UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de Software en Instituciones Financieras

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniera en Computación

P R E S E N T A

Blanca Gabriela Martínez Almaraz

ASESOR DE INFORME

Ing. Maricela Castañeda Perdomo

Ciudad Universitaria, Cd. Mx., 2016

AGRADECIMIENTOS

Es sumamente difícil agradecer a todas y cada una de las personas que me han acompañado a lo largo de mi recorrido en esta vida y que me ayudaron para que yo llegará hasta aquí.

Agradezco a mi mamá por todas esas noches en vela que se mantenía a mi lado para que no estuviera sola, por su apoyo incondicional, por su sabio y firme consejo: “Si uno pudo todos pueden, tú puedes” en los días que yo creía que ya no podía más, por su paciencia y sobre todo por el amor que me ha dado siempre. Sin duda es la “F” de mi FODA.

A mi papá por todo el empeño que puso para que yo pudiera ser quien soy y darme la confianza de que yo podía lograr lo que yo quisiera, por ejemplo estudiar ingeniería.

A mi novio, Paco, por ser parte fundamental en mi vida y el soporte que necesitaba, también le agradezco por esos días de estudio a mi lado, por nunca soltar mi mano y ayudarme a ser mejor cada día, gracias por siempre creer en mí.

A mi hermano, Paco, por siempre estar al pendiente de mí, por cada uno de los consejos que me dio y por ser un gran ejemplo a seguir.

A mi hermano, Marco, por su apoyo incondicional.

A mi sobrino, Paquito, por todas esas tardes que tuvo que esperar a que terminará la tarea para jugar conmigo. Te amo

A mi sobrina, Frida, por llegar en uno de los mejores momentos de mi vida y ser un motivo más para seguir adelante.

A la Ing. Maricela Castañeda por ayudarme a llegar a este momento de mi vida, por su paciencia y motivación.

A la UNAM por hacer de mi vida algo mejor, por todas y cada una de las veces que me enseñó a valorar el lugar privilegiado en el que me encontraba al estar en sus aulas. No tengo como pagar todo lo que me ha dado.

A todos y cada uno de los profesores que me enseñaron el camino para ser una Ingeniera con ética y profesionalismo.

Al Ing. Josué Martínez por permitirme ser parte de su equipo de trabajo y confiar en mí.

Hay tantos a quienes debo agradecerles como mí cuñada Lupita, mis amigos Memo, Luis y Graciela, muchas gracias por toda la confianza que pusieron en mí y por su apoyo siempre.

¡LO LOGRAMOS!

Hay que vigilar a los ingenieros. Comienzan con una máquina de coser y terminan con una bomba atómica

Marcel Pagnol

Contenido

1. Introducción

El presente trabajo ha sido realizado como un compendio sobre los diferentes proyectos, en la industria, en que he estado involucrada. Cada proyecto se ha llevado a cabo de diferente manera, ya que cada uno tiene particularidades y fines distintos.

Los proyectos en que he participado han empleado a Java como entorno de desarrollo, motivo por este trabajo se ha enfocado en incluir los principales *frameworks*, y algunas otras bibliotecas que fueron empleadas para su desarrollo.

Es importante mencionar que, con cada uno de los proyectos, se ha puesto en práctica lo aprendido en el transcurso de la licenciatura. A su vez considero que, tanto he generado experiencia laboral, como obtuve conocimientos propios del ámbito laboral, lo que complementa mi formación con carácter integral.

1.1 Objetivos

- Presentar el esquema de desarrollo de *software* empleado a nivel industrial.
- Enunciar diferentes tecnologías empleadas hoy en día a nivel industrial.
- Conocer las diferencias entre las metodologías usadas para el desarrollo de *software*, así como las tecnologías que se usan para el desarrollo de aplicaciones en plataforma *web*.

2. Ingeniería de *Software*

El concepto de "desarrollo de *software*" implica la construcción de una herramienta para solventar una necesidad, empleando únicamente la descripción de la misma. Por lo anterior, el conjunto de conocimientos y habilidades para realizar el desarrollo de *software* es considerado dentro del campo de la ingeniería. En general, la relación existente entre un *software* con su entorno es clara, ya que el *software* es introducido en el mundo con la finalidad de provocar efectos en el propio entorno.

Considerando lo anterior, podemos definir que "Ingeniería de *software*" es tanto la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de *software*, como el estudio de estos enfoques y su aplicación. Esta disciplina integra matemáticas, ciencias de la computación y prácticas cuyos orígenes se encuentran en la ingeniería. Dado lo anterior, el desarrollo de *software* se debe manejar como un proceso, y para ello es necesario crear un ciclo de vida que se ajuste al *software*. Este ciclo de vida dará la pauta para continuar o detener el desarrollo, es decir, que cada actividad tenga un inicio y una conclusión establecida.

El rol principal de la ingeniería basada de software basada en componentes consiste en conseguir el desarrollo de sistemas mediante un ensamblaje de sus partes (componentes), el desarrollo de dichas partes concebido como entidades reusables, y el mantenimiento y actualización de los sistemas mediante la personalización y, en su caso, el reemplazo de dichas partes. Lo anterior requiere el establecimiento de metodologías y herramientas de soporte que consideren el ciclo completo del sistema y sus componentes (Crnkovic 2001).

El desarrollo de software ágil representa un cambio significativo a partir de las metodologías tradicionales de desarrollo, basadas en planificación, usadas inicialmente en la ingeniería de software (Dybå y Dingsøyr 2008).

En los últimos años, el software se ha convertido en un componente vital para virtualmente todos los negocios. Derivado de ello, el éxito de las empresas depende, cada vez en mayor medida, del empleo de software como un arma competitiva (Herbsleb y Moitra 2001).

Desarrollar el producto correcto siempre es un reto importante, no solamente en el ámbito del desarrollo de software, ya que requiere no solamente buenas prácticas de ingeniería, sino también buenas decisiones de negocio (Wallin, Ekdahl y Larsson 2002).

La ingeniería de software busca reducir el tiempo, esfuerzo, costo y complejidad de los desarrollos de programas mediante el aprovechamiento de herramientas comunes dentro de una cartera de productos similares. La eficacia de este proceso depende de la forma en que se logre dar un mantenimiento y desarrollo adecuado durante el ciclo de vida de software (Völter y Groher 2007).

A partir de la década de 1990, investigadores han estudiado una amplia variedad de prácticas disponibles para el desarrollo de software, y las diferencias entre las usadas (y sus niveles de desempeño) alrededor del mundo (Cusumano, y otros 2003).

El uso de modelos de desarrollo de software para el diseño de sistemas complejos es una práctica de rigor para las disciplinas tradicionales de ingeniería. Nadie puede imaginar edificar algo tan complejo como un automóvil o un puente sin primero haber construido modelos y sistemas previos

especializados. Los modelos ayudan a entender un problema complejo, y a alcanzar potenciales soluciones al mismo mediante la abstracción. Por tanto, es obvio que para sistemas de software, que se encuentran entre las creaciones más complejas, el uso de modelos puede traer grandes beneficios (Selic 2003).

2.1 Ciclo de vida de *Software*

El ciclo de vida es el conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial, necesidad, hasta que el *software* es retirado o remplazado.

Entre las funciones con las que cuenta un ciclo de vida destacan:

- Determinar el orden de las fases del proceso de *software*, es necesario saber qué orden llevan las cosas y como se deben relacionar cada una de las actividades a realizar por los recursos.
- Establecer los criterios de transición para pasar de una fase a la siguiente, se debe definir una parte entregable por cada fase con el fin de obtener un objetivo tangible.
- Definir las entradas y salidas de cada fase; cada fase debe tener que se requiere para iniciar (entradas) la fase, así como el/los resultado(s) (salida(s)) que obtendremos de la misma.
- Describir los estados por los que pasa el producto, es decir lo que se hará en cada fase, lo que se obtendrá por cada paso en el que se avance, así como los alcances de cada fase.
- Describir las actividades a realizar en cada fase.
- Definir un esquema que sirve como base para planificar, organizar, coordinar, desarrollar, etc.

El ciclo de vida de un proyecto se compone de fases sucesivas, compuestas a su vez, por tareas que se pueden planificar. La sucesión de fases puede ampliarse con bucles de realimentación. Un bucle de realimentación es un mecanismo que, en cada iteración, suministra al sistema aportaciones de resultados intermedios que se van obteniéndose. Por lo tanto, lo que conceptualmente se considera una misma fase se pueda ejecutar más de una vez a lo largo de un proyecto.

El ciclo de vida depende del desarrollo de *software* que se requiere llevar a cabo. Para fines de este trabajo, en lo sucesivo haré mención de los diferentes procesos para el desarrollo de *software*, y haré una descripción de la forma es que se han aplicado en proyectos en que he intervenido. Además, haré mención de algunos otros modelos que considero importantes, ya que son de especial interés para adaptarse al desarrollo específico de *software*. En la Figura 1 se muestra un ejemplo general del ciclo de vida de *software*.

Figura 1. Ciclo de vida de Software

2.1.1 Modelo Cascada

Este modelo puntualiza que el desarrollo de *software* es realizado a través de una secuencia simple de fases (Figura 2), con flechas mostrando el flujo de información entre las fases. Ejemplos de estas etapas son: Análisis, Diseño, Implementación, Pruebas y Mantenimiento. Cada una de dichas fases tiene un conjunto de metas bien definidas. Además, este modelo utiliza puntos de control para continuar a la fase subsecuente.

El uso de este modelo puede implicar el uso extensivo de recursos, pues las fases van ligadas una con la subsecuente. La principal causa del fracaso del *software* desarrollado mediante este modelo subyace en la comunicación con el usuario final. El escenario es que se utilice en proyectos con requerimientos bien definidos.

En la actualidad, este modelo de desarrollo de *software* no tiene un uso generalizado, ya que es complicado definir el comportamiento externo antes de diseñar la arquitectura interna. Además, cabe destacar que no para todos los proyectos se pueden documentar los resultados de cada actividad realizada.

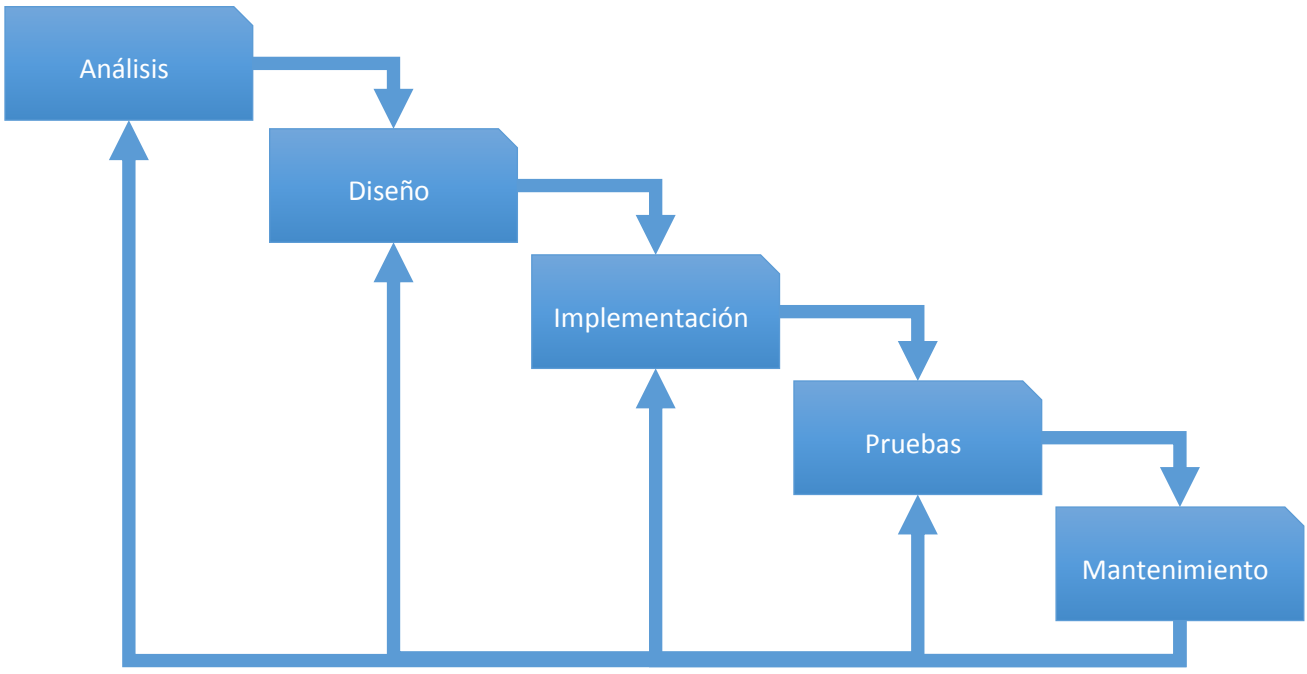


Figura 2. Modelo Cascada

Como se mencionó previamente, las fases de este modelo de desarrollo son, por lo general:

Análisis

En esta fase se deben considerar las necesidades del usuario del *software*, para así determinar los objetivos que se deberán cumplir. En esta fase se debe elaborar el documento denominado SRD, *Specification Requirements Document,* por sus siglas en inglés. Dicho documento contiene la especificación de lo que debe ejecutar el sistema, sin incluir detalles específicos. En esta fase se debe consensuar todo lo que se requiere que el sistema sea capaz de realizar. Consecuentemente, con esta información se obtendrá lo necesario para llevar a cabo cada fase posterior. En este modelo, no se permite el establecimiento de nuevos requerimientos o nuevos resultados, parciales, de algún proceso de elaboración del *software*.

Diseño

Puede ser de dos tipos:

- ✓ Diseño del sistema. En este aspecto, se descompone y organiza el sistema en elementos que puedan elaborarse por separado, aprovechando las ventajas del desarrollo en equipo. Como resultado, se genera un documento llamado *Software Design Document* (SDD), el que contiene la descripción de la estructura relacional global del sistema, la especificación de lo que debe hacer cada una de sus partes, y la manera en que se combinan unas con otras.
  Es conveniente distinguir entre "diseño de alto nivel" (o arquitectónico) y "diseño detallado". El primero de ellos tiene como objetivo definir la estructura de la solución (una vez que la fase de análisis ha descrito el problema) identificando grandes módulos (conjuntos de funciones que van a estar asociadas) y sus relaciones, con lo que se define la arquitectura de la solución elegida. El segundo tipo ("diseño detallado") define los algoritmos empleados y la organización del código para comenzar la implementación.

- ✓ Diseño del programa. Refiere a la fase en la que se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del usuario, así como también los análisis necesarios para saber las herramientas que deben usarse en la etapa de Codificación.

Implementación

Corresponde a la fase en que se aplica el código fuente, haciendo uso de prototipos así como de pruebas y ensayos para corregir errores.

Dependiendo del lenguaje de programación y su versión, se crean las bibliotecas y componentes reutilizables dentro del mismo proyecto, para hacer que la programación sea un proceso más rápido.

Pruebas

Los elementos, ya programados, se ensamblan para conformar el sistema, y se comprueba el correcto funcionamiento del mismo, así como la verificación de que cumple con los requisitos (actividades requeridas antes de ser entregado el desarrollo al usuario final). Posterior a las pruebas realizadas por los encargados del desarrollo, se llevan a cabo similares actividades con el usuario final, para validar que el sistema funcione conforme a los requerimientos.

Mantenimiento

Dado que se destina un 75 % de los recursos del desarrollo a esta etapa, el mantenimiento del *software* es imprescindible, ya que usuario final puede ser que no cumpla con todas sus expectativas, y hay que realizar actividades propias para mantener cubierto el alcance del proyecto.

2.1.2 Modelo incremental

En este modelo se desarrolla el sistema para, primeramente, satisfacer a un subconjunto de requisitos especificados, y en posteriores versiones se incrementa el sistema con nuevas funcionalidades que satisfagan más requisitos. Por tanto, combina elementos tanto del modelo de desarrollo en cascada como del modelo de desarrollo interactivo de construcción de prototipos. A su vez, se fundamenta en la construcción de *software* al ir incrementando las funcionalidades del programa. Para lograr lo anterior, este modelo aplica secuencias lineales, de forma escalonada, mientras progresa el tiempo de desarrollo en el calendario. Cada secuencia lineal produce, por tanto, un incremento de la funcionalidad de *software*. El diseño que ocupa el modelo incremental se muestra en la Figura 3.

Cuando se desarrolla *software* mediante un modelo incremental, el primer incremento consiste en constituirse en, a menudo, un producto esencial (sólo con los requisitos básicos). Dicho modelo se centra en la entrega de un producto operativo con cada incremento. Los primeros incrementos se constituyen como versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad tal como lo hará el producto final y también una plataforma para la evaluación.

Este modelo es muy similar al modelo en cascada, la diferencia está en la forma en que se aplican todas las fases de desarrollo a requerimientos específicos, por lo que se da solución a los mismos requerimientos sin necesidad de atrasar el desarrollo de *software*, y teniendo una meta fija constituida por entregables menos robustos y con mejor nivel de detalle.

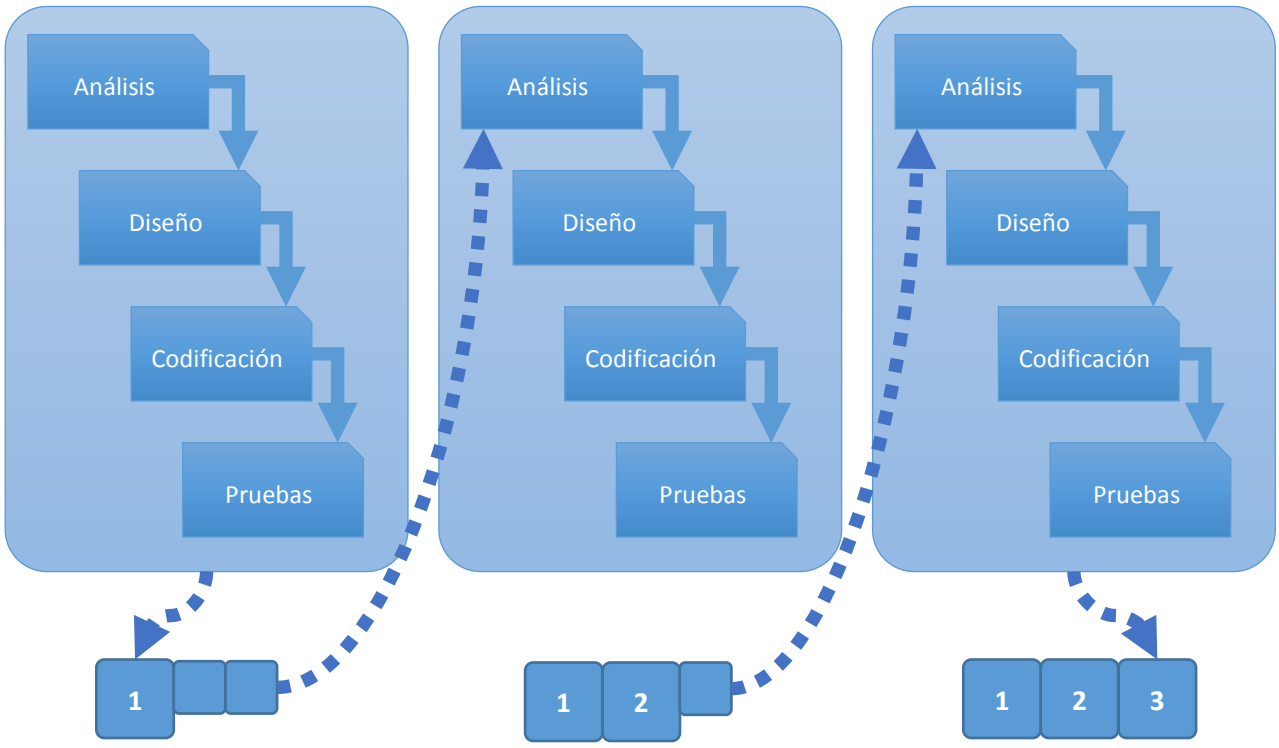


Figura 3. Ciclo de vida, modelo incremental

2.1.3 Modelo Espiral

En este modelo de desarrollo de *software*, el sistema se construye mediante una serie de versiones incrementales, por lo que es muy similar al Modelo Incremental. Durante las primeras iteraciones del desarrollo, la versión incremental podría constituirse como un modelo en papel o un prototipo. Durante las últimas iteraciones se producen versiones cada vez más completas de ingeniería del sistema.

El Modelo Espiral para el desarrollo de *software* se divide en una serie de actividades estructurales, las cuales también son llamadas "regiones de tareas", un ejemplo de las tareas se muestran en la Figura 4. Por lo general se definen de cuatro y seis regiones de tareas:

Comunicación con el cliente: En esta región de tarea se establece la comunicación entre el desarrollador y el cliente, ya sea para revisar especificaciones (toma de requerimientos), plantear necesidades, etc.

Planificación: Se realiza la definición de los recursos para desarrollar lo necesario para satisfacer las necesidades especificadas en el punto anterior, incluyendo tiempos e información relacionada con el proyecto.

Análisis de riesgos: En este apartado se deben evaluar tanto riesgos técnicos como de gestión.

Ingeniería: En esta región de tarea se debe construir al menos una representación de la aplicación, con la finalidad de verificar el aspecto y la funcionalidad, al menos de forma parcial.

Construcción y adaptación: En esta región se realiza la construcción, prueba e instalación de *software* desarrollado, además de otorgar soporte al usuario. Por lo general se da una capacitación para el uso del sistema.

