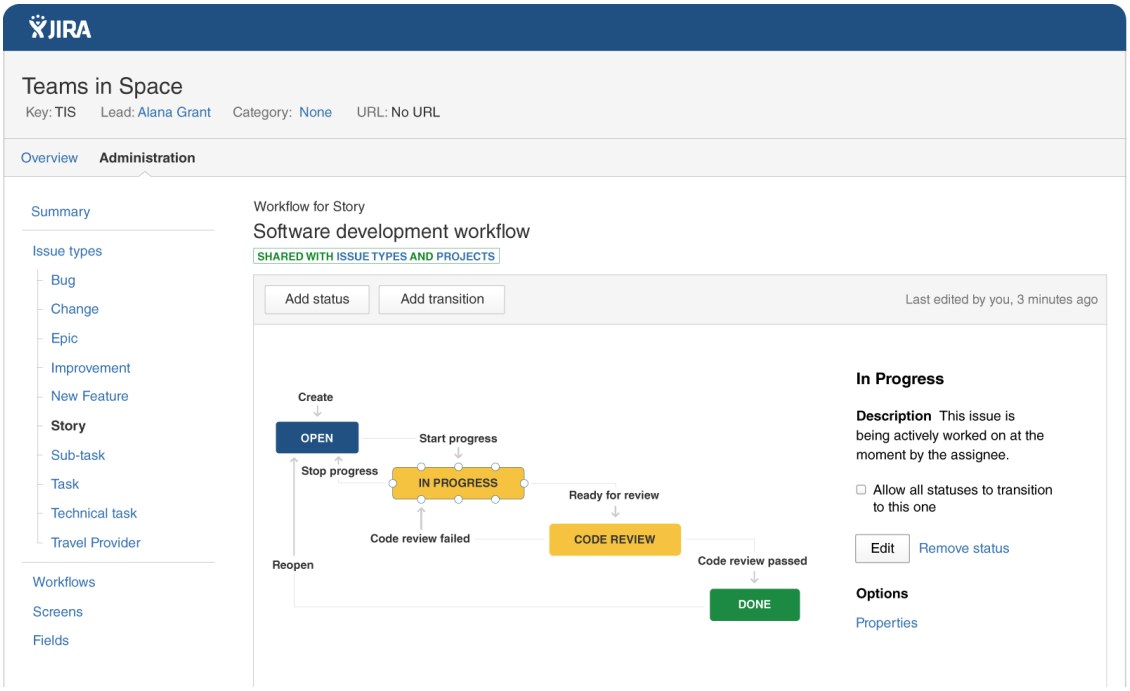


Ilustración 26 Flujo de trabajo en un proyecto de software

RESULTADOS

El conjunto de tecnologías y metodologías utilizadas demostraron ser eficientes para la construcción de un sistema de calidad y alta exigencia. La implementación de prácticas de desarrollo ágiles fomentó el trabajo colaborativo y dinámico que una aplicación de estas dimensiones requiere.

El sistema fue probado con movimientos de empleados afiliados al ISSSTE. Al ser estos trabajadores del estado, existen dependencias que tienen registrados cientos de miles de trabajadores y que cada bimestre tienen que ser reportados sus pagos ante el SAR. El día límite que tienen los centros de pago (Entidades o dependencias gubernamentales) es el 17 de cada mes non. En este día se tiene un pico de actividad muy importante. En ese día PROCESAR tiene que analizar los cientos de miles de registros enviados en lotes y generar líneas de captura para que los centros de pago realicen sus aportaciones en el banco.

Además de los movimientos en las subcuentas, también se pueden recibir solicitudes de modificaciones de datos de trabajadores, como actualizaciones de CURP, RFC, domicilio, etc. Dichos cambios requieren una interacción con varias dependencias externas al SAR, como lo es RENAPO. Este tipo de interacciones con otras dependencias gubernamentales y privadas, como en el caso de los bancos, exige una mayor sofisticación en los procesos y comunicaciones ya que la falla en estas puede repercutir en la confiabilidad y tiempo de respuesta de todo el proceso.

Una práctica común en la implementación final de proyectos como este es la creación de un “war room”. Consiste en un equipo de trabajo que representa a todas las áreas involucradas en el proyecto y cuyo objetivo es estar al pendiente del comportamiento del sistema durante las pruebas con el cliente. Ante alguna eventualidad, este equipo tiene la capacidad de responder rápidamente y realizar los ajustes necesarios para la afinación final del sistema.

Al final el sistema aprobó la prueba con el órgano del ISSSTE y gracias a su arquitectura y diseño será posible utilizarlo para otras entidades de salud pública como IMMS o el Seguro popular

CONCLUSIONES

La ingeniería de software es una rama de la ingeniería que evoluciona rápidamente. El uso de tecnologías robustas y probadas por la industria permite una mayor confiabilidad en el éxito de un sistema empresarial como el caso de SIRI. Como ingenieros de sistemas tenemos que estar constantemente capacitándonos y actualizándonos con las tecnologías de punta que están usándose en otras partes del mundo.