Evaluación del cliente: En esta región se encuentran las tareas requeridas para obtener la reacción del cliente, según la evaluación de las representaciones del *software* creadas durante la etapa de ingeniería, e implementadas durante la etapa de instalación.

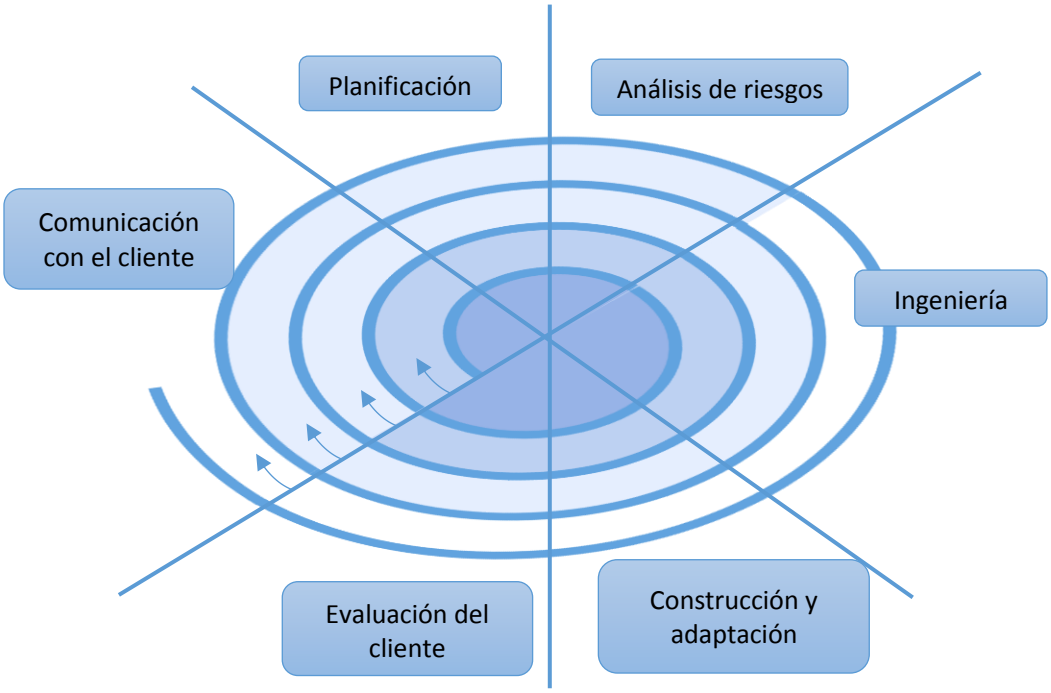


Figura 4. Ciclo de vida, modelo en espiral de seis regiones de tareas

Derivado de lo anterior, cada una de las regiones de tareas está constituida por pequeños requerimientos que se adaptan a las características del proyecto que va a emprenderse. Para proyectos pequeños el número de tareas a realizar y su nivel de formalidad es bajo, mientras que para proyectos mayores y más críticos, cada región contiene tareas que se definen para lograr un nivel más alto de formalidad.

Conforme va presentándose el proceso evolutivo de este método de desarrollo de *software*, las actividades y alcance del equipo de trabajo comienza a ser análogo al giro alrededor de las agujas del reloj. El primer circuito de la espiral produce el desarrollo de una especificación de productos, y los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones más sofisticadas del *software*. Cada paso de la región de planificación produce ajustes en el plan del proyecto, por lo que el costo y la planificación se ajustan en función de la evaluación del cliente. Además, el líder de proyecto ajusta el número planificado de iteraciones requeridas para completar el *software* de que se trate.

El desarrollo de *software* mediante el modelo en espiral es un enfoque realista del desarrollo de sistemas y de *software* en gran escala. Como el *software* evoluciona, a medida que progresa el proceso, el desarrollador y el usuario comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos. El modelo en espiral utiliza la construcción de prototipos como mecanismo de reducción de riesgos, pero lo que es más importante, permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto. A su vez, mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo interactivo que refleja mejor el mundo real. El modelo demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto, y si se aplica adecuadamente, debe reducir los riesgos antes de que se conviertan en problemas a resolver.

Tabla de comparación de Modelos de Ciclos de vida revisados en el punto de vista anterior

Modelo	Resumen	Ventajas	Desventajas
Cascada	Determina que ninguna de las etapas se puede comenzar si no se ha completado la fase anterior. El ciclo de vida de *software* tiene que seguir la secuencia.	• Es un modelo lineal y, por supuesto, los modelos lineales son las más simples de ser implementados. • La cantidad de recursos necesarios para implementar este modelo es mínimo. • La documentación se produce en cada etapa del desarrollo del modelo de cascada, lo que hace que la comprensión del producto a diseñar sea un procedimiento más sencillo (además de también simplificar el mantenimiento en un futuro). • Después de cada etapa importante de codificación de *software*, las pruebas se realicen para comprobar el correcto funcionamiento del sistema y validar las peticiones del usuario.	• Al ser un modelo lineal no se puede volver atrás, por lo que si la fase de diseño ha fallado, el desarrollo puede complicarse en las fases sucesivas. • La mayoría de las veces, el cliente no es lo suficientemente claro sobre lo que requiere del sistema, por lo cual cada cambio que se realice en el medio puede causar confusión para el equipo de trabajo. • Los cambios o errores que surgen en el *software* completo pueden causar mucho problema.
Incremental	Se divide en diferentes procesos pequeños que a su vez se dividen en 4 partes: Análisis, Diseño, Código y Prueba. Sin embargo, para la producción de *Software*, se usa el principio de trabajo en cadena o "Pipeline", utilizado en muchas otras formas de programación.	• Este método incremental reduce el tiempo de desarrollo inicial, ya que se implementa la funcionalidad parcial, con lo que se hacen entregas tempranas de partes operativas del sistema al cliente. • Resulta más sencillo acomodar cambios al acotar el tamaño de los incrementos. • Por su versatilidad, requiere de una planeación cuidadosa tanto a nivel administrativo como técnico. • Se mantiene al cliente en constante contacto con los resultados obtenidos en cada incremento.	• El modelo Incremental no es recomendable para casos de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos. • Requiere de metas claras para conocer el estado del proyecto.

Modelo	Resumen	Ventajas	Desventajas
Espiral	El *software* se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones la versión incremental podría ser un modelo en papel o un prototipo, y durante las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado.	• Como el *software* evoluciona a medida que progresa el proceso, el desarrollador y el cliente comprenden mejor los riesgos que se pueden generar en cada nivel evolutivo. • El modelo en espiral permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa. • El modelo en espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto y si se aplica adecuadamente debe reducir los riesgos antes de que se conviertan en problemas.	• Debido a su elevada complejidad, no se aconseja utilizarlo en pequeños sistemas. • Genera demasiado tiempo en el desarrollo de sistemas. • Si no existen grupos pequeños de trabajo no se puede trabajar en éste método.

2.1.4 Herramientas

En este apartado hablare sobre algunas herramientas útiles para los ciclos de vida de *Software* como lo son diagramas que nos ayudan al diseño del desarrollo de *software* y en algunos casos para la planeación del proyecto.

2.1.4.1 Diagrama de Gantt

Un diagrama de Gantt es la representación gráfica (Figura 5) del tiempo que dedicamos a cada una de las tareas en un proyecto concreto, siendo especialmente útil para mostrar la relación que existe entre el tiempo dedicado a una tarea y la carga de trabajo que supone. Una de sus limitaciones es que no muestra la relación de dependencia que pueda existir entre grupos de tareas.
Los diagramas de Gantt fueron ideados por Henry L. Gantt en 1917 con la intención de ofrecer un método óptimo para visualizar la situación de un proyecto.

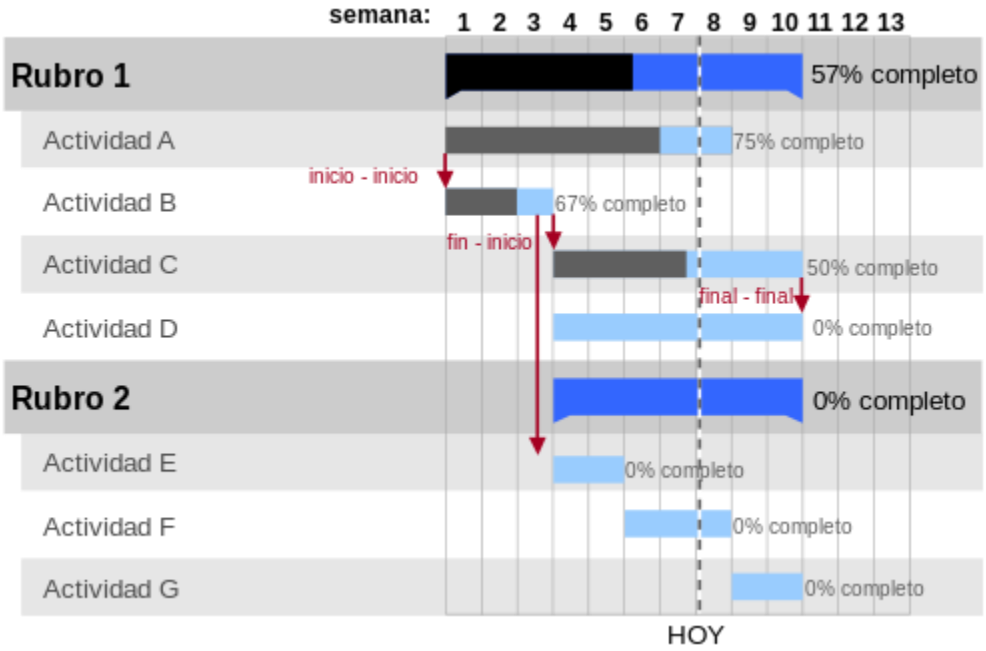


Figura 5. Ejemplo de diagrama de Gantt

En gestión de proyectos, el diagrama de Gantt muestra el origen y el final de las diferentes unidades mínimas de trabajo y los grupos de tareas o las dependencias entre unidades mínimas de trabajo (no mostradas en la imagen).

Desde su introducción los diagramas de Gantt se han convertido en una herramienta básica en la gestión de proyectos de todo tipo, con la finalidad de representar las diferentes fases, tareas y actividades programadas como parte de un proyecto o para mostrar una línea de tiempo en las diferentes actividades haciendo el método más eficiente.

Básicamente el diagrama está compuesto por un eje vertical donde se establecen las actividades que constituyen el trabajo que se va a ejecutar, y un eje horizontal que muestra en un calendario la duración de cada una de ellas.

2.1.4.2 UML (Unified Modeling Language)

Lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, es la sucesión de una serie de métodos de análisis y diseño orientadas a objetos que aparecen a fines de los 80's y principios de los 90s.UML es llamado un lenguaje de modelado, no un método. Los métodos consisten de ambos de un lenguaje de modelado y de un proceso.

UML es un lenguaje para hacer modelos y es independiente de los métodos de análisis y diseño. Existen diferencias importantes entre un método y un lenguaje de modelado. Un método es una manera explícita de estructurar el pensamiento y las acciones de cada individuo. Además, el método le dice al usuario qué hacer, cómo hacerlo, cuándo hacerlo y por qué hacerlo; mientras que el lenguaje de modelado carece de estas instrucciones. Los métodos contienen modelos y esos modelos son utilizados para describir algo y comunicar los resultados del uso del método

UML incrementa la capacidad de lo que se puede hacer con otros métodos de análisis y diseño orientados a objetos. Los autores de UML apuntaron también al modelado de sistemas distribuidos y concurrentes para asegurar que el lenguaje maneje adecuadamente estos dominios.

El lenguaje de modelado es la notación (principalmente gráfica) que usan los métodos para expresar un diseño. El proceso indica los pasos que se deben seguir para llegar a un diseño.

Principales beneficios de UML:

- Mejores tiempos totales de desarrollo (de 50 % o más).
- Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- Establecer conceptos y artefactos ejecutables.
- Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.
- Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización y minimización de costos.

Un lenguaje de modelado consiste de vistas, diagramas, elementos de modelo — los símbolos utilizados en los modelos — y un conjunto de mecanismos generales o reglas que indican cómo utilizar los elementos. Las reglas son sintácticas, semánticas y pragmáticas.

- *Vistas*: Las vistas muestran diferentes aspectos del sistema modelado. Una vista no es una gráfica, pero sí una abstracción que consiste en un número de diagramas y todos esos diagramas juntos muestran una "fotografía" completa del sistema. Las vistas también ligan el lenguaje de modelado a los métodos o procesos elegidos para el desarrollo. Las diferentes vistas que UML tiene son:

    - *Vista Caso de Uso:* Una vista que muestra la funcionalidad del sistema como la perciben los actores externos.

- *Vista Lógica:* Muestra cómo se diseña la funcionalidad dentro del sistema, en términos de la estructura estática y la conducta dinámica del sistema.

- *Vista de Componentes:* Muestra la organización de los componentes de código.

- *Vista Concurrente:* Muestra la concurrencia en el sistema, direccionando los problemas con la comunicación y sincronización que están presentes en un sistema concurrente.

- *Vista de Distribución:* muestra la distribución del sistema en la arquitectura física con computadoras y dispositivos llamados nodos.

- *Diagramas:* Los diagramas son los que describen el contenido de una vista. UML tiene nueve tipos de diagramas que son utilizados en combinación para proveer todas las vistas de un sistema: diagramas de caso de uso, de clases, de objetos, de estados, de secuencia, de colaboración, de actividad, de componentes y de distribución.

- *Símbolos o Elementos de modelo:* Los conceptos utilizados en los diagramas son los elementos de modelo que representan conceptos comunes orientados a objetos, tales como clases, objetos y mensajes, y las relaciones entre estos conceptos incluyendo la asociación, dependencia y generalización. Un elemento de modelo es utilizado en varios diagramas diferentes, pero siempre tiene el mismo significado y simbología.

- *Reglas o Mecanismos generales:* Proveen comentarios extras, información o semántica acerca del elemento de modelo; además proveen mecanismos de extensión para adaptar o extender UML a un método o proceso específico, organización o usuario.

2.2 Metodologías ágiles

En febrero de 2001, tras una reunión celebrada en Utah (EUA), nace el término "ágil" aplicado al desarrollo de *software*. En tal reunión participaron un grupo de 17 expertos de la industria de *software*, incluyendo algunos de los creadores o impulsores de metodologías de *software*. El objetivo de esta reunión fue esbozar los valores y principios que deberían permitir a los equipos desarrollar *software* rápidamente, respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Con esta metodología se buscó ofrecer una alternativa a los procesos de desarrollo de *software* tradicionales, caracterizados por ser rígidos y regidos por la documentación que se genera en cada una de las actividades desarrolladas. Consecuencia de esta reunión, se creó *The Agile Alliance*, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de *software* y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el *Manifiesto Ágil*, un documento que resume la filosofía "ágil".

Según el citado manifiesto sus postulados fundamentales son:

- *La gente es el principal factor de éxito de un proyecto de software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.*
- *La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental.*
- *Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha el proyecto y asegure su éxito.*
- *La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto determinan también el éxito o fracaso del mismo. Por lo tanto, la planificación no de ser estricta sino flexible y abierta.*

El desarrollo ágil de *software* refiere a métodos de ingeniería de *software* basados en el desarrollo iterativo e incremental, donde los requisitos y las soluciones requeridas evolucionan mediante la colaboración de grupos auto-organizados y multidisciplinarios.

Existen muchos métodos de desarrollo ágil. La mayoría tiene por finalidad el minimizar riesgos desarrollando *software* en lapsos cortos. El *software* desarrollado en una unidad de tiempo recibe el nombre de "iteración", la cual debe durar de una a cuatro semanas.

Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, sino que la meta es tener una "demo" (sin errores) al final de cada iteración. A su vez, con el final de cada iteración, el equipo de desarrollo vuelve a evaluar las prioridades del proyecto.

La finalidad de las metodologías ágiles consiste en la satisfacción del cliente, tiempos menores de desarrollo, y menor cantidad de defectos. El proceso de desarrollo ágil es de carácter iterativo e incremental, en el que los requerimientos pueden cambiarse en función de las necesidades del cliente, por lo que las actividades de planeación, análisis, diseño, codificación, prueba y mantenimiento deben llevarse a cabo de forma acorde con la demanda del cliente (Sharma, Sarkar y Gupta 2012).

El desarrollo de software mediante el uso de metodologías ágiles, incluso, parecen indicar que se tiene un efecto de generación de entusiasmo en los desarrolladores de software en los proyectos más dinámicos que se realicen (Syed-Abdullah, Holcombe y Gheorge 2006).

La ingeniería de software basada en componentes (CBSE, por sus siglas en inglés) es una disciplina intensiva basada en conocimiento en el que todas las actividades requieren el uso y transferencia de conocimiento entre los colaboradores, por lo que para afrontar los retos que la propia metodología enfrenta es necesario un mejor uso, transferencia y aplicación del conocimiento involucrado (Amine 2011).

La economía de internet ha cambiado sustancialmente las reglas aplicables a la ingeniería de software. Las metodologías tradicionales para el desarrollo de software han demostrado no ser capaces de adaptarse a la velocidad de cambios que se requieren, por lo que los desarrolladores de software han creado las llamadas “metodologías ágiles”, las que emplean desarrollo iterativo, , desarrollo de prototipos, plantillas y mínimos requerimientos de documentación. Una gran cantidad de factores que impactan en la implementación de una metodología ágil en la empresa en realidad están bajo el control de la propia empresa, por lo que ella debe ser capaz de maniobrar dichos factores para incrementar sus posibilidades de éxito (Livermore 2008).

En la comunidad de la ingeniería de software, se está viviendo con intensidad un debate entre los partidarios de las metodologías tradicionales (referidas como “metodologías pesadas”) y aquellos que apoyan las ideas emanadas del “Manifiesto Ágil”. Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y basados en nuevas tecnologías. Estas metodologías están especialmente orientadas a proyectos que necesitan de una solución a la medida, con una elevada simplificación sin dejar de lado el aseguramiento de la calidad en producto (Orjuela Duarte y Rojas 2008).

Las metodologías ágiles para el desarrollo están fundamentalmente basadas en la colaboración con los usuarios finales del software durante el proceso completo de desarrollo, la simplicidad para adaptar al producto en cambios de requerimientos, y en un desarrollo incremental. Basados en el “Manifiesto Ágil”, dichas metodologías se han aceptado como útiles en proyectos donde los requerimientos detallados son indefinidos inicialmente e identificados durante el proceso de desarrollo, derivado de la interacción entre los usuarios y el equipo de desarrollo (Mendes Calo, Elsa Clara y Fillottranni 2010).

El proceso de desarrollo de software asumido en las últimas dos décadas llevaba asociada una marcada tendencia hacia el control del proceso mediante una rigurosa definición de actividades, artefactos y roles. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado

ser efectivo en proyectos de gran envergadura donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el contexto es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. En la práctica, para muchos equipos de desarrollo, ante las dificultades para utilizar metodologías tradicionales, se llegó a la resignación de prescindir del "buen hacer" de la ingeniería de software con el objetivo de ajustarse a estas restricciones. Ante esta situación, las metodologías ágiles aparecen como una posible respuesta para llenar este vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida, con una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto (Letelier Torres y Sánchez López 2003).

Las metodologías ágiles han sido elegidas como las metodologías de programación apropiadas para proyectos de alta velocidad de desarrollo, volátiles y basadas en servicios de internet. A su vez, estas metodologías han sido criticadas, señalando una eventual falta de rigor y disciplina durante su uso. La realidad de esta situación depende de la filosofía con que estas metodologías sean incluidas y el entorno en el que ello sucede, siendo el principal tema a asegurarse el que no se caiga en los extremos de los enfoques de desarrollo (M. C. Paulk 2002).

A su vez, otros motivos que se exponen para criticar el empleo de metodologías ágiles están en la falta de atención que se coloca en la arquitectura y en el diseño formal, lo que implica que se tengan que tomar decisiones a pequeña escala que pueden influir de gran manera en el desempeño global del software desarrollado (Kumar y Kumar Bhatia 2012).

Distintas prácticas de desarrollo ágil de software, como XP (*Xtreme Programming*) o SCRUM han sido adoptadas de forma creciente para responder a los entornos cambiantes de negocios, donde los mercados y las tecnologías evolucionan rápidamente y se encuentra la empresa en espera de lo inesperado (Pikkarainen, y otros 2008).

Dentro de los principios de la metodología ágil no existe un rechazo per sé para efectuar la documentación de un proyecto, sino que en realidad representa un cambio de enfoque respecto de lo que en realidad representa el desarrollo de software. El cliente tiene que entender que una amplia documentación aumenta la utilización de recursos, lo que se traduce a su vez en un mayor costo y la entrega quizá más lenta del sistema. Siempre que sea posible, el equipo de desarrollo debe utilizar las herramientas que automatizan la generación de documentación para reducir el uso de recursos (Theunissen, Kourie y Watson 2003).

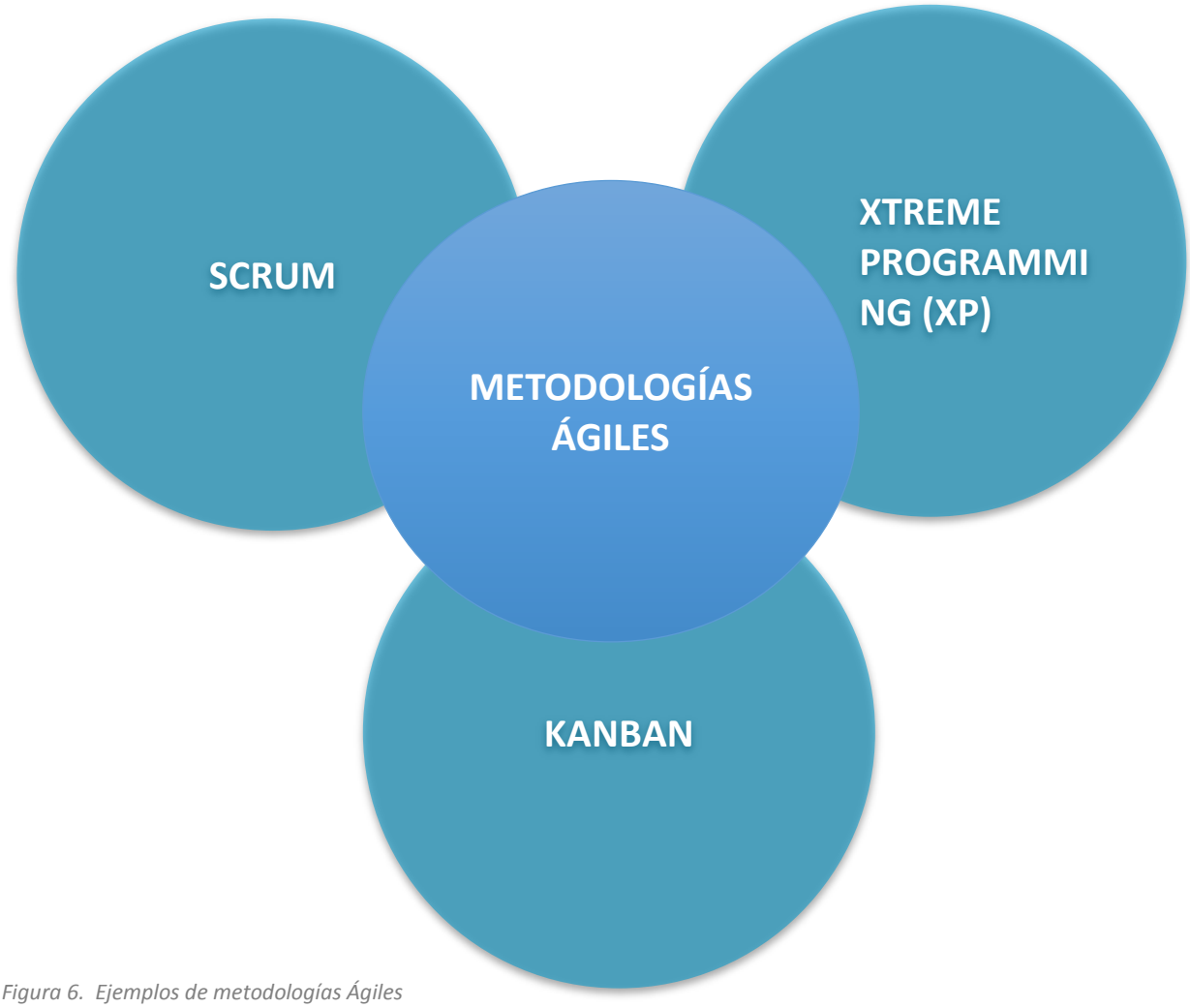


Figura 6. Ejemplos de metodologías Ágiles

A continuación incluyo la descripción de las tres metodologías ágiles enunciadas en la Figura 6, las cuales se han empleado en las empresas en donde he laborado, y personalmente me ha correspondido participar en desarrollos guiados por dos de estas metodologías.

2.2.1 Scrum

La metodología *Scrum* es un proceso en el que se aplican, de manera regular, un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

Sobre el producto final, se realizan entregas parciales y regulares del entregable final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, *Scrum* es especialmente en los proyectos en que se necesita obtener resultados en el menor tiempo posible y donde los requisitos son cambiantes o poco definidos.

Las características más marcadas que se notan en esta metodología son:

- Gestión regular de las expectativas del cliente,
- Resultados anticipados, flexibilidad y adaptación.
- Retorno de inversión, mitigación de riesgos, productividad y calidad.
- Alineamiento entre cliente y equipo, por último equipo motivado.

Cada uno de estos puntos mencionados permite que la metodología de *Scrum* sea utilizada de manera regular en un conjunto de buenas prácticas para el trabajo en equipo y, de esa manera, obtener resultados factibles.

Para poder llevar a cabo un proyecto con la metodología *Scrum*, es necesario definir los roles que tendrá el equipo de trabajo, así como el rol de los clientes con quienes se trabajará conjuntamente.

Roles involucrados en la metodología *Scrum*	
Roles principales	
Product Owner	Personifica la voz del cliente. Se asegura de que el equipo *Scrum* trabaje de forma adecuada desde la perspectiva del negocio. Además escribe historias de usuario, las prioriza, y las coloca en el Product Backlog
*Scrum*Master (o Facilitador)	La función primordial es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. Cabe señalar que no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. También se asegura de que el proceso *Scrum* se utiliza como es debido. Él es quien hace que las reglas se cumplan
Equipo de desarrollo	Tiene la responsabilidad de entregar el producto. Es recomendable un pequeño equipo de 3 a 9 personas con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc).
Roles auxiliares	
Stakeholders (Clientes, Proveedores, Vendedores, etc)	Se refiere a la gente que hace posible el proyecto y para quienes el proyecto producirán el beneficio acordado que justifica su producción. Sólo participan directamente durante las revisiones del sprint
Administradores (Managers)	Es el personal que establece el ambiente para el desarrollo del producto

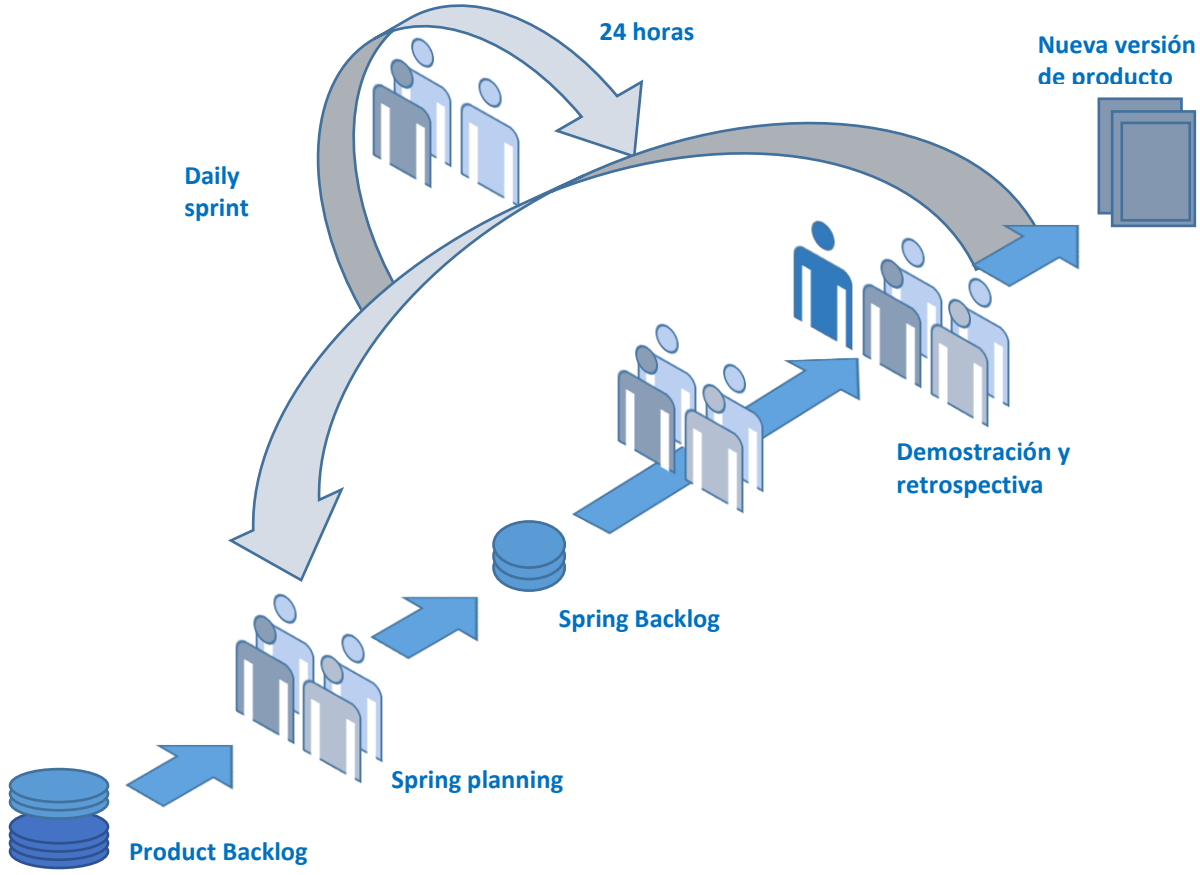


Figura 7. Metodología Scrum

La metodología SCRUM es una de las propuestas de desarrollo de metodologías ágiles que más presencia tiene actualmente, ya que involucra pocos procesos "burocráticos" durante el desarrollo de los sistemas. Este sistema ha sido desarrollado para ser empleado por pequeños equipos de trabajo (alrededor de 10 personas), y el software producido es producto de desarrollos incrementales basados en la colecta de requerimientos llevada a cabo en el proceso de creación del código (Fernandes y Sousa 2010).

La gestión de proyectos es una parte esencial de la Ingeniería de Software. Aun cuando no garantiza su éxito, usualmente una mala gestión conlleva al fracaso. Además, la selección adecuada de una metodología es trascendental para el éxito de un proyecto. SCRUM es una metodología para la gestión y control de proyectos, centrada en la construcción de software que satisface las necesidades del cliente, cumple con los objetivos del negocio y el equipo de desarrollo que construye el producto. A su vez, se combina fácilmente con otras metodologías de desarrollo. Las prácticas empleadas por SCRUM para mantener un control ágil en el proyecto son: i) Revisión de las iteraciones, ii) Desarrollo incremental, iii) Desarrollo evolutivo, iv) Auto-organización del equipo, y v) Colaboración (Alfonzo 2011).

La administración de desarrollo de proyectos usando la metodología SCRUM permite a varias compañías el establecimiento de buenas prácticas de negocio, tales casos como Fuji-Xerox, Honda, Canon, o Toyota. Dada la naturaleza de las empresas, es factible que los integrantes del equipo de desarrollo se encuentren físicamente en diferentes ubicaciones. La metodología SCRUM permite la colaboración en estos ambientes globalizados (Sutherland, y otros 2007).

SCRUM es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto, y además, éstas prácticas se apoyan unas a otras, la elección de SCRUM aportará al estudio la manera de trabajar en equipos colaborativos. Un principio clave de SCRUM es el reconocimiento de que durante un proyecto los participantes pueden cambiar de idea sobre lo que quieren y necesitan (a menudo llamado *requirements churn*), y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, SCRUM adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido (Rey y Lanza Castelli 2013).

La metodología SCRUM implica un desarrollo de software basado en un ciclo de vida predefinido basado en las metodologías ágiles, las que promueven inspecciones y adaptaciones frecuentes, una filosofía de liderazgo mediante el trabajo en equipo, autogestión y rendición de cuentas (Potter y Sakry 2009).

2.2.2 Kanban

La palabra *Kanban*, de origen japonés, se compone de dos términos: Kan que puede traducirse como "visual" y ban, como "tarjeta", siendo una traducción aproximada, "tarjeta visual".

Kanban se basa en un sistema de producción que enuncia las actividades a realizar sólo cuando existe capacidad para procesarles. El "disparador" de trabajo es representado por tarjetas K*anban*, de las cuales se dispone de una cantidad limitada (justamente, para asegurar que los recursos disponibles puedan disponerse en la justa medida para cumplir con el requerimiento de desarrollo).

Kanban Software Development (kanban) esta basada en la metodología de fabricación industrial del mismo nombre. Su objetivo es gestionar de manera general como se van completando tareas, pero también se ha utilizado en la gestión de proyectos de desarrollo de software. Se base en el desarrollo incremental, dividiendo el trabajo en partes. Una de las principales aportaciones es que utiliza técnicas visuales para ver la situación de cada tarea. No existen unas fases definidas, sino que se habla de un flujo (Sáez Martínez, y otros 2014).

La metodología Kanban se ha reportado también de utilidad para el seguimiento de tiempo muerto, lo que se puede corregir con la aplicación de un liderazgo efectivo en equipos con alto grado de autonomía en su gestión (Ikonen 2010).

Enfocándose a Tecnologías de la Información, el método kanban fue definido en 2010 como el conjunto de cuatro principios (empezar con las prácticas actuales, comprometerse a buscar cambios incrementales y evolutivos, animar el liderazgo, e inicialmente respetar procesos y responsabilidades), seis prácticas (visualizar el flujo de trabajo, limitar el trabajo en curso, establecer políticas explícitas de calidad, gestionar el flujo, establecer mecanismos de retroalimentación, y mejorar colaborando) y nueve valores (enfoque al cliente, acuerdo, colaboración, entendimiento, respeto, equilibrio, liderazgo, transparencia, y flujo), todos ellos entrelazados, a través de los cuáles una organización ágil subsiste, enfocando sus esfuerzos a la creación de valor para sus clientes y evolucionando continuamente (Bozheva de Berriprocess 2013).

Dentro de la investigación realizada, se encontró que los principales motivos por los cuales la metodología Kanban ha sido adoptada son su simplicidad, enfoque en el flujo de trabajo y su no obligatoriedad a la realización de iteraciones (Ahmad 2013).

En el pasado reciente, la adopción de la metodología Kanban se ha diseminado a lo largo del mundo, siendo variadas las razones por las que esto sucede: orientación nativa al servicio, mejora continua, flexibilidad para adaptarse a los cambios que solicite el cliente, entre otras (Anderson 2011).

Cada tarjeta *Kanban* acompaña a un ítem de trabajo durante todo el proceso de producción, hasta que este es empujado fuera del sistema, liberando una tarjeta (incluyendo a los recursos que fueron usados para su desarrollo). Consecuentemente, un nuevo ítem de trabajo sólo podrá ser ingresado o aceptado si se dispone de una tarjeta *Kanban* libre.

Derivado de lo anterior, este proceso de producción se denomina *pull* (tirar) en contraste con el mecanismo *push* (empujar), donde el trabajo se introduce al entorno de desarrollo en función de la demanda.

La metodología *Kanban* se basa en una serie de principios que la diferencian del resto de metodologías conocidas como ágiles. Dichas diferencias se enuncian a continuación:

- Todo lo que se hace debe salir bien a la primera, no hay margen de error. De aquí a que en *Kanban* no se premie la rapidez, sino la calidad final de las tareas realizadas. Esto se basa en el hecho que muchas veces cuesta más arreglarlo después que hacerlo bien a la primera.
- Se basa en hacer solamente lo justo y necesario, pero hacerlo bien. Esto supone la reducción de todo aquello que es superficial o secundario.
- *Kanban* no es simplemente un método de gestión, sino también un sistema de mejora en el desarrollo de proyectos, según los objetivos a alcanzar.
- Las actividades sucesivas se deciden con base en el *backlog* (o tareas pendientes acumuladas), pudiéndose priorizar aquellas tareas entrantes según las necesidades del momento (capacidad de dar respuesta a tareas imprevistas).

Para la aplicación del método *Kanban* es necesaria la generación de un tablero de tareas que permitirá mejorar el flujo de trabajo y alcanzar un ritmo sostenible. Para implantar esta metodología, es imprescindible tener claro los siguientes aspectos:

1. *Definir el flujo de trabajo de los proyectos*

   Se debe crear un tablero que presentará las labores a realizar, el que deberá ser visible y accesible por parte de todos los miembros del equipo de trabajo. Cada una de las columnas corresponderá a un estado concreto del flujo de tareas, y servirá para saber la situación en que se encuentra cada etapa del proyecto. El tablero debe tener tantas columnas como estados por los que pasa una tarea, desde que se inicia hasta que finaliza (diagnóstico, definición, programación, ejecución, *testing*, etc.).

   A diferencia de *Scrum*, una de las características del tablero es que opera de forma continua, lo que significa que no se compone de tarjetas que se van desplazando hasta que la actividad queda realizada por completo. En este caso, a medida que se avanza, las nuevas tareas (mejoras, corrección de incidencias o nuevas funcionalidades) se acumulan en la sección inicial, de manera que en las reuniones periódicas con el cliente se priorizan y se colocan dentro de la sección que se estima oportuna.

Dicho tablero puede ser específico para un proyecto en, o genérico. No hay unas fases del ciclo de producción establecidas sino que se definirán según el caso en cuestión, o se establecerá un modelo aplicable genéricamente para cualquier proyecto de la organización. Ejemplos de tableros se presentan en la Figura 8.

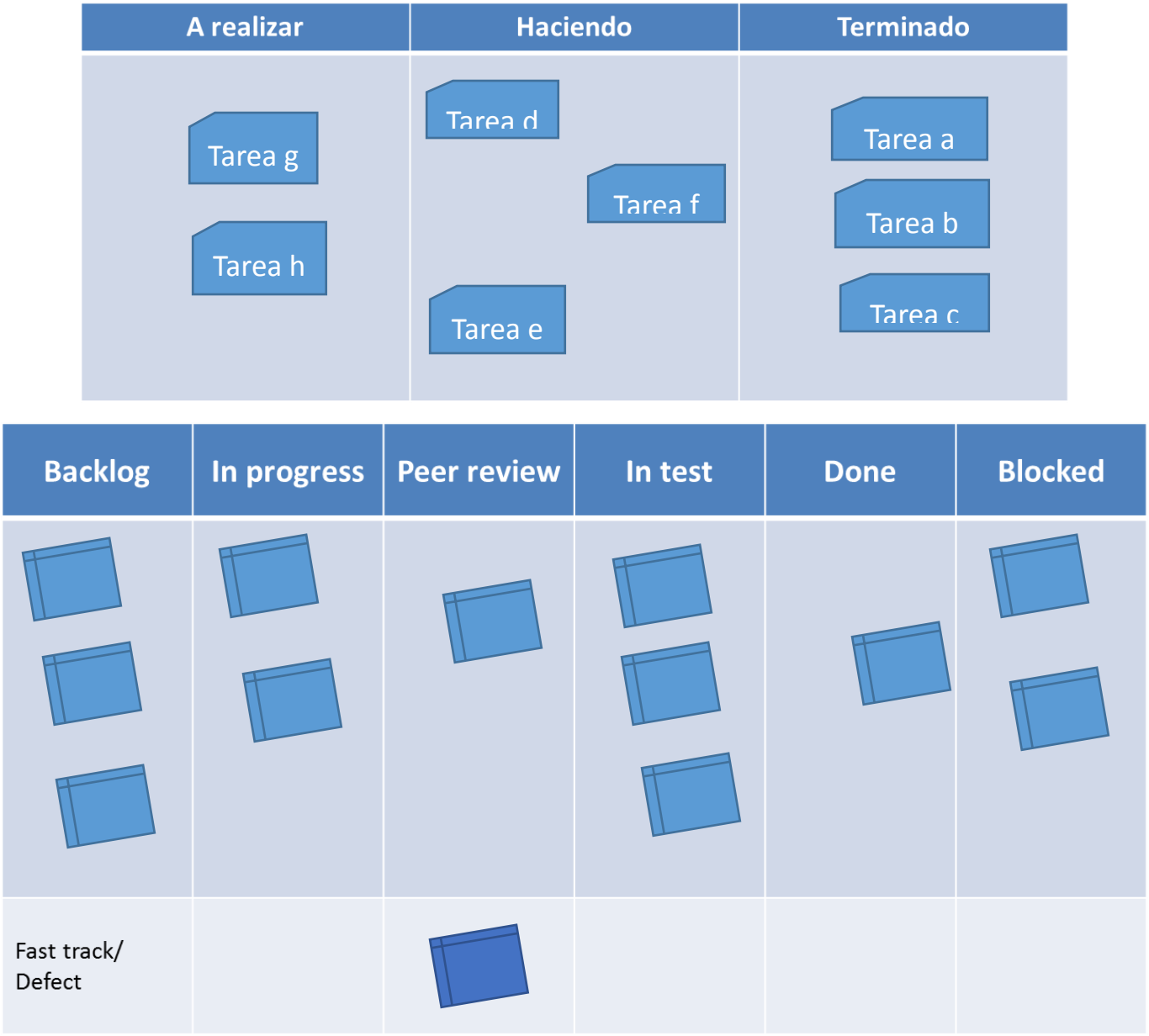


Figura 8. Ejemplos de tablero.

2. *Visualizar las fases del ciclo de producción*

   Al igual que *Scrum*, *Kanban* se basa en el principio de desarrollo incremental, es decir, dividiendo el trabajo. Esto significa que no se considera una la tarea en sí, sino que tal tarea es segmentada en distintos pasos para agilizar el proceso de producción.

   Normalmente, cada una de esas partes se escribe en una nota adherible y se pega literalmente en el tablero, en la fase que corresponda. Dichas notas contienen la información básica para que el equipo sepa rápidamente la carga total de trabajo que supone. Además, se pueden emplear fotos para asignar responsables, así como también usar tarjetas con distintas formas para poner observaciones o indicar bloqueos (cuando una tarea no puede hacerse ya que depende de otra).

   En realidad, el objetivo de la visualización es clarificar al máximo el trabajo a realizar, las tareas asignadas a cada equipo de trabajo (o departamento), así como también las prioridades y la meta asignada.