Gracias a la sólida formación que obtuve en la carrera y a mi experiencia en otros proyectos similares (gubernamentales y de equipos multidisciplinarios grandes) pude hacer buenas aportaciones al proyecto.

Trabajar en proyectos de este tipo representan un gran reto y al mismo tiempo una gran satisfacción ya que impactan a un gran número de personas en el país. La calidad en los sistemas y procesos involucrados en la administración de aportaciones para subcuentas de retiro y vivienda son de muchísima relevancia para los mexicanos. Errores en los sistemas pueden afectar negativamente la calidad de vida de las personas ya que están ligados al tema de jubilaciones y sus respectivas pensiones de las que dependen para su día a día.

Es muy importante que como profesionales estemos al tanto del impacto económico y social que tienen nuestras contribuciones. Requiere de un gran criterio y una visión mayor el poder ver el panorama completo de los proyectos de ingeniería de software. Nuestro país tiene aún muchas deficiencias en materia tecnológica pero la inversión en los últimos años demuestra una voluntad de los sectores públicos y privados para mejorar las condiciones del país.

La calidad de este proyecto no sólo se consiguió con la ingeniería en el desarrollo del sistema, sino también en la documentación y capacitación que se hizo con los usuarios. PROCESAR creó un sitio especial para entrenamiento en línea que permite a los usuarios saber cómo funciona el portal y que problemas pueden encontrar durante su uso, además de cómo solucionarlo.

También se creó una mesa de ayuda y un chat en vivo que amplían el abanico de posibilidades para obtener ayuda directa del equipo del proyecto SIRI.

BIBLIOGRAFÍA

www.consar.gob.mx

http://www.procesar.com.mx/

http://www.capacitacionprocesar.com/aulasiri

http://www.oracle.com/technetwork/java/javaee/tech/javaee5-jsp-135162.html

http://www.sparxsystems.com/products/ea/index.html

http://en.wikipedia.org/wiki/Unified_Modeling_Language

http://projects.spring.io/spring-framework/

http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html

https://www.atlassian.com/software/jira

http://www.sonarqube.org/

http://help.eclipse.org/luna/index.jsp

https://maven.apache.org/

http://junit.org/index.html

https://subversion.apache.org/

http://www.soapui.org/

http://www.gofpatterns.com/

Core J2EE Patterns: Best Practices and Design Strategies
Deepak Alur, John Crupi and Dan Malks

Prentice Hall / Sun Microsystems Press

Aprendiendo UML En 24 Horas,

Joseph Schmuller

PRENTICE-HALL

GLOSARIO

AFORES	Administradoras de Fondos para el Retiro
AOP	Aspect-Oriented Programming
API	Application Programming Interface
CGI	Common Gateway Interface
CONSAR	Comisión Nacional del Sistema de Ahorro para el Retiro
CURP	Clave Única de Registro de Población
CVS	Sistema de Control de Versiones
DRP	Disaster Recovery Plan
EJB	Enterprise Java Bean
FODA	Fuerzas, Oportunidades, Debilidades y Amenazas
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Entorno de Desarrollo Integrado
IMSS	Instituto Mexicano del Seguro Social
INFONAVIT	Instituto del Fondo Nacional de la Vivienda para los Trabajadores
ISEM	Instituto de Salud del Estado de México
ISSSTE	Instituto de Seguridad y Servicios Sociales de los Trabajadores del Estado
IV	Invalidez y Vida
IVCM	Invalidez, Vejez, Cesantía en Edad Avanzada y Muerte
J2ME	Java 2 Platform, Micro Edition
JAXB	Java Architecture for XML Binding
JAX-RPC	API Java para RPC
JAX-WS	API Java para XML Web Services
JDBC	Java Database Connectivity
JDT	Java Development Toolkit
JEE	Java Enterprise Edition
JMS	Java Message Service
JPA	Java Persistence API
JRE	Entorno de Ejecución Java
JSP	Java Server Pages
JVM	Java Virtual Machine
MDB	Message-Driven Beans
NAFIN	Nacional Financiera
ORM	Object-Relational Mapping
POJO	Plain Old Java Object
POM	Project Object Model

RCV	Retiro, Cesantía en Edad Avanzada y Vejez
RENAPO	Registro Nacional de Población
RFC	Registro Federal de Contribuyentes
RIA	Rich Internet Applications
SAR	Sistema de Ahorro para el Retiro
SAT	Servicio de Administración Tributaria
SFSB	Stateful Session Beans
SIEFORES	Sociedades de Inversión Especializadas de Fondos para el Retiro
SIRI	Sistema de Recepción de Información
SLSB	Stateless Session Beans
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
SUA	Sistema Único de Autodeterminación
TI	Tecnologías de la Información
UML	Unified Modeling Language
XML	Extensible Markup Language