3. *Stop Starting, start finishing*

   Esta línea consiste en que se prioriza el trabajo que está en curso en vez de empezar nuevas tareas. Precisamente, una de las principales aportaciones del *Kanban* es que el trabajo en curso debe estar limitado y, por tanto, existe un número máximo de tareas a realizar en cada fase.

   En última instancia, busca definir el máximo número de tareas que se pueden tener en cada una de las fases y, por tanto, restringir el trabajo en curso. A esto, se le añade otra idea que, por muy obvia que pueda parecer, la práctica nos demuestra que no es así: no se puede abrir una nueva tarea sin finalizar otra. De esta manera, se pretende dar respuesta al problema habitual de muchas empresas de tener muchas tareas abiertas pero sin que estas se cierren de forma apropiada.

4. *Control del Flujo*

   A diferencia de *Scrum*, la metodología *Kanban* no se aplica a un único proyecto, sino que combina tareas y proyectos en su alcance. Como herramienta de control, busca mantener a los trabajadores con un flujo de trabajo constante, a las tareas más importantes en cola para ser desarrolladas y un seguimiento pasivo para no tener que interrumpir al trabajador en cada momento.

2.2.3 XP (Extreme Programming)

Esta metodología ágil está centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. *XP* se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y aptitud para enfrentar los cambios. *XP* se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Consiste básicamente en ajustarse estrictamente a una serie de reglas que se centran en las necesidades del cliente para lograr un producto de buena calidad en poco tiempo.

Al igual que en la metodología *Kanban*, es necesario tener en cuenta el rol de cada uno de los integrantes del equipo de trabajo. Dentro de estos roles también debe incluirse al cliente, con el fin de que forme parte activa durante el desarrollo, es decir, que evalúe, verifique y valide cada parte de los entregables que estén siendo generados en el desarrollo. Los roles sirven, por tanto, para determinar la actividad específica de cada uno de los integrantes del equipo de trabajo. Los principales roles en esta metodología son los siguientes:

Roles involucrados en la metodología *Scrum*	
Rol	Características
Programador	El programador escribe las pruebas unitarias y produce el código del sistema
Cliente	Escribe las historias de los usuarios y las pruebas funcionales para validar su implementación. El cliente da una gran prioridad a las historias de usuarios y decide cual implementar en cada iteración centrándose en aportar mayor valor al negocio.
Encargado de pruebas (*tester*)	Ayuda al cliente a escribir las pruebas funcionales. Se encarga de ejecutar las pruebas con regularidad, difunde los resultados obtenidos al equipo y es el responsable de las herramientas que dan soporte a las pruebas.
Encargado de seguimiento (*tracker*)	Proporciona la realimentación al equipo. Realiza el seguimiento del proceso de cada iteración y verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado en ello para la mejora de futuras estimaciones.
Entrenador (*coach*)	Responsable del proceso global. Se encarga de proveer guías al equipo de forma que se apliquen las practicas *XP* y se vaya siguiendo el proceso correctamente.
Consultor	Miembro externo del equipo con un conocimiento específico en algún tema que es necesario para el proyecto, en el que surjan problemas.
Gestor (*big boss*)	Vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es la de coordinación.

Las características fundamentales de la metodología son:

1. *Desarrollo iterativo e incremental,* implementación de pequeñas mejoras, en cuanto a su alcance, en una sucesión continua

2. *Pruebas unitarias continuas,* dichas pruebas se llevan a cabo de forma frecuente, de manera repetida y automatizada, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

3. *Programación en parejas,* se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto, de tal forma que el código es revisado y discutido mientras se escribe. Uno de los estandartes de esta característica es garantizar la calidad por encima de una posible pérdida de productividad inmediata.

4. *Frecuente integración del equipo de programación con el cliente o usuario,* se recomienda que un representante del cliente (en el mejor de los casos, el propio usuario directamente) trabaje junto al equipo de desarrollo.

5. *Corrección de todos los errores,* de forma previa a la adición de una nueva funcionalidad, se considera la corrección de todas las incidencias que se encuentren. Derivado de lo anterior, es previsible realizar una entrega frecuente del proyecto a desarrollar.

6. *Refactorización del código,* implica la reescritura ciertas partes del código para aumentar su legibilidad y mantenibilidad, sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

7. *Propiedad del código compartida,* en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.

8. *Simplicidad en el código,* cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Dado que las organizaciones dedicadas al desarrollo de software tienden a enfocarse a sistemas basados en la red (web-based systems), y dado que es un entorno dinámico, las metodologías ágiles permiten incrementar la productividad mientras se mantiene la calidad del producto y su flexibilidad. Una de las metodologías ágiles se conoce como XP (Xtreme Programming), apropiada para el trabajo en equipo de entre 5 y 15 desarrolladores (Maurer y Martel 2002).

Cantidad importante de organizaciones están adoptando prácticas asociadas con la metodología XP (Xtreme Programming). No obstante, la eficacia de la aplicación de esta metodología ha sido considerada de forma general como anecdótica, por lo que un área de oportunidad importante se constituye al encontrar mecanismos que permitan la evaluación sistemática de los beneficios obtenidos mediante el uso de esta metodología (Williams, y otros 2004).

Existe una amplia cantidad de citas y argumentos que indican el uso exitoso de esta metodología, no obstante evidencia concreta de su aplicación exitosa parece no estar del todo clara (Erickson, Lyytinen y Siau 2005).

La metodología XP es una "metodología ágil" que algunas personas señalan como de aplicabilidad para entornos cambiantes, escenarios volátiles y desarrollo de software basado en internet. Dado que la XP es un proceso riguroso, ha sido usado como una contraparte en oposición de modelos rígidos, tales como el *Software Capability Maturity Model (SCMM)*. (M. Paulk 2001).

La metodología XP, y en general las metodologías ágiles han emergido como métodos alternativos para abordar también proyectos extensos. Mientras la industria continúa desarrollándose, la metodología seguirá adaptándose para cubrir con las necesidades del entorno cambiante (Lindstrom y Jeffries 2004).

A continuación me permito colocar una tabla que resume a las características más relevantes de las metodologías de desarrollo que acaban de ser enunciadas.

COMPARATIVA DE LAS PRINCIPALES METODOLOGÍAS ÁGILES		
Scrum		
Resumen	Ventajas	Desventajas
Define un conjunto de prácticas y roles. Los roles principales en *Scrum* son el *ScrumMaster*, que procura facilitar la aplicación de *Scrum* y gestionar cambios, el *ProductOwner*, que representa a los *stakeholders* (interesados externos o internos), y el *Team* que ejecuta el desarrollo y demás elementos relacionados con el mismo. Durante cada sprint (un periodo entre una y cuatro semanas, cuya magnitud es definida por el equipo), se crea un incremento de *software* potencialmente entregable.	• Gran capacidad de reacción ante los cambiantes requerimientos generados por las necesidades del cliente o la evolución del mercado. • El trabajo metódico y la necesidad de obtener una versión de trabajo funcional después de cada iteración, ayuda a la obtención de un *software* de alta calidad. • Llevar a cabo las funcionalidades de mayor valor en primer lugar y saber la velocidad a la que el equipo avanza en el proyecto, permite despejar riesgos de manera anticipada.	• Parte del paradigma consiste en usar el método “tal cual”, evitando adaptarlo a la empresa. • Presupone que el cliente está muy involucrado en el desarrollo, participa de forma activa y continua, y revisa frecuentemente el avance de la funcionalidad conforme salen a la luz. Esto sin embargo no parece producirse en la mayoría de nuestros proyectos: el cliente participa, pero no hasta el punto de dedicar tiempo y recursos para revisar pequeños avances en el desarrollo.
KANBAN		
Resumen	Ventajas	Desventajas
El método *Kanban* es una aproximación al proceso gradual, evolutivo y al cambio de sistemas para las organizaciones. Utiliza un sistema de extracción limitada del trabajo en curso como mecanismo básico para exponer los problemas de funcionamiento del sistema (o proceso) y estimular la colaboración para la mejora continua del sistema. Un ejemplo del sistema de extracción es el sistema *Kanban*, y es después de esta popular forma de trabajo en curso, que se ha denominado el método.	• Disminuir o eliminar el stock intermedios (entre procesos). • Cumplir los tiempos de entrega demandados por el cliente. • Mejorar la calidad del producto por una mejor detección de los defectos del mismo.	• El sistema no tiene ninguna anticipación en caso de fluctuaciones en la demanda. • Es difícil de imponer este método a los proveedores. Las aplicaciones son limitadas. • El método es aplicable a producciones de tipo "masa" en que el número de referencias no es muy elevado, y la petición es regular o a reducidas variaciones. • Reducir el número de Kanban sin aportar mejoramientos radicales al sistema de producción, arrastrará retrasos de entrega y de espera entre operaciones.
XP (Extreme Programming)		

COMPARATIVA DE LAS PRINCIPALES METODOLOGÍAS ÁGILES		
XP se define como especialmente adecuada para proyectos (pequeños, medianos y grandes) con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.	• Programación organizada. • Menor taza de errores. Al ir resolviendo los cambios a la par del desarrollo los errores disminuyen • Satisfacción del programador	• Es recomendable emplearlo solo en proyectos a corto plazo. • Altas comisiones en caso de fallar.

3. Tecnologías para aplicaciones Web en Java

3.1 Introducción a Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo.

El objetivo era utilizarlo en un set-top box, un tipo de dispositivo que se encarga de la recepción y la descodificación de la señal televisiva pero la idea fracasó. El primer nombre del lenguaje fue Oak, luego se conoció como Green y finamente adoptó la denominación de Java. Uno de los fundadores de Sun rescató la idea para utilizarla en el ámbito de Internet y convirtieron a Java en un lenguaje potente, seguro y universal gracias a que lo puede utilizar todo el mundo y es gratuito. Una de los primeros triunfos de Java fue que se integró en el navegador Netscape y permitía ejecutar programas dentro de una página web, hasta entonces impensable con el HTML.

Actualmente Java se utiliza en un amplio abanico de posibilidades y casi cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas. Para lo que nos interesa a nosotros, con Java podemos programar páginas web dinámicas, con accesos a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. En general, cualquier aplicación que deseemos hacer con acceso a través web se puede hacer utilizando Java.

En el lenguaje Java sus objetivos principales son:

1. *Usar el paradigma de la programación orientada a objetos:* Se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el *software* de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización de *software* entre proyectos, una de las premisas fundamentales de la Ingeniería de *Software*.

   En este punto me parece importante mencionar que gracias a la creación de objetos podemos reutilizar código sin necesidad de volver a desarrollar, simplemente basta con heredar objetos de la clase padre.

2. *Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos:* La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run anywhere".

   Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode) — instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el

código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

Podemos citar como algunas de las funcionalidades de Java:

1. *Aplicaciones "cliente":* son las que se ejecutan en un solo ordenador (por ejemplo el portátil de tu casa) sin necesidad de conectarse a otra máquina. Pueden servirte por ejemplo para realizar cálculos o gestionar datos.

2. *Aplicaciones "cliente/servidor":* son programas que necesitan conectarse a otra máquina (por ejemplo un servidor de datos) para pedirle algún servicio de forma más o menos continua, como podría ser el uso de una base de datos. Pueden servir por ejemplo para el homeOffice: trabajar desde casa pero conectados a un ordenador de una empresa.

3. *Podemos hablar también de "aplicaciones web":* son programas Java que se ejecutan en un servidor de páginas web. Estas aplicaciones reciben "solicitudes" desde un ordenador y envían al navegador (Internet Explorer, Firefox, Safari, etc.) que actúa como su cliente páginas de respuesta en HTML.

Los principales recursos que tiene Java son:

- *JRE (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java):* Es el *software* necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes *software* o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK

- *API's:* Sun define tres plataformas en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus APIs (Application Program Interface) de forma que pertenezcan a cada una de las plataformas:

- Java ME (Java Platform, Micro Edition) o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.
- Java SE (Java Platform, Standard Edition) o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.
- Java EE (Java Platform, Enterprise Edition) o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Las clases en las APIs de Java se organizan en grupos disjuntos llamados paquetes. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información sobre los paquetes que ofrece cada plataforma puede encontrarse en la documentación de ésta.

El conjunto de las APIs es controlado por Sun Microsystems junto con otras entidades o personas a través del programa JCP (Java Community Process). Las compañías o individuos participantes del JCP pueden influir de forma activa en el diseño y desarrollo de las APIs, algo que ha sido motivo de controversia.

Para el paradigma orientado a objetos es necesario considerar algunas características de cómo debe ser manipulado comenzando por la terminología que indispensable en Java, a continuación explico definiciones importantes sobre Java:

Objeto: Los objetos son la clave para entender la tecnología orientada a objetos. A nuestro alrededor tenemos muchos ejemplos de objetos del mundo real: su perro, su escritorio, su televisor, su bicicleta. Un ejemplo de objeto en programación orientada a objetos se muestra en la Figura 9.

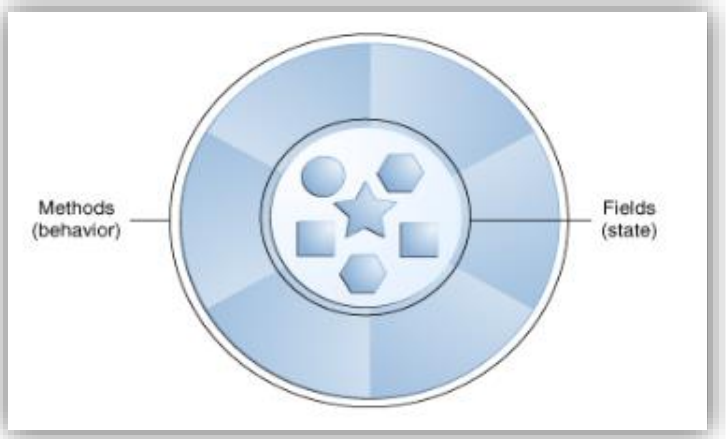


Figura 9. Ejemplo de Objeto de Java

Los objetos del mundo real comparten dos características: Todos tienen estado y comportamiento. Los perros tienen estado (nombre, color, raza, tiene hambre) y comportamiento (ladrando, buscando, moviendo la cola).

Los objetos de Java son conceptualmente similares a los objetos del mundo real: también consisten de estado y comportamiento. Un objeto almacena su estado en campos “variables” y muestra su comportamiento a través de métodos “funciones”. Los métodos operan sobre el estado interno del objeto y sirven como el mecanismo principal para la comunicación entre objetos.

Encapsulamiento: Este concepto consiste en la ocultación del estado o de los datos de un objeto, de forma que sólo es posible modificar los mismos mediante los métodos definidos para dicho objeto. De manera gráfica el encapsulamiento se muestra en la Figura 10.

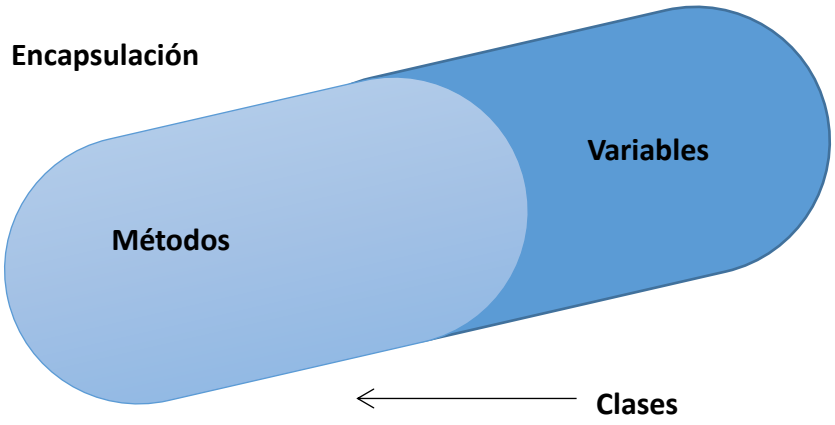


Figura 10. Ejemplo de encapsulamiento en Java

Cada objeto está aislado del exterior, de forma que la aplicación es un conjunto de objetos que colaboran entre sí mediante el paso de mensajes invocando sus operaciones o métodos. De esta forma, los detalles de implementación permanecen "ocultos" a las personas que usan las clases, evitando así modificaciones o accesos indebidos a los datos que almacenan las clases.

Clase: Abstracción que define un tipo de objeto especificando qué propiedades, estado del objeto, y operaciones, comportamiento del objeto, disponibles va a tener. También Puede considerarse como una plantilla o prototipo de objetos. En la Figura 11 se puede observar un ejemplo grafico de una clase en Java.

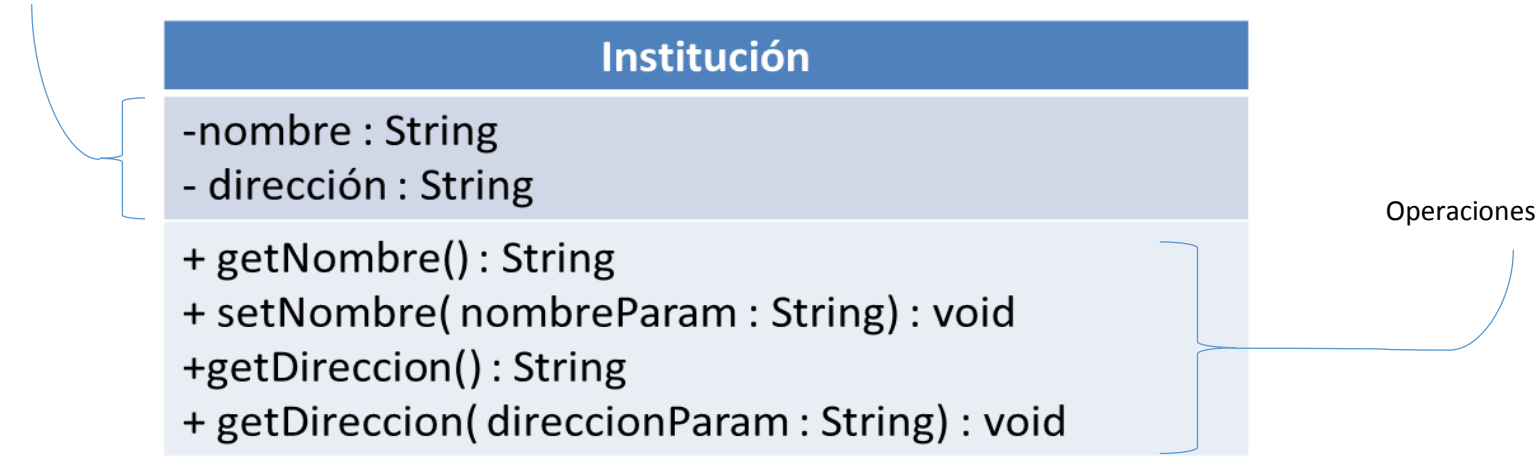


Figura 11. Ejemplo de Clase en Java

Herencia: Posibilita la definición de una clase a partir de otra. Cuando heredamos de una clase existente, reusamos (o heredamos) métodos y campos, y agregamos nuevos campos y métodos para cumplir con la situación nueva. Como ejemplo en la Figura 12 se muestra cómo es que las clases tienen herencia.

La clase ya existente es llamada superclass, o clase base, o clase padre y la clase nueva es llamada subclase, clase derivada, o clase hija.

A través de la herencia podemos agregar nuevos campos, y podemos agregar o sobre montar métodos (override). Sobre montar un método es redefinirlo en la clase heredada.

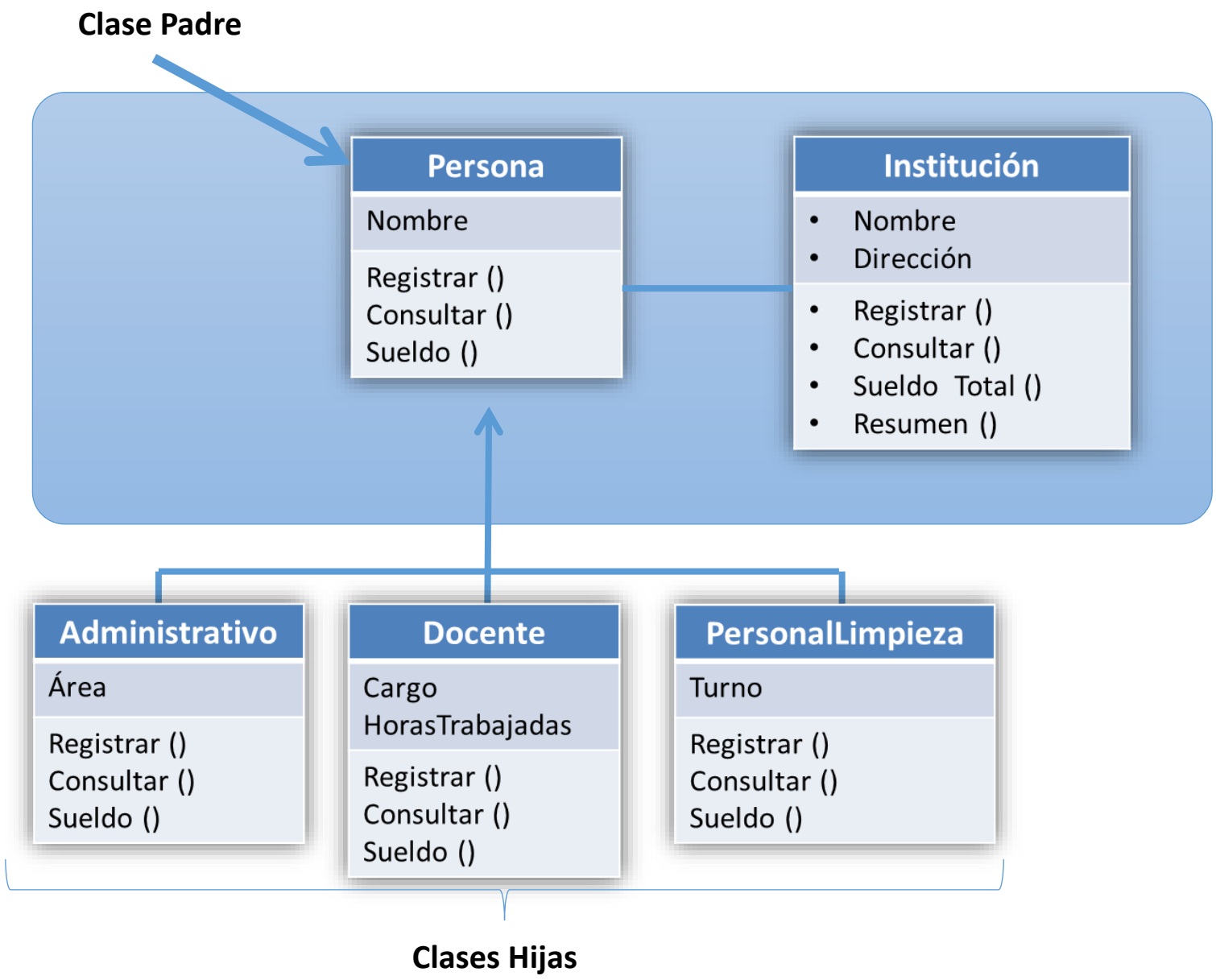


Figura 12. Ejemplo de herencia en Java

Polimorfismo: El polimorfismo es un concepto de la programación orientada a objetos que nos permite programar en forma general, en lugar de hacerlo en forma específica. En general nos sirve para programar objetos con características comunes y que todos estos compartan la misma superclase en una jerarquía de clases, como si todas fueran objetos de la superclase (Figura 13).

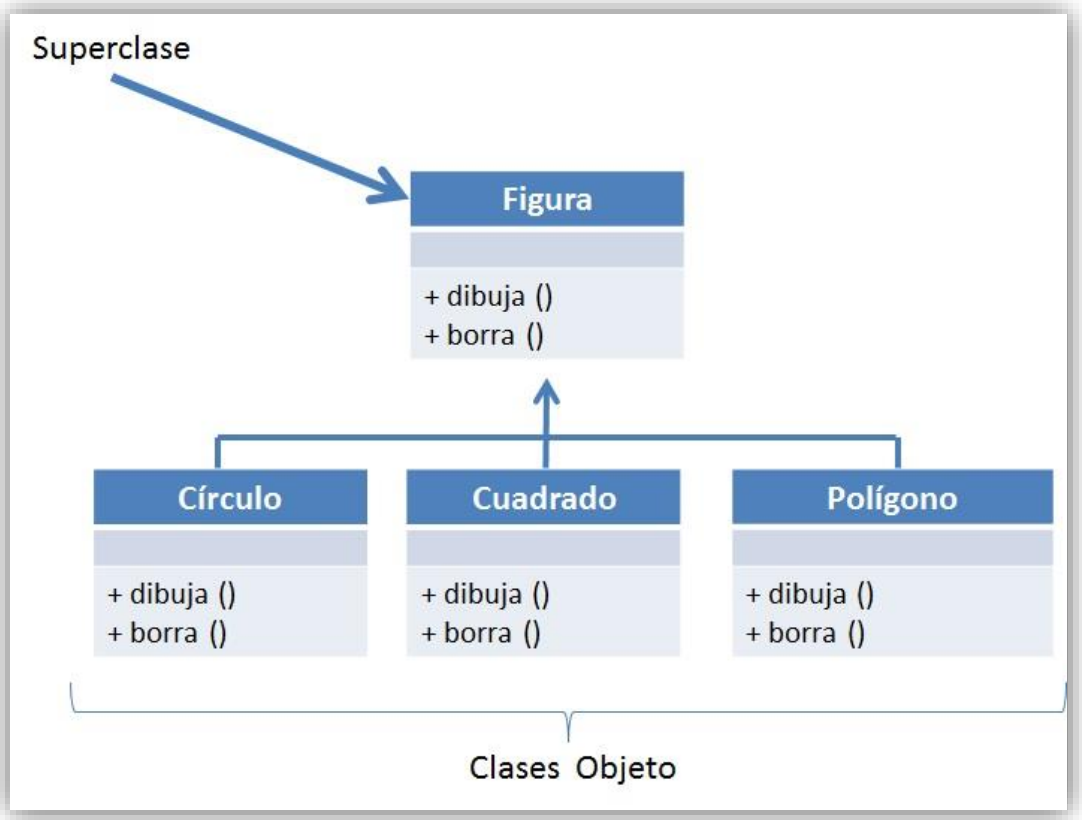


Figura 13. Ejemplo de polimorfismo en Java

COMPARATIVA DE LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS			
Lenguaje	Descripción	Ventajas	Desventajas
C++	• Desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Pues la intención de su creación fue el extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos.	• Lenguaje de programación orientado a objetos. • Permite elaborar aplicaciones sencillas como un "Hello World!" hasta sistemas operativos y mucho más, todo eso dependiendo del manejo del lenguaje. • Actualmente, puede compilar y ejecutar código de C, ya viene con librerías para realizar esta labor.	• Uso de DLLs (librerías dinámicas) muy complejo. Java y .Net han evolucionado estos conceptos manipulando las DLLs mediante los *frameworks* que proveen. En cambio, en C++ el desarrollador debe encargarse de cargar y liberar de memoria estas librerías, y correr los riesgos por el manejo de esta memoria. • Elaborar un sistema en C++ es como construir un rascacielos: tiene buen soporte y es robusto, pero si existen errores en los pisos inferiores toda la parte superior se viene abajo terriblemente. • No es recomendable para desarrollo de páginas Web.

COMPARATIVA DE LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS			
Lenguaje	Descripción	Ventajas	Desventajas
Phyton	• Python es un lenguaje de programación dinámico y orientado a objetos. • El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño. • Python es un lenguaje de programación multiparadigma, permite varios estilos: programación orientada a objetos, programación estructural y funcional	• Simplificado y rápido: Cuenta con una sintaxis elegante y fácil de entender muy parecida al pseudocodigo. • Flexible: Es un lenguaje interpretado en scripts lo cual permite que sea modular y que sea ejecutado en diferentes interpretes para diferentes arquitecturas computacionales. El código python se ejecuta en tiempo real y no es necesario compilarlo de antemano. • Las variables no tienen que ser definidas como de un solo tipo lo que facilita la creación y reduce las palabras necesarias para escribir un programa.	• Conforme se crean aplicaciones más complejas es más complicado escribir el código. • Es problemático distribuir el trabajo en grupos de trabajo como lo es con lenguajes de programación como Java. • Es un lenguaje muy reciente comparado con C dentro del ámbito de la programación profesional, motivo por el que no se encuentra suficiente información relacionada con este lenguajes

COMPARATIVA DE LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS			
Lenguaje	Descripción	Ventajas	Desventajas
Java	• Java es un lenguaje de programación de ordenadores, diseñado como una mejora de C++, y desarrollado por Sun Microsystems • Usa el paradigma de la programación orientada a objetos. • Permite la ejecución de un mismo programa en múltiples sistemas operativos. • Incluye por defecto soporte para trabajo en red. • Es diseñado para ejecutar código en sistemas remotos de forma segura. • Fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.	• Java es un lenguaje multiplataforma con el cual se pueden desarrollar programas que se ejecuten sin problemas en sistemas operativos como Windows, Linux, Mac, Unix, etc. • Es un lenguaje seguro, tiene ciertas políticas que evitan se puedan codificar virus con este lenguaje • Es Multihreaded (multi-hilos), es decir, que puede ejecutar diferentes líneas de código al mismo tiempo.	• Este lenguaje por ser una mejora a un a otro lenguaje no posee muchas desventajas pero resaltando las más importantes tenemos que: • Requiere un intérprete. • Algunas implementaciones y librerías pueden tener código rebuscado. • Una mala implementación de un programa en Java, puede resultar en algo muy lento.

3.1.1 Framework

En general los *frameworks*, en términos generales, son un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar o bien son soluciones completas que contemplan herramientas de apoyo a la construcción (ambiente de trabajo o desarrollo) y motores de ejecución (ambiente de ejecución). Siendo muy simple, es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación.

En el desarrollo de *software*, un *framework* o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos, que puede servir de base para la organización y desarrollo de *software*. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Los *frameworks* tienen como objetivo principal ofrecer una funcionalidad definida, auto-contenida, siendo construidos usando patrones de diseño, y su característica principal es su alta cohesión y bajo acoplamiento. Para acceder a esa funcionalidad, se construyen piezas, objetos, llamados objetos calientes, que vinculan las necesidades del sistema con la funcionalidad que este presta. Esta funcionalidad, está constituida por objetos llamados fríos, que sufren poco o ningún cambio en la vida del *framework*, permitiendo la portabilidad entre distintos sistemas

Dentro del lenguaje Java en el ámbito específico de aplicaciones Web tenemos los *framework*: Struts, "Java Server Faces", Hibernate, o Spring. Estos *frameworks* de Java en la práctica son conjuntos de librerías (API'1s) para desarrollar aplicaciones Web, más librerías para su ejecución (o motor), y más un conjunto de herramientas para facilitar esta tarea (Figura 14).

Figura 14. Frameworks para Java

3.1.1.1 Spring

Spring es un *framework* open source que proporciona un marco de trabajo para el desarrollo de aplicaciones J2EE (Java Enterprise Edition). El *framework* está basado en el uso de textos planos JavaBeans para la lógica de aplicación y archivos XML para la configuración. Por lo regular Spring va acompañado de más *framework* como Hibernate.

Spring está basado en un conjunto de módulos que proporcionan todo lo necesario para desarrollar una aplicación empresarial. Además no es necesario basar la aplicación al completo en el *framework* de Spring, basta con hacer uso del/los módulos que requiera la aplicación e ignorar el resto.

Cuando se diseña una aplicación en Java se puede disponer 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 por los programadores, 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 debe declarar en un archivo de extensión XML los componentes de tu aplicación y sus dependencias. Spring leerá este archivo XML, llamado *Application Context*, y con esto se crean 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.

Módulos

Spring está dividido en alrededor de 20 módulos. Estos a su vez módulos están agrupados en 7 aspectos fundamentales que son: Core Container (Contenedor del coreu), Data Access/Integration (Acceso/Integración de datos) Web, AOP (Aspect Oriented Programming), Instrumentación (Instrumentación) and Test (Pruebas). En la Figura 15 se muestran gráficamente los módulos en que comúnmente se divide Spring.

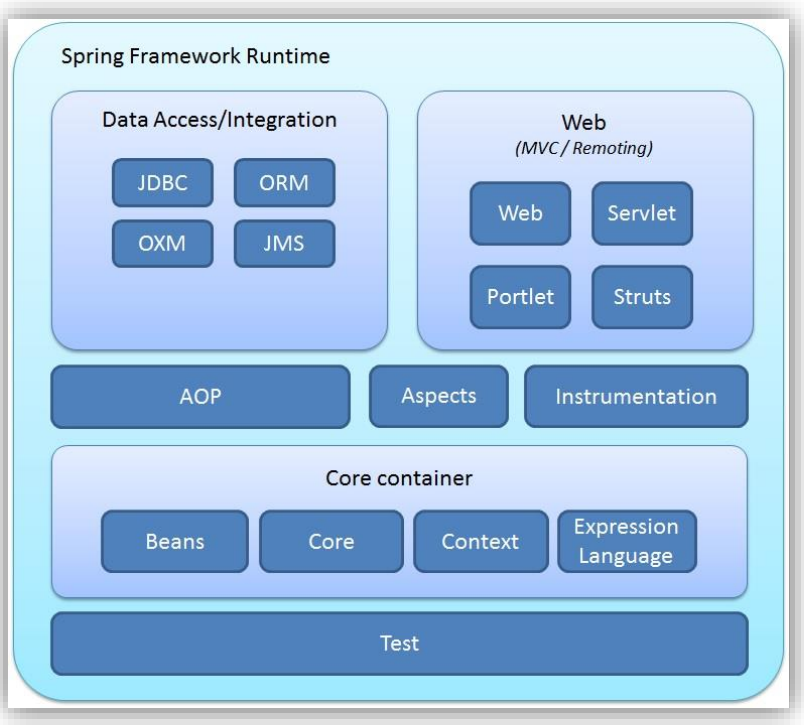


Figura 15. Módulos de Spring

A continuación se describen con un poco más de detalle cada uno de los grupos fundamentales que conforman a Spring.

- *Contenedor del Core:* Cómo su nombre lo dice está compuesto por el núcleo del sistema, esto es, el código que se desarrolla para cubrir con las necesidades propias del *software*. Está compuesto por Core, Beans, Context and Expression Language modules.

    - *Core y Beans:* Estos módulos son la parte fundamental del *Framework*. Incluyen las características de la inversión de control y la inyección de dependencias. BeanFactory es una sofisticada implementación de una fábrica de patrones. Remueve la necesidad para programación única y permite desacoplar la configuración y especificación de dependencias de la lógica del programa actual.
    - *Context:* Es un medio de acceso a los objetos de forma similar a como se hace en un registro de JNDI. Hereda sus características desde el módulo de *Core y Beans* y agrega el soporte para la internacionalización (Utilizando, por ejemplo, paquetes de recursos). El archivo de configuración "Application Context" es punto central de este módulo.
    - *Expression Language:* proporciona un lenguaje de expresión de gran alcance para consultar y manipular un gráfico de objetos en tiempo de ejecución. También es compatible con la lista de proyección y selección, así como la lista de agrupaciones comunes.

- *Acceso/integración de datos:* Esta capa consiste en las librerías necesarias para la conexión y obtención de datos, en una base de datos. Estas librerías son JDBC, ORM, OXM, JMS y módulos de transacción.

    - El *JDBC* proporciona una JDBC-abstracción que remueve la necesidad de hacer el tedioso JDBC codificando y parseando los códigos de error especifico del proveedor de base de datos.
    - El *ORM* proporciona capas de integración para las API's de mapeo objeto-relacional populares, incluyendo JPA, JDO, Hibernate y iBatis. Usando el paquete ORM puede utilizar todos estos marcos de O / R-mapping en combinación con todas las otras características que Spring ofrece.
    - El OXM proporciona una capa de abstracciónque da soporte a los objetos/XML mapeando la implementación para JAXB, Castorm, XMLBeans, JiBX y XStream.
    - El módulo JMS (Java Menssaging Service) contiene las características para producir y consumir mensajes.
    - El módulo de transacción soporta la gestión de transacciones programáticas y declarativas para clases que implementan interfaces especiales y para todos sus POJO (Plain Old Java Objects).

- *Web:* La capa Web consta de estos módulos Web, Web-Servlet, Web-Struts y Web –Portlet

    - El módulo de *Spring Web* ofrece funciones de integración web-oriented básicos como la funcionalidad de carga de archivos en diferentes partes y la inicialización del contenedor usando oyentes servlet y un contexto de aplicación orientada a la web. También contiene las partes relacionados con la web de soporte de soporte remoto de Spring.
    - El módulo Web-Servlet contiene Spring modelo-vista-controlador (MVC) la implementación de aplicaciones web. *Framework* Spring MVC proporciona una separación limpia entre el código del modelo de dominio y formularios web, y se integra con todas las otras características del *framework* de Spring.
    - El módulo Web-Struts contiene las clases de apoyo para la integración de una capa web Struts clásicos dentro de una aplicación de Spring.
    - El módulo Web-Portlet proporciona la implementación MVC para ser utilizado en un entorno de portlet y refleja la funcionalidad del módulo de Web-Servlet.

- *AOP e Instrumentación:* El módulo Spring AOP proporciona una implementación Alliance-compliant aspect-oriented que le permite definir, por ejemplo, métodos interceptores y puntos de corte para desacoplar limpiamente el código que implementa la funcionalidad que deben separarse. Utilizando la funcionalidad de metadatos a nivel de fuente, también puede incorporar información de comportamiento en su código, de una manera similar a la de los atributos de .NET.

  El módulo de Aspectos es independiente y proporciona integración con AspectJ.

  El módulo de Instrumentación proporciona soporte clases de instrumentación y las implementaciones de classloader que se utilizan en ciertos servidores de aplicaciones.

- *Test:* Este módulo apoya la prueba de los componentes de Spring con JUnit o TestNG. Se proporciona una carga constante de Spring ApplicationContexts y almacenamiento en caché de los contextos. También proporciona objetos de imitación que puede utilizar para probar su código de manera aislada.

3.1.1.2 JSF (Java Server Faces)

JSF (Java Server Faces) es una tecnología y *framework* para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (acrónimo de XML-based User-interface Language, lenguaje basado en XML para la interfaz de usuario). En la Figura 16 se muestra un flujo de JSF.

JSF incluye:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto por defecto de componentes para la interfaz de usuario.
- Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP.
- Un modelo de eventos en el lado del servidor.
- Administración de estados.
- Beans administrados.

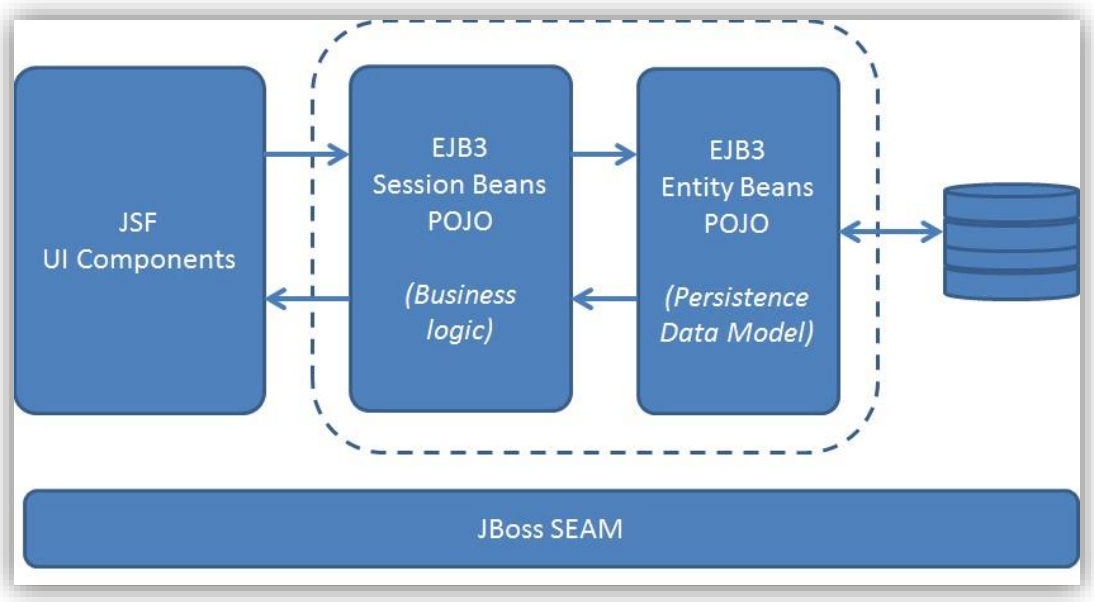


Figura 16. Uso de JSF

3.1.1.3 Hibernate

Hibernate es un ORM (Mapeo objeto-relacional) que se encarga facilitar y optimizar las operaciones contra la base de datos. Un ORM nos permite tratar las tablas de nuestra base de datos como si fuesen objetos, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Hibernate es *software* libre, distribuido bajo los términos de la licencia GNU LGPL.

Hibernate busca solucionar el problema de la diferencia entre los dos modelos usados hoy en día para organizar y manipular datos: El usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

Hibernate ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria").

Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente Hibernate Annotations que implementa el estándar JPA, que es parte de esta plataforma. La figura 17 nos muestra un ejemplo de la estructura de Hibernate.

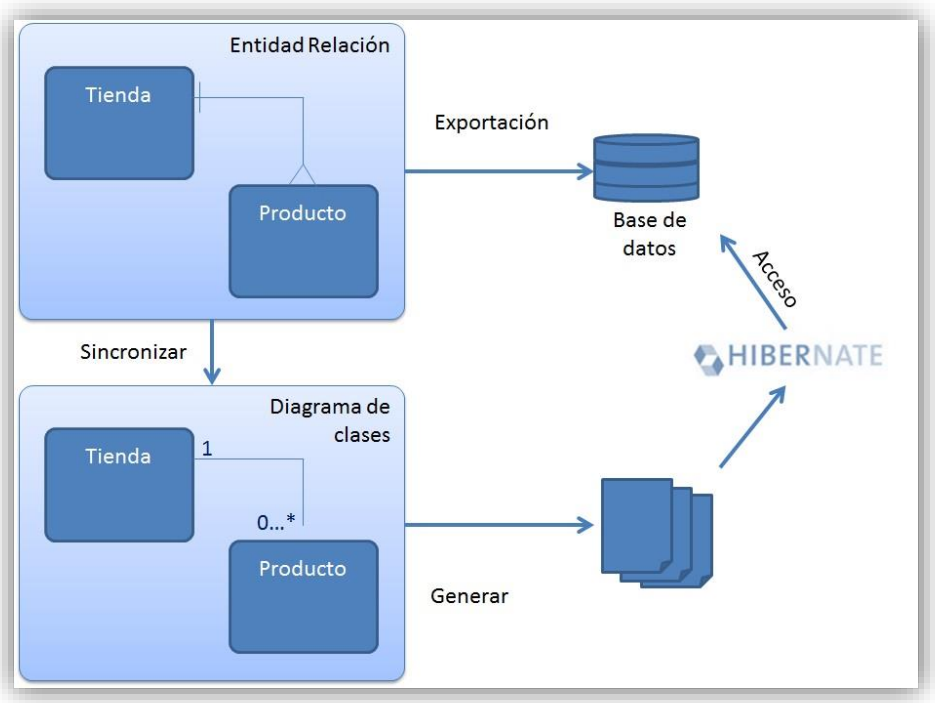


Figura 17. Ejemplo de Hibernate

3.1.1.4 Struts

Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma Java EE (Java Enterprise Edition).

Struts se basa en el patrón de arquitectura de *software* Modelo-Vista-Controlador (MVC) el cual se utiliza ampliamente y es considerado de gran solidez. De acuerdo con este *Framework*, el procesamiento se separa en tres secciones diferenciadas llamadas el modelo, las vistas y el controlador.

Evidentemente, como todo *framework* intenta, simplifica notablemente la implementación de una arquitectura según el patrón MVC. El mismo separa muy bien lo que es la gestión del workflow de la aplicación, del modelo de objetos de negocio y de la generación de interfaz.

El controlador ya se encuentra implementado por Struts, aunque si fuera necesario se puede heredar y ampliar o modificar, y el workflow de la aplicación se puede programar desde un archivo XML Las acciones que se ejecutarán sobre el modelo de objetos de negocio se implementan basándose en clases predefinidas por el *framework* y siguiendo el patrón Facade. Y la generación de interfaz se soporta mediante un conjunto de Tags predefinidos por Struts cuyo objetivo es evitar el uso de Scriplets (los trozos de código Java entre "<%" y "%>"), lo cual genera ventajas de mantenibilidad y de perfomance (pooling de Tags, caching, etc).

Logísticamente, separa claramente el desarrollo de interfaz del workflow y lógica de negocio permitiendo desarrollar ambas en paralelo o con personal especializado.

También es evidente que potencia la reutilización, soporte de múltiples interfaces de usuario (Html, sHtml, Wml, Desktop applications, etc.) y de múltiples idiomas, localismos, etc.

Funciona de la Siguiente manera, el navegador genera una solicitud que es atendida por el Controller (un Servlet especializado). El mismo se encarga de analizar la solicitud, seguir la configuración que se le ha programado en su XML y llamar al Action correspondiente pasándole los parámetros enviados. El Action instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que retorne el Action, el Controller derivará la generación de interfaz a una o más JSPs, las cuales podrán consultar los objetos del Model a fines de realizar su tarea. (Figura 18)

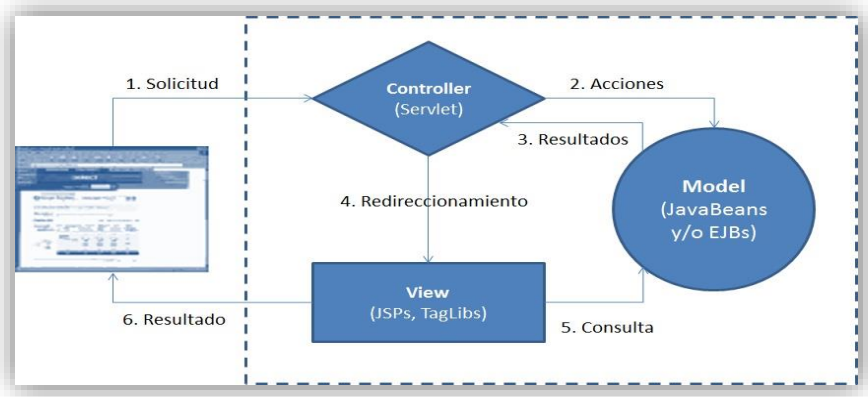


Figura 18. Ciclo de Struts

El uso de *frameworks* es muy común en la vida laboral, actualmente para desarrollos en Java es muy usado Spring en combinación con Hibernate, está combinación realmente facilita las cosas para el desarrollador pues es más fácil el uso de un objeto Java que un dato como se obtiene de la base de datos. El *framework* más utilizado es Spring, es por la facilidad que tiene para acoplarse con los demás *frameworks* y por el hecho de poder utilizar beans para tener referencias a proyectos asociados.

En lo personal, creo que los *frameworks* son una buen herramienta de trabajo, pero también se debe conocer el camino que se debe de seguir para hacer el desarrollo sin ese tipo de herramientas, pues aún muchos *software* no han sido actualizados para aceptar ese tipo de tecnología.

Durante mi experiencia profesional he usado los *frameworks* como Spring para hacer referencia a diferentes proyectos que se necesita ocupar para su ejecución esto con el fin de generar objetos Java para facilidad en su manejo, en combinación con algunos otros suele ser muy cómodo usarlo, por ejemplo si es usado con Hibernate se puede manejar con mucha mayor facilidad cada una de las tablas que se tienen en la base de datos, pues creo que es más fácil para el desarrollador ocupar objeto, en mi caso Java, que objetos o cadenas no conocidas.

Con referencia a Struts, ya es una arquitectura muy poco usada, pero es muy parecida Spring con la diferencia de

3.1.2 Ext JS

Ext JS (pronunciado como "ekst") es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM.

Esta librería incluye:

- Componentes UI del alto performance y personalizables.
- Modelo de componentes extensibles.
- Un API fácil de usar.
- Licencias Open source y comerciales.
- Este *framework* cuenta con un conjunto de componentes (widgets) para incluir dentro de una aplicación web, como:
- Cuadros y áreas de texto.
- Campos para fechas.
- Campos numéricos.
- Combos.
- Radiobuttons y checkboxs.
- Editor HTML.
- Elementos de datos (con modos de sólo lectura, datos ordenables, columnas que se pueden bloquear y arrastrar, etc.).
- Árbol de datos.
- Pestañas.
- Barra de herramientas.
- Menús al estilo de Windows.
- Paneles divisibles en secciones.
- Sliders.

Varios de estos componentes están dotados de comunicación con el servidor usando AJAX. También contiene numerosas funcionalidades que permiten añadir interactividad a las páginas HTML, como:

- Cuadros de diálogo.
- *quicktips* para mostrar mensajes de validación e información sobre campos individuales.

3.1.3 Web Services

El término Web Services describe una forma estandarizada de integrar aplicaciones WEB mediante el uso de XML, SOAP, WSDL y UDDI sobre los protocolos de la Internet. XML es usado para describir los datos, SOAP se ocupa para la transferencia de los datos, WSDL se emplea para describir los servicios disponibles y UDDI se ocupa para conocer cuáles son los servicios disponibles. Uno de los usos principales es permitir la comunicación entre las empresas y entre las empresas y sus clientes. Los Web Services permiten a las organizaciones intercambiar datos sin necesidad de conocer los detalles de sus respectivos Sistemas de Información.

Los Web Services permiten a distintas aplicaciones, de diferentes orígenes, comunicarse entre ellos sin necesidad de escribir programas costosos, esto porque la comunicación se hace con XML. Los Web Services no están ligados a ningún Sistema Operativo o Lenguaje de Programación. Por ejemplo, un programa escrito en Java puede conversar con otro escrito en Pearl; Aplicaciones Windows puede conversar con aplicaciones Unix. Por otra parte los Web Services no necesitan usar browsers (Explorer) ni el lenguaje de especificación HTML.

Los Web Services están construidos con varias tecnologías que trabajan conjuntamente con los estándares que están emergiendo para asegurar la seguridad y operatibilidad, de modo de hacer realidad que el uso combinado de varios Web Services, independiente de la o las empresas que los proveen, este garantizado. A continuación se describen brevemente los estándares que están ocupando los Web Services.

- ✓ XML

  Abreviación de Extensible Markup Language. El XML es una especificación desarrollada por W3C. Permite a los desarrolladores crear sus propios tags, que les permiten habilitar definiciones, transmiciones, validaciones, e interpretación de los datos entre aplicaciones y entre organizaciones.

- ✓ SOAP

  Abreviación de Simple Object Access Protocol, es un protocolo de mensajería construido en XML que se usa para codificar información de los requerimientos de los Web Services y para responder los mensajes antes de enviarlos por la red. Los mensajes SOAP son independientes de los sistemas operativos y pueden ser transportados por los protocolos que funcionan en la Internet, como ser: SMTP, MIME y HTTP.

- ✓ WSDL

  Abreviación de Web Services Description Language, es un lenguaje especificado en XML que se ocupa para definir los Web Service como colecciones de punto de comunicación capaces de intercambiar mensajes. El WSDL es parte integral de UDDI y parte del registro global de XML, en otras palabras es un estándar de uso público (no se requiere pagar licencias ni royalties para usarlo).

- ✓ UDDI

  Abreviación de Universal Description, Discovery and Integration. Es un directorio distribuido que opera en la Web que permite a las empresas publicar sus Web Services, para que otras empresas conozcan y utilicen los Web Services que publican, opera de manera análoga a las páginas amarillas.

Los Web Services son comúnmente usados para empresas con diferentes sistemas, por ejemplo, en una institución financiera para la que trabaje donde realice la aplicación móvil de la banca, era necesario usar servicios provenientes del sistema de Banca Electrónica esto porque de no ser usado se debería rehacer el código para la aplicación móvil y de haber cambios en la versión electrónica de la banca se tendría que modificar el código, con los Web Services fue más fácil la compatibilidad de los sistemas de ser necesaria alguna modificación se hará a través de cada sistema y que para el sistema cliente será transparente.

Otros sistemas que compartían datos mediante Web Services eran el Sistema de Casa de Bolsa y el Sistema de Inversiones, de otra institución financiera, en este caso era más necesario pues los precios de las emisoras se tenían que conocer al momento de operar, en línea, pues pueden cambiar según el día y la hora. Las modificaciones que se llegarán a necesitar en el Sistema de Casa de Bolsa pasaban de forma transparente para el Sistema de Inversiones y viceversa.

En mi opinión son útiles para comunicar dos sistemas de la misma empresa, sobre todo si los sistemas necesitan información en línea o están en constante modificación por cambio de regulaciones o cambio de necesidades. Con este tipo de servicios no expones el sistema servidor porque de no responder el Web Service lanzará una excepción que sin mayor problema se puede solucionar del lado del cliente.

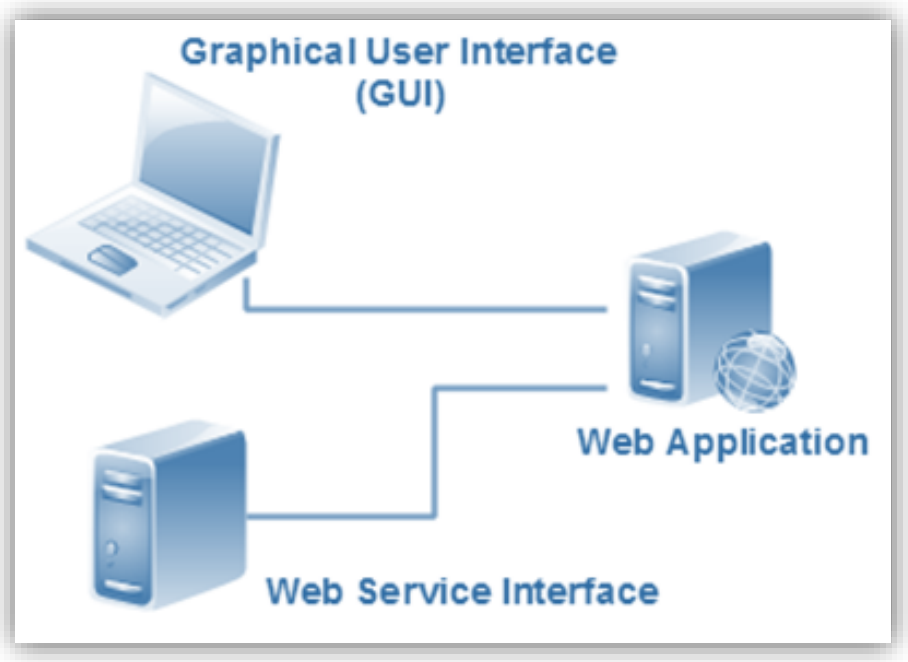


Figura 19. Diagrama de funcionamiento de un Web Service

4. Diseño de bases de datos

4.1 Introducción a las bases de datos

Una base de datos es una colección de información organizada de forma que un programa de ordenador pueda seleccionar rápidamente los fragmentos de datos que necesite. Una base de datos es un sistema de archivos electrónico.

Se le llama base de datos a los bancos de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto.

Una Base de datos puede ser de diverso tipo, desde un pequeño fichero casero para ordenar libros y revistas por clasificación alfabética hasta una compleja base que contenga datos de índole gubernamental en un Estado u organismo internacional. Recientemente, el término base de datos comenzó a utilizarse casi exclusivamente en referencia a bases construidas a partir de *software* informático, que permiten una más fácil y rápida organización de los datos. Las bases de datos informáticas pueden crearse a partir de *software* o incluso de forma online usando Internet. En cualquier caso, las funcionalidades disponibles son prácticamente ilimitadas.

Los usos de las bases de datos son tan múltiples que, por ejemplo, pueden utilizarse en una biblioteca o archivo que guarda libros para su consulta frecuente, pero también pueden emplearse para guardar material biológico, como un archivo genético o un banco de esperma que almacena esperma para ser utilizado en fertilización.

Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro.

Características

Entre las principales características de los sistemas de base de datos podemos mencionar:

- *Independencia lógica y física de los datos:*
    - *Independencia física de datos:* Es la capacidad de modificar el esquema físico sin provocar que se vuelvan a escribir los programas de aplicación. Las modificaciones en el nivel físico son ocasionalmente necesarias para mejorar el funcionamiento.

    - *Independencia lógica de datos:* Capacidad de modificar el esquema conceptual sin provocar que se vuelvan a escribir los programas de aplicación. Las modificaciones en el nivel lógico son necesarias siempre que la estructura lógica de la base de datos se altere.

- *Redundancia mínima:* hace referencia al almacenamiento de los mismos datos varias veces en diferentes lugares. La redundancia de datos puede provocar problemas como:

    - *Incremento del trabajo:* como un mismo dato está almacenado en dos o más lugares, esto hace que cuando se graben o actualicen los datos, deban hacerse en todos los lugares a la vez.

    - *Desperdicio de espacio de almacenamiento:* ya que los mismos datos están almacenados en varios lugares distintos, ocupando así más bytes del medio de almacenamiento. Este problema es más evidente en grandes bases de datos.

    - *Inconsistencia de datos:* esto sucede cuando los datos redundantes no son iguales entre sí. Esto puede suceder, por ejemplo, cuando se actualiza el dato en un lugar, pero el dato duplicado en otro lugar no es actualizado.

Si una base de datos está bien diseñada, no debería haber redundancia de datos (exceptuando la redundancia de datos controlada, que se emplea para mejorar el rendimiento en las consultas a las bases de datos).

- *Integridad de los datos:* Se define como estado de corrección y completitud de los datos ingresados en una base de datos.

  Los SGBD relacional deben encargarse de mantener la integridad de los datos almacenados en una base de datos con respecto a las reglas predefinidas o restricciones. La integridad también puede verificarse inmediatamente antes del momento de introducir los datos a la base de datos (por ejemplo, en un formulario empleando validación de datos). Un claro ejemplo de error de integridad es el ingreso de un tipo de dato incorrecto dentro de un campo. Por ejemplo, ingresar un texto cuando se espera un número entero.

- *Acceso concurrente por parte de múltiples usuarios:* Un sistema que permita a varias estaciones de trabajo modificar en forma simultánea una misma base de datos, debe tomar precauciones para evitar operaciones concurrentes sobre un mismo registro. Esto es, si un usuario de una estación de trabajo solicita el registro 3 para ser modificado, el sistema debe advertir a otro usuario que solicite el mismo registro 3, que está siendo actualizado por otra estación de trabajo.

- *Consultas complejas optimizadas:* La optimización de consultas permite la rápida ejecución de la misma, por ello se debe de verificar que no sean tan grandes los códigos de búsqueda para hacer una consulta debemos introducir datos CLAROS y PRECISOS.

- *Seguridad de acceso y auditoría:* Se refiere al derecho de acceder a los datos contenidos en la B.D. por parte de personas y organismos. El sistema de auditoria mantiene el control de acceso a la B.D, con el objeto de saber que o quien realizo una determinada modificación y en qué momento.

- *Respaldo y recuperación:* Esto quiere decir la capacidad que tiene un sistema de base de datos de recuperar su estado en un momento previo a la pérdida de datos. Siempre se debe de contar con un respaldo, en el cual, si le llegara a pasar algún daño o tener perdida de información, con tu respaldo tienes con que ampararte.

- *Acceso a través de lenguajes de programación estándar: E*sto quiere decir, la posibilidad que se tiene de acceder a la información o datos de una B.D. por medio de un lenguaje de programación que se encuentre ajeno al sistema de la base de datos ya sea con sus altas, bajas y cambios.

4.2 Sistema de Gestión de base de datos (SGBD)

Es el *software* que permite la utilización y/o la actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista y a la vez, se denomina sistema de gestión de bases de datos (SGBD).

El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manipular, en términos abstractos, los datos, o sea, de forma que no le sea necesario conocer el modo de almacenamiento de los datos en la computadora, ni el método de acceso empleado.

Los programas de aplicación operan sobre los datos almacenados en la base utilizando las facilidades que brindan los SGBD, los que, en la mayoría de los casos, poseen lenguajes especiales de manipulación de la información que facilitan el trabajo de los usuarios.

Los SGDB brindan facilidad a la hora de elaborar tablas y establecer relaciones entre las informaciones contenidas en ellas. Pueden mantener la integridad de una base de datos permitiéndole a más de un usuario actualizar un registro al mismo tiempo y también puede impedir registros duplicados en una BD.

En general las principales características de los SGBD son:

- Permite crear y gestionar base de datos de forma fácil, cómoda y rápida.
- Ofrece una gran flexibilidad para el trabajo con base de datos relacionales.
- Ofrece un ambiente agradable dado por su interfaz gráfica.

Las bases de datos han estado en uso desde los primeros días de las computadoras electrónicas. A diferencia de los sistemas modernos, que se pueden aplicar a datos y necesidades muy diferentes, la mayor parte de los sistemas originales estaban enfocados a bases de datos específicas y pensadas para ganar velocidad a costa de perder flexibilidad. Los SGBD originales sólo estaban a disposición de las grandes organizaciones que podían disponer de las complejas computadoras necesarias.

4.3 Modelo Entidad-Relación

El modelo de datos entidad-relación está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre esos objetos. El modelado de datos no acaba con el uso de esta técnica. Son necesarias otras técnicas para lograr un modelo que se pueda implementar directamente en una base de datos.

Permite mostrar resultados entre otras entidades pertenecientes a las existentes de manera que se encuentre la normatividad de archivos que se almacenaran

- Transformación de relaciones múltiples en binarias, cambio al modelo relacional.
- Normalización de una base de datos de relaciones (algunas relaciones pueden transformarse en atributos y viceversa).
- Conversión en tablas (en caso de utilizar una base de datos relacional).

A continuación describiré brevemente los objetos que conforman el modelo Entidad-Relación

- ✓ *Entidad:* Representa una “cosa” u "objeto" del mundo real con existencia independiente, es decir, se diferencia únicamente de otro objeto o cosa, incluso siendo del mismo tipo, o una misma entidad.
  Una entidad puede ser un objeto con existencia física como: una persona, un animal, una casa, etc. (entidad concreta); o un objeto con existencia conceptual como: un puesto de trabajo, una asignatura de clases, un nombre, etc. (entidad abstracta).

  Una entidad está descrita y se representa por sus características o atributos. Por ejemplo, la entidad Persona las características: Nombre, Apellido, Género, Estatura, Peso, Fecha de nacimiento.

- ✓ *Atributos:* Los atributos son las características o propiedades asociadas a la entidad a una entidad. Estas pueden ser muchas, y el diseñador solo utiliza o implementa las que considere más relevantes.

  En un conjunto de entidades del mismo tipo, cada entidad tiene valores específicos asignados para cada uno de sus atributos, de esta forma, es posible su identificación unívoca.

- ✓ *Relación:* Se entiende por relación a la asociación entre 2 o más entidades, se pueden clasificar en 2 tipos:

  - *Clasificación por Cardinalidad:*

    - ✓ *Relación uno a uno:* Cuando un registro de una tabla sólo puede estar relacionado con un único registro de la otra tabla y viceversa. En este caso la clave foránea se ubica en alguna de las 2 tablas.

- ✓ *Relación uno a muchos o muchos a uno:* Cuando un registro de una tabla (tabla secundaria) sólo puede estar relacionado con un único registro de la otra tabla principal puede tener más de un registro relacionado en la tabla secundaria. En este caso la clave foránea se ubica en tabla secundaria.

- ✓ *Relación mucho a muchos:* Cuando un registro de una tabla puede estar relacionado con más de un registro de la otra tabla y viceversa. En este caso las dos tablas no pueden estar relacionadas directamente, se tiene que añadir una tabla entre las dos (tabla débil o de vinculación) que incluya los pares de valores relacionados entre sí.

  El nombre de la tabla débil devine que con sus atributos propios no se puede encontrar la clave, por estar asociada a otra entidad. La clave de esta tabla se conforma por la unión de los campos claves de las tablas que relaciona.

Cardinalidad	Relación entre entidades	Llave foránea (FK)
Uno a uno	No es necesario crear una relación entre las entidades	Se crea una llave foránea en una de las entidades relacionadas que corresponde a la llave primaria en la otra
Uno a muchos	No es necesario crear una relación entre las entidades	Se crea una llave foránea del lado de la entidad "muchos" que corresponde a la llave primaria del lado "uno"
Muchos a muchos	Se crea una relación con llave primaria compuesta formada por las llaves primarias de las entidades que une la relación	En las entidades no hay llave foránea

Figura 20. Resumen de relaciones por cardinalidad.

- *Clasificación por modalidad:* Dadas las tablas A y B, se encuentran relacionadas:

  Si para to registro de A debe existir siempre al menos un registro de B asociado, se dice que la relación en sentido A –> B es **obligatoria**.
  Si para todo registro de A, pueden existir o no, uno o varios registros de B asociados, se dice que la relación en sentido A -> B es **optativa**.

✓ *Claves:* Es un subconjunto del conjunto de atributos comunes en una colección de entidades, que permite identificar inequívocamente cada una de las entidades pertenecientes a dicha colección. Asimismo, permiten distinguir entre sí las relaciones de un conjunto de relaciones.

Dentro de los conjuntos de entidades existen los siguientes tipos de claves:

- *Clave primaria o principal:* Es el atributo o conjunto minino de atributos (uno o más campos) que permiten identificar en forma única instancia de la entidad, es decir, a cada registro de la tabla. Las claves principales se utilizan cuando se necesita hacer referencia a registros específicos de una tabla desde otra.

- *Clave candidata:* Son atributos que cumplen con las condiciones para ser clave.

- *Clave simple:* Si la clave primaria se determina mediante un solo atributo de la entidad, entonces se dice que la misma se dice que la misma es una clave simple.

- *Clave compuesta:* En caso de estar conformada por más de un atributo, la misma se conoce como clave compuesta.

- *Clave foránea:* Es un atributo que es clave primaria en otra entidad con la cual se relaciona.

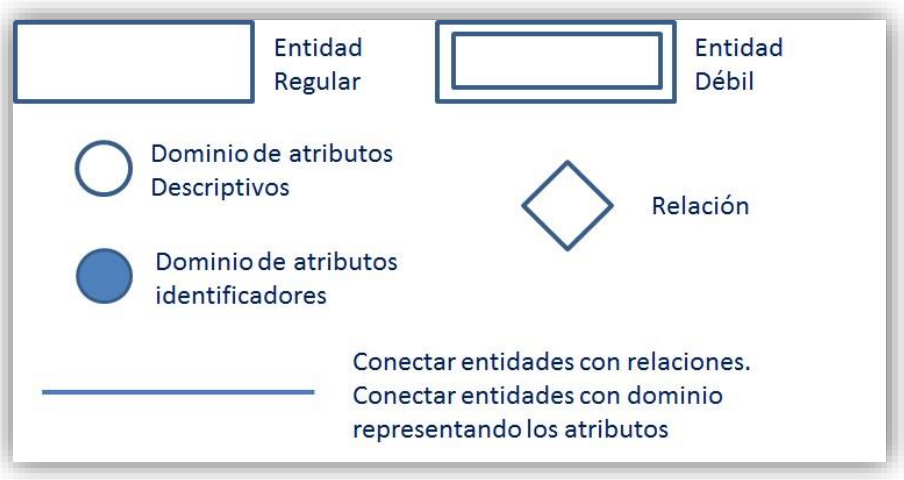


Figura 21. Simbología básica del Modelo entidad-relación

El modelo entidad-relación es importante como primer paso para el diseño de una base de datos, pues podemos dividir los datos en peños objetos de fácil manejo y mejor compresión, pues se visualizan, mejor, los atributos que cada tabla debe tener.

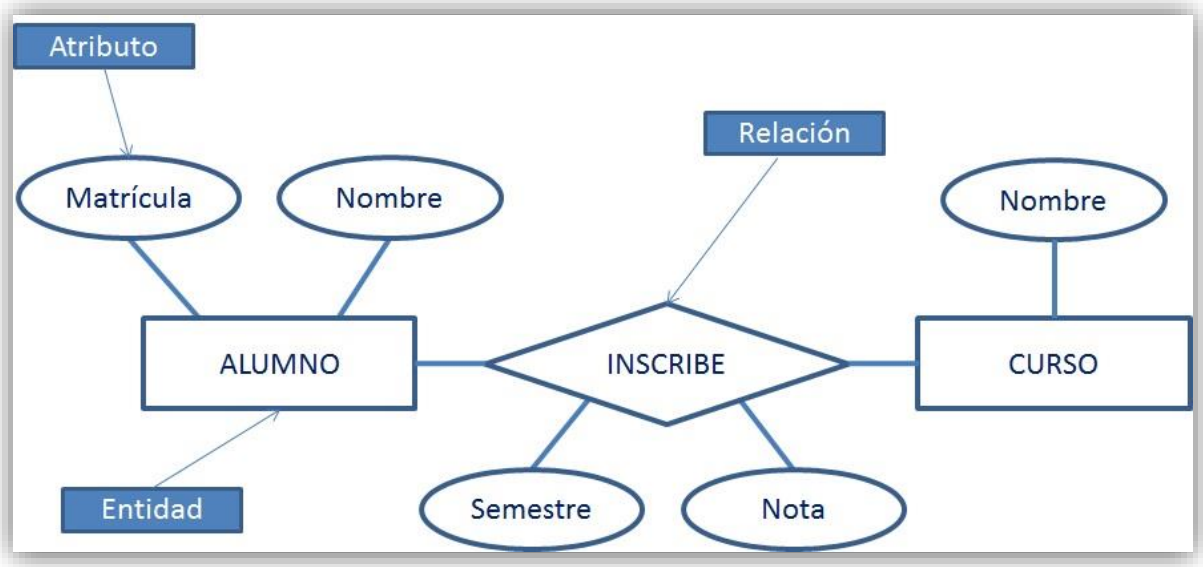


Figura 22. Ejemplo de modelo entidad-relación

4.4 Modelo Relacional

Puede resultar confuso el concepto de modelo entidad-relación vs modelo relacional, quizás porque ambos comparten casi las mismas palabras, el objetivo del modelo relacional es crear un "esquema" (schema).

La idea fundamental, de Cood, es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. Pese a que esta es la teoría de las bases de datos relacionales creadas por Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, pensando en cada relación como si fuese una tabla que está compuesta por registros (cada fila de la tabla sería un registro o "tupla") y columnas (también llamadas "campos").

Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente.

En el modelo relacional se basa en el concepto matemático de relación. En este modelo, la información se representa en forma de "tablas" o relaciones, donde cada fila de la tabla se interpreta como una relación ordenada de valores (un conjunto de valores relacionados entre sí). El siguiente ejemplo presenta una relación que representa al conjunto de los departamentos de una determinada empresa, y que recoge información sobre los mismos.

Definiciones:

Formalmente, una *relación* se define como un conjunto de n-tuplas; donde una *n-tupla* se define a su vez como un conjunto ordenado de valores atómicos (esto es, no divisibles ni descomponibles en valores más "pequeños". En el ejemplo de la figura 20, la relación mostrada incluye dos 3-tuplas: ('D-01', 'Ventas', 'A Coruña') y ('D-02', 'I+D', 'Ferrol'). Cada tupla incluye información sobre los departamentos de una determinada empresa con sede en Galicia: el identificador del departamento dentro de la empresa, su nombre, y la localidad donde tiene su sede. En cada tupla, los tres valores están relacionados por el hecho de describir todos ellos al mismo departamento.

Num	Nombre	Localidad
D-01	Ventas	A Coruña
D-02	I+D	Ferrol

Figura 23. Ejemplo de relación

Cada relación, vista como una tabla, consta de un conjunto de columnas; cada una de esas columnas recibe el nombre de *atributo*. A cada atributo de una relación le corresponde un *nombre*, que debe ser único dentro de la relación, y un *dominio*: el conjunto de valores válidos para un atributo; o, dicho de otra manera, el conjunto de valores que cada tupla de la relación puede tomar para ese atributo.

El *esquema* de una relación es una descripción de su estructura interna (es decir, los atributos que la componen), en la forma siguiente:

$$R\ (A_1, \ldots, A_n)$$

Siendo R el nombre de la relación, y A1, ... , An los nombres de sus n atributos. A partir del esquema de la relación es posible determinar su grado: el número de atributos de los que consta.

Restricciones de integridad

Cada tupla de una relación debe proporcionar valores a sus atributos. ¿De cualquier manera? No. Para garantizar la consistencia y la facilidad de manipulación de la información representada, existen una serie de reglas que deben ser cumplidas y que son un elemento constituyente del modelo relacional. A esas reglas de consistencia se las conoce, en la terminología del modelo, como restricciones de integridad. Podemos distinguir varios tipos de restricciones:

- ✓ Restricción de DOMINIO: Exige que lo valores de cualquier tupla de una relación R correspondientes a los atributos A_1, …, A_n de R deben ser valores atómicos. Esto es, esos valores no se pueden descomponer en valores más pequeños o simples. Esta condición pretende garantizar que todas las relaciones presenten un formato "estándar", que pueda ser fácilmente manipulable por medio de un sencillo procedimiento o algoritmo, implementado en forma de un programa informático.

- ✓ Restricción de clave: Ningún conjunto admite, por definición la existencia de elementos repetidos en su contenido. Tratándose de un conjunto de tuplas, las relaciones requieren la misma exigencia. Que la extensión de una relación ni incluya tuplas repetidas, implica que todas las tuplas que contiene puedan ser diferenciadas entre sí por el valor de al menos un atributo, como se lo definimos en el modelo anterior la llave primaria.

- ✓ Restricción de Integridad de entidad: Dado que es la clave primaria la que nos permite distinguir a las tuplas entre sí, los valores correspondientes a las clave deben ser conocidos en cada tupla para poder diferenciarla.

- ✓ Restricciones de integridad referencial: Este tipo de restricciones permite garantizar la consistencia en el caso de relaciones que mantengan una cierta vinculación.

Para pasar del modelo entidad relación simplemente debemos aplicar el siguiente cuadro:

CAMBIO ENTRE EL MODELO ENTIDAD-RELACIÓN Y EL MODELO RELACIONAL		
Modelo Entidad-Relación	Modelo relacional	Observaciones
Entidad	Tabla	
Atributo	Columna/Campo	
Identificador único	Clave primaria	
Relaciones M:M (muchos a muchos)	Nueva tabla con clave primaria la concatenación de las claves de las entidades de la tabla que la forman (la relación pasa a ser una tabla, y en esa tabla se pone como C.A. las entidades que une).	
Relaciones 1:M (uno a muchos o muchos a uno)	Transformar la relación en una tabla si no todos los elementos de la entidad que participa con muchos tienen asociado un elemento de la entidad que participa con uno. Otra opción es propagar la llave primaria de 1 en la de muchos, es decir, creamos un campo en la tabla de muchos que haga referencia a la tabla de 1, sólo si cada elemento de la entidad que participa con muchos aparece en la entidad de uno, es decir, si todos los elementos de la entidad de muchos tienen asociado uno de la entidad de uno.	Esta diferencia se debe a que todas las clases ajenas deben hacer referencia a la clave primaria de otras tablas y consecuentemente no pueden ser anuladas. Dicho de otra manera, toda referencia ajena debe hacerse en un campo único.
Relaciones 1:1 (uno a uno)	Transformar la relación en tabla si no todos los elementos de la entidad que participa con muchos tienen asociado un elemento de la entidad que participa con uno. Otra opción es propagar la clave (igual que en la de 1:M) si cada elemento de la entidad que participa con muchos aparece en la entidad de uno, es decir, si TODOS los elementos de la entidad de muchos tienen asociado uno de la entidad uno.	

4.5 Normalización

El proceso de normalización de bases de datos consiste en designar y aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional.

Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

En el modelo relacional es frecuente llamar tabla a una relación, aunque para que una tabla sea considerada como una relación tiene que cumplir con algunas restricciones:

- Cada tabla debe tener su nombre único.
- No puede haber dos filas iguales. No se permiten los duplicados.
- Todos los datos en una columna deben ser del mismo tipo.

4.5.1 Formas Normales:

Las formas normales son aplicadas a las tablas de una base de datos. Decir que una base de datos está en la forma normal N es decir que todas sus tablas están en la forma normal N.

Primera forma normal: No elementos repetidos o grupos de elementos, se puede satisfacer la necesidad de la atomicidad de la PFN simplemente: separando cada ítem de estas listas en su propia fila. Por otra parte la PFN también aborda el tema de la llave primaria, esto es que cada fila debe tener un único identificador.

Segunda forma normal: Sin dependencias parciales en llaves concatenadas. Esto significa que para una tabla que tiene una llave primaria concatenada, cada columna de la tabla que no forma parte de la llave primaria. Tiene que depender de la llave concatenada completamente. Si hay alguna columna que solamente dependa de una parte de la llave concatenada, entonces decimos que la tabla completa no cumple con SFN y se tiene que crear otra tabla para rectificar el fallo.

Tercera forma normal: Sin dependencia de atributos que no son llaves, es decir que no existe ninguna dependencia funcional transitiva entre los atributos que no son clave y además se encuentra en SFN.

4.5.2 Reglas de Codd

Hay tablas las cuales dicen ser relacionales, pero lo único que hacen es guardar la información en las tablas, sin estar estas tablas literalmente normalizadas; por lo que Codd decidió publicar estas 12 reglas que un verdadero sistema relacional debería tener, en la práctica algunas de ellas son difíciles de realizar.

Regla No. 1 - La Regla de la información

Toda la información en un RDBMS está explícitamente representada de una sola manera por valores en una tabla.

Cualquier cosa que no exista en una tabla no existe del todo. Toda la información, incluyendo nombres de tablas, nombres de vistas, nombres de columnas, y los datos de las columnas deben estar almacenados en tablas dentro de las bases de datos. Las tablas que contienen tal información constituyen el Diccionario de Datos. Esto significa que todo tiene que estar almacenado en las tablas.

Toda la información en una base de datos relacional se representa explícitamente en el nivel lógico exactamente de una manera: con valores en tablas. Por tanto los metadatos (diccionario, catálogo) se representan exactamente igual que los datos de usuario. Y puede usarse el mismo lenguaje (ej. SQL) para acceder a los datos y a los metadatos (regla 4).

Regla No. 2 - La regla del acceso garantizado

Cada ítem de datos debe ser lógicamente accesible al ejecutar una búsqueda que combine el nombre de la tabla, su clave primaria, y el nombre de la columna.

Esto significa que dado un nombre de tabla, dado el valor de la clave primaria, y dado el nombre de la columna requerida, deberá encontrarse uno y solamente un valor. Por esta razón la definición de claves primarias para todas las tablas es prácticamente obligatoria.

Regla No. 3 - Tratamiento sistemático de los valores nulos

La información inaplicable o faltante puede ser representada a través de valores nulos

Un RDBMS (Sistema Gestor de Bases de Datos Relacionales) debe ser capaz de soportar el uso de valores nulos en el lugar de columnas cuyos valores sean desconocidos.

- Se reconoce la necesidad de la existencia del valor nulo, el cual podría servir para representar, o bien, una información desconocida (ejemplo, no se sabe la dirección de un empleado), o bien una información que no aplica(a un empleado soltero no se le puede asignar un nombre de esposa). Así mismo, consideremos el caso de un alumno que obtiene 0 puntos en una prueba y el de un alumno que no presentó la prueba.

- Hay problemas para soportar los valores nulos en las operaciones relacionales, especialmente en las operaciones lógicas, para lo cual se considera una lógica trivaluada, con tres (no dos) valores de verdad: Verdadero, Falso y null. Se crean tablas de verdad para las operaciones lógicas:

null AND null = null

Verdadero AND null = null

Falso AND null = Falso

Verdadero OR null = Verdadero, etc.

Regla No. 4 - La regla de la descripción de la base de datos

La descripción de la base de datos es almacenada de la misma manera que los datos ordinarios, esto es, en tablas y columnas, y debe ser accesible a los usuarios autorizados.

La información de tablas, vistas, permisos de acceso de usuarios autorizados, etc, debe ser almacenada exactamente de la misma manera: En tablas. Estas tablas deben ser accesibles igual que todas las tablas, a través de sentencias de SQL (o similar).

Regla No. 5 - La regla del sub-lenguaje Integral

Debe haber al menos un lenguaje que sea integral para soportar la definición de datos, manipulación de datos, definición de vistas, restricciones de integridad, y control de autorizaciones y transacciones.

Esto significa que debe haber por lo menos un lenguaje con una sintaxis bien definida que pueda ser usado para administrar completamente la base de datos.

Regla No. 6 - La regla de la actualización de vistas

Todas las vistas que son teóricamente actualizables, deben ser actualizables por el sistema mismo.

La mayoría de las RDBMS permiten actualizar vistas simples, pero deshabilitan los intentos de actualizar vistas complejas.

Regla No. 7 - La regla de insertar y actualizar

La capacidad de manejar una base de datos con operandos simples aplica no sólo para la recuperación o consulta de datos, sino también para la inserción, actualización y borrado de datos.

Esto significa que las cláusulas para leer, escribir, eliminar y agregar registros (SELECT, UPDATE, DELETE e INSERT en SQL) deben estar disponibles y operables, independientemente del tipo de relaciones y restricciones que haya entre las tablas o no.

Regla No. 8 - La regla de independencia física

El acceso de usuarios a la base de datos a través de terminales o programas de aplicación, debe permanecer consistente lógicamente cuando quiera que haya cambios en los datos almacenados, o sean cambiados los métodos de acceso a los datos.

El comportamiento de los programas de aplicación y de la actividad de usuarios vía terminales debería ser predecible basados en la definición lógica de la base de datos, y éste comportamiento debería permanecer inalterado, independientemente de los cambios en la definición física de ésta.

Regla No. 9 - La regla de independencia lógica

Los programas de aplicación y las actividades de acceso por terminal deben permanecer consistente lógicamente cuando quiera que se hagan cambios (según los permisos asignados) en las tablas de la base de datos.

La independencia lógica de los datos especifica que los programas de aplicación y las actividades de terminal deben ser independientes de la estructura lógica, por lo tanto los cambios en la estructura lógica no deben alterar o modificar estos programas de aplicación.

Regla No. 10 - La regla de la independencia de la integridad

Todas las restricciones de integridad deben ser definibles en los datos, y almacenables en el catálogo, no en el programa de aplicación.

Las reglas de integridad:

- Ningún componente de una clave primaria puede tener valores en blanco o nulos (ésta es la norma básica de integridad).
- Para cada valor de clave foránea deberá existir un valor de clave primaria concordante. La combinación de estas reglas aseguran que haya integridad referencial.

Regla No. 11 - La regla de la distribución

El sistema debe poseer un lenguaje de datos que pueda soportar que la base de datos esté distribuida físicamente en distintos lugares sin que esto afecte o altere a los programas de aplicación.

El soporte para bases de datos distribuidas significa que una colección arbitraria de relaciones, bases de datos corriendo en una mezcla de distintas máquinas y distintos sistemas operativos y que esté conectada por una variedad de redes, pueda funcionar como si estuviera disponible como en una única base de datos en una sola máquina.

Regla No. 12 - Regla de la no-subversión

Si el sistema tiene lenguajes de bajo nivel, estos lenguajes de ninguna manera pueden ser usados para violar la integridad de las reglas y restricciones expresadas en un lenguaje de alto nivel (como SQL).

Algunos productos solamente construyen una interfaz relacional para sus bases de datos No relacionales, lo que hace posible la subversión (violación) de las restricciones de integridad. Esto no debe ser permitido.

5. Experiencia Profesional

He trabajado en 3 diferentes instituciones bancarias, cada una de estas instituciones tiene diferentes métodos para trabajar, cada una de estas instituciones ha dejado aprendizaje, como el buen manejo de las metodologías ágiles, el diseño de base de datos, el trabajo en equipo que es indispensable en el desarrollo de software, así como el ser autodidacta.

5.1 Primera Institución

Ahí no tenían implementada ninguna metodología ágil, sólo se tenía un calendario de actividades, que nuca se cumplía porque los tiempos de pruebas y de desarrollos no estaban contabilizados. De este lugar aprendí que se deben poner en practica metodologías ágiles que ayuden a un mejor desempeño del equipo de trabajo así como el manejo real de tiempo de desarrollo de alguna etapa porque en muchas ocasiones al desarrollar se asignaban tiempo irreales, muy cortos o muy largos para una actividad específica. Es importante mencionar que en esta empresa el trato con los usuarios era cotidiano y a la par del desarrollo, esto facilitaba el control de cambios que se hacían pero también retrasaba el proyecto, pues se iban haciendo modificaciones y hasta que el usuario no diera el visto bueno no se terminaba la actividad. Referente al diseño de base de datos, no se tenía un diseño como tal, iniciaron con el diseño de una base que era como el proyecto principal de la empresa y en ese se iban agregando las tablas necesarias, al final del día muchas tablas quedaban sin ser usadas o muchas otras con registros duplicados, incongruente y la mayoría de las veces datos son redundantes, esto se debe a no aplicar una normalización correcta, en cuanto al desarrollo de *software*, programación, el equipo de trabajo estaba muy organizado, se asignaban tareas concretas y no se podía comenzar otra hasta terminar la actual.

En esta Institución participe en un sistema completamente nuevo, este sistema se desarrolló para las diferencias de billetes que se tenían de las sucursales bancarias que trabajaban para esta empresa.

Este sistema tenía como objetivo mantener almacenados reportes en una base de datos, estos reportes los realizaban las personas que trabajan en el conteo de los billetes y detectan alguna diferencia, sobraban o faltaban billetes en el mazo (100 billetes de la misma denominación). También el sistema requería que el supervisor del área pudiera llevar el control de diferencias encontradas al mes, de que institución bancaria pertenecía para obtener datos estadísticos. Finalmente el sistema debía contar con una opción para que el jefe del área o cualquier otro directivo pudieran consultar los reportes así como los datos estadísticos.

Este sistema se desarrolló en el lenguaje de programación Java, todas las interfaces gráficas del sistema fueron hechas con la biblioteca Swing, para que el sistema se presentara de la misma forma en cualquier plataforma. La base de datos se implementó en el manejador Sybase pues era el manejador de base de datos que la institución usaba para aplicaciones stand-alone.

DEL DESARROLLO

Este sistema se dividió en 3 partes:

- La primera parte comprendía el tiempo que se invertiría en el diseño del front del usuario, el funcionamiento de cada componente en la pantalla, así como el flujo que llevaría cada uno de los roles en el sistema.

  A continuación se muestran los diagramas de caso de Uso para cada actor (vista) de usuario dentro del sistema.

  El sistema fue diseñado para 3 actores (Figura 24):

  - *Operador:* Es quien captura los reportes sobre las diferencias de efectivo
  - *Supervisor:* Es quien revisa y autoriza cambios en los reportes sobre las diferencias de efectivo, y hace un compendio de los reportes por mes (Reporte Mensual).
  - *Directivo:* Es quien únicamente revisa los reportes realizados.

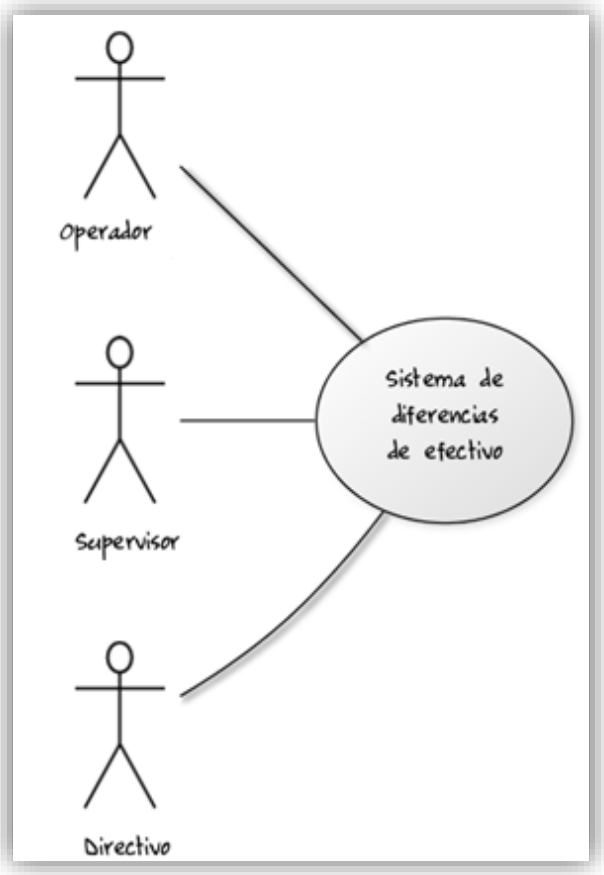


Figura 24. Roles del sistema de diferencias de Efectivo

La Figura 25 muestra la interacción que tendrá cada uno de los roles mencionados en el punto anterior con el sistema, caso de uso.

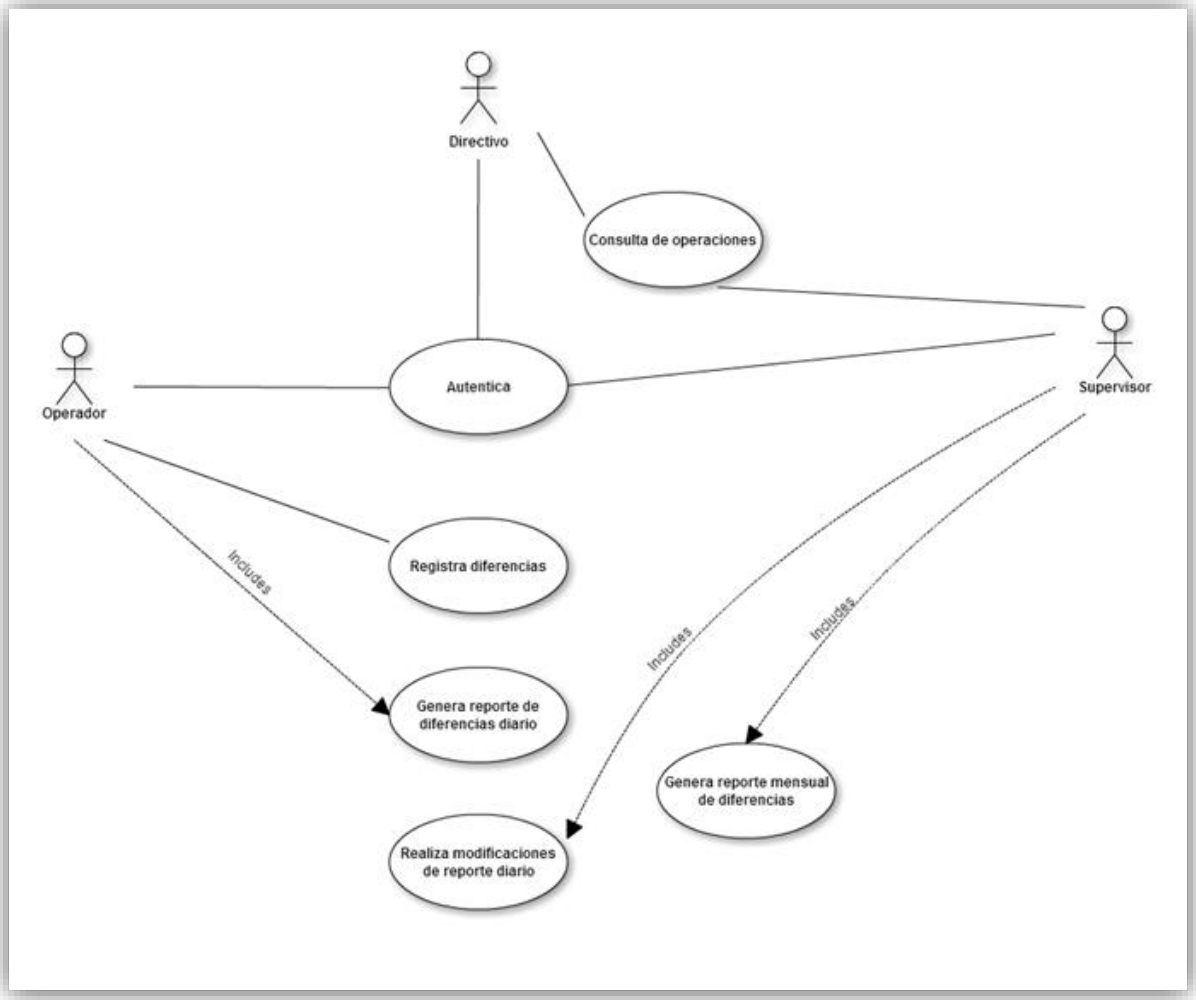


Figura 25. Diagrama de uso por rol.

- La segunda parte comprendía el tiempo que se invertía en el desarrollo del negocio, es decir comprobar que los datos que se mostraban en el front eran los correctos así como los que se guardaban correspondían a los que el usuario había colocado (diseño y conexión con base de datos). A continuación se muestra el diseño de base de datos que se realizó (figura 26).

Diagrama entidad-relación

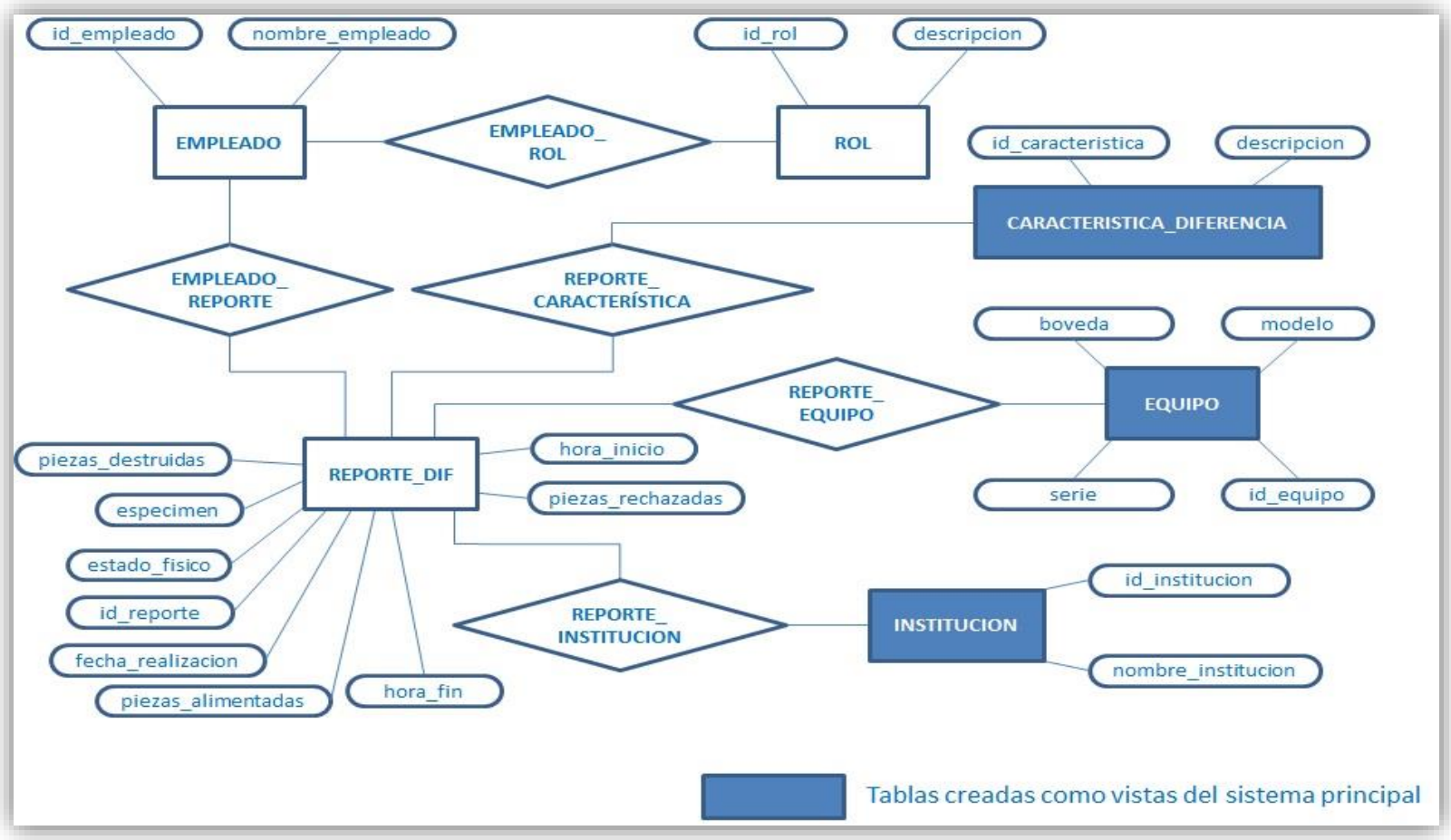


Figura 26. Modelo Entidad - Relación del sistema de diferencias

Descripción de Entidades:

- EMPLEADO: Es un catálogo que contiene la información sobre los usuarios que podrán usar el sistema.

- ROL: Es un catálogo que contiene la información sobre cada uno de los roles a los que pueden estar asociados los empleados.

- REPORTE_DIF: Esta entidad tiene contenida la información acerca de los reportes que se pueden obtener en el sistema, como son los eBUmpleados que participaron en ese reporte, las diferencias obtenidas, la máquina en que se procesó el billete, etc.

- MODIFICACION_REPORTE_DIF: Esta entidad tiene la información sobre el número de modificaciones que tiene el reporte y por quien fue realizada dicha modificación.

- CARACTRISTICA_DIFERECIA: Es un catálogo que contiene los diferentes tipos de diferencias que se pueden hallar al ser procesado el billete.

- EQUIPO: Es un catálogo que contiene la información de los equipos en los cuales se trabaja y así como la ventanilla en la que se encuentran.

- INSTITUCION: Entidad que contiene los datos indispensables de las instituciones financieras de donde se obtienen los billetes deteriorados

Diccionario de Datos

Notación del diccionario de datos.

= Está compuesto de.

+ Y.

() Optativo (puede o no estar presente).

[] Seleccionar una de varias alternativas.

** Comentarios.

@ Identificador (campo llave).

| Separa opciones alternativas.

Diccionario de Datos	
Campo	Descripción
BOVEDA	Bóveda en la que se encuentra el equipo BPS. Campo de tipo carácter con longitud de 15
CLAVE_EMP	Clave de empleado brindado por la institución bancaria, sirve para identificar el rol del usuario en el sistema. Campo de tipo numérico con longitud de 9
COMENTARIO	Comentario que se hace sobre las diferencias de sustitución. Campo de tipo carácter con longitud de 15
DESCRIPCION_CARACTERISTICA	Describe la característica de la diferencia. Campo de tipo carácter con longitud de 50
DESCRIPCION_INSTITUCION	Describe el nombre de la institución bancaria de la que procede la diferencia. Campo de tipo carácter con longitud de 15
DESCRIPCION_ROL	Describe los tipos de rol existentes en el sistema. Campo de tipo carácter con longitud de 30
EDO_FISICO	Estado físico en el que se encuentra el billete procesado. Campo de tipo carácter con longitud de 15
EMPLEADO	Archivo que contiene los datos del empleado @clave_emp
EQUIPO	Archivo que contiene los datos del equipo (Máquina lectoclasificadora) @id_equipo
ESPECIMEN	Espécimen que puede tener el billete, es decir; la denominación y a que familia pertenece el billete. Campo de tipo carácter con longitud de 15

Diccionario de Datos	
Campo	Descripción
FECHA_REALIZACION	Fecha en la que se realiza el reporte. Campo de tipo Fecha
HORA_FIN	Hora al final de la jornada laboral, debe coincidir con la hora registrada en la máquina BPS. Campo de tipo carácter con longitud de 15
HORA_INICIO	Hora inicial de la jornada laboral, debe coincidir con la hora registrada en la máquina BPS. Campo de tipo carácter con longitud de 15
ID_CARACTERISTICA	Identificador de la característica que contiene la diferencia. Campo de tipo numero con longitud de 9
ID_EQUIPO	Identificador del equipo en el que se procesan los billetes. Campo de tipo numérico con longitud de 9
ID_INSTITUCION	Identificador de la institución de la que proviene el billete procesado. Campo de tipo numérico con longitud de 9
ID_REPORTE	Identificador de los reportes diarios de diferencias. Campo de tipo numérico con longitud de 9
ID_REPORTE_INSTITUCION	Identificador de la institución y el reporte en el que se encuentra asociada. Campo de tipo numérico con longitud de 9
ID_REPORTE_MENSUAL	Identificador del reporte mensual. Campo de tipo numérico con longitud de 9
ID_ROL	Identificador del rol de usuarios. Campo de tipo numérico con longitud de 9
ID_ROL_EMPLEADO	Identificador del rol asociado a cada empleado. Campo de tipo numérico con longitud de 9
INSTITUCION	Archivo que contiene la información sobre la institución de la que proviene la diferencia
INSTITUCION_DIFERENCIA	Archivo que contiene la información sobre la institución de la que proviene la diferencia y las diferencias asociadas @ID_INSTITUCION_DIFERENCIA
MODELO_EQUIPO	Modelo de máquina BPS que procesa el billete en el reporte descrito. Campo de tipo carácter con longitud de 15
NOMBRE_COMPLETO	Nombre completo del empleado. Campo de tipo carácter con longitud de 50

Diccionario de Datos	
Campo	Descripción
NUMERO_PIEZAS	Número de piezas que se procesan. Campo de tipo numérico con longitud de 15
PIEZAS_ALIMENTADAS	Número de piezas alimentadas a la máquina BPS. Campo de tipo numérico con longitud de 15
PIEZAS_DESTRUIDAS	Número de piezas destruidas por la máquina BPS. Campo de tipo numérico con longitud de 15
PIEZAS_RECHAZADAS	Número de piezas rechazadas por la máquina BPS. Campo de tipo numérico con longitud de 15
REPORTE_DIF	Archivo que contiene los datos del reporte de diferencia diario @ID_REPORTE
REPORTE_INSTITUCION	Archivo que contiene la información sobre la institución de la que proviene la diferencia y los reporte que tiene asociados @ID_REPORTE_INSTITUCION
REPORTE_MENSUAL	Archivo que contiene la información sobre el reporte mensual @ID_REPORTE_MENSUAL
REPORTE_ROLEMPLEADO	Archivo que contiene los reportes y los empleados asociados al mismo @ID_REPORTE_ROLEMPLEADO
ROL	Archivo que contiene la información sobre los diferentes tipos de rol que contiene el sistema @ID_ROL
ROL_EMPLEADO	Archivo que contiene la información del empleado y los roles que tiene asociados @ID_EMPLEADO_ROL
SERIE_EQUIPO	Serie de la máquina BPS que procesa los billetes. Campo de tipo carácter con longitud de 15
TOTAL_FALTANTES	Número total de piezas faltantes al final de la jornada laboral. Campo de tipo numérico con longitud de 15
TOTAL_SOBRANTE	Número total de piezas sobrantes al final de la jornada laboral. Campo de tipo numérico con longitud de 15

A continuación se describen las entidades de acuerdo a los datos que se encuentran el diccionario de datos.

Nomenclatura.

> Llave primaria

E = Entero

N = Numérico

D = Decimal

C = Carácter

L = Lógico

DATE = Tipo fecha (longitud de 8 caracteres)

NOMBRE DE LA TABLA:	CARACTERISTICA_DIFERENCIA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_CARACTERISTICA	N E9 D0	ID_CARACTERISTICA
DESCRIPCION_CARACTERISTICA	C50	DESCRIPCION_CARACTERISTICA

NOMBRE DE LA TABLA:	EMPLEADO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>CLAVE_EMP	N E9 D0	CLAVE_EMP
NOMBRE_COMPLETO	C30	NOMBRE_COMPLETO

NOMBRE DE LA TABLA:	EQUIPO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EQUIPO	N E9 D0	ID_EQUIPO
BOVEDA	C30	BOVEDA
MODELO	C30	MODELO
SERIE	C30	SERIE

NOMBRE DE LA TABLA:	INSTITUCION	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_INSTITUCION	N E9 D0	ID_INSTITUCION
DESCRIPCION_INSTITUCION	C30	DESCRIPCION_INSTITUCION

NOMBRE DE LA TABLA:	INSTITUCION_DIFERENCIA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_REPORTE_INSTITUCION	N E9 D0	ID_REPORTE_INSTITUCION
>ID_CARACTERISTICA	N E9 D0	ID_CARACTERISTICA
COMENTARIO	C50	COMENTARIO
NUMERO_PIEZAS	N E15 D0	NUMERO_PIEZAS

NOMBRE DE LA TABLA:	REPORTE_DIF	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_REPORTE	N E9 D0	ID_REPORTE
EDO_FISICO	C30	EDO_FISICO
ESPECIMEN	C30	ESPECIMEN
FECHA_REALIZACION	DATE	FECHA_REALIZACION
HORA_FIN	C15	HORA_FIN
HORA_INCIO	C15	HORA_INCIO
PIEZAS_ALIMENTADAS	N E15 D0	PIEZAS_ALIMENTADAS
PIEZAS_DESTRUIDAS	N E15 D0	PIEZAS_DESTRUIDAS
PIEZAS_RECHAZADAS	N E15 D0	PIEZAS_RECHAZADAS

NOMBRE DE LA TABLA:	REPORTE_INSTITUCION	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_REPORTE_INSTITUCION	N E9 D0	ID_REPORTE_INSTITUCION
ID_INSTITUCION	N E9 D0	ID_INSTITUCION
ID_REPORTE	N E9 D0	ID_REPORTE
TOTAL_FALTANTES	N E15 D0	TOTAL_FALTANTES
TOTAL_SOBRANTES	N E15 D0	TOTAL_SOBRANTES

NOMBRE DE LA TABLA:	REPORTE_MENSUAL	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_REPORTE_MENSUAL	N E9 D0	ID_REPORTE_MENSUAL
FECHA_REALIZACION	DATE	FECHA_REALIZACION
ID_REPORTE	N E9 D0	ID_REPORTE

NOMBRE DE LA TABLA:	REPORTE_ROLEMPLEADO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_REPORTE_ROLEMPLEADO	N E9 D0	ID_REPORTE_ROLEMPLEADO
ID_REPORTE	N E9 D0	ID_REPORTE

NOMBRE DE LA TABLA:	ROL	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_ROL	N E9 D0	ID_ROL
DESCRIPCION_ROL	C30	DESCRIPCION_ROL

NOMBRE DE LA TABLA:	ROL_EMPLEADO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_ROL_EMPLEADO	N E9 D0	ID_ROL_EMPLEADO
CLAVE_EMP	N E9 D0	CLAVE_EMP
ID_ROL	C30	ID_ROL

- La tercera parte comprendía el tiempo de pruebas que el usuario tendría para confirmar lo solicitado en los requerimientos, así como realizar las mejoras que se solicitaran durante las pruebas

En las figuras 27 a 32 se muestra un ejemplo de las ventanas creadas para el sistema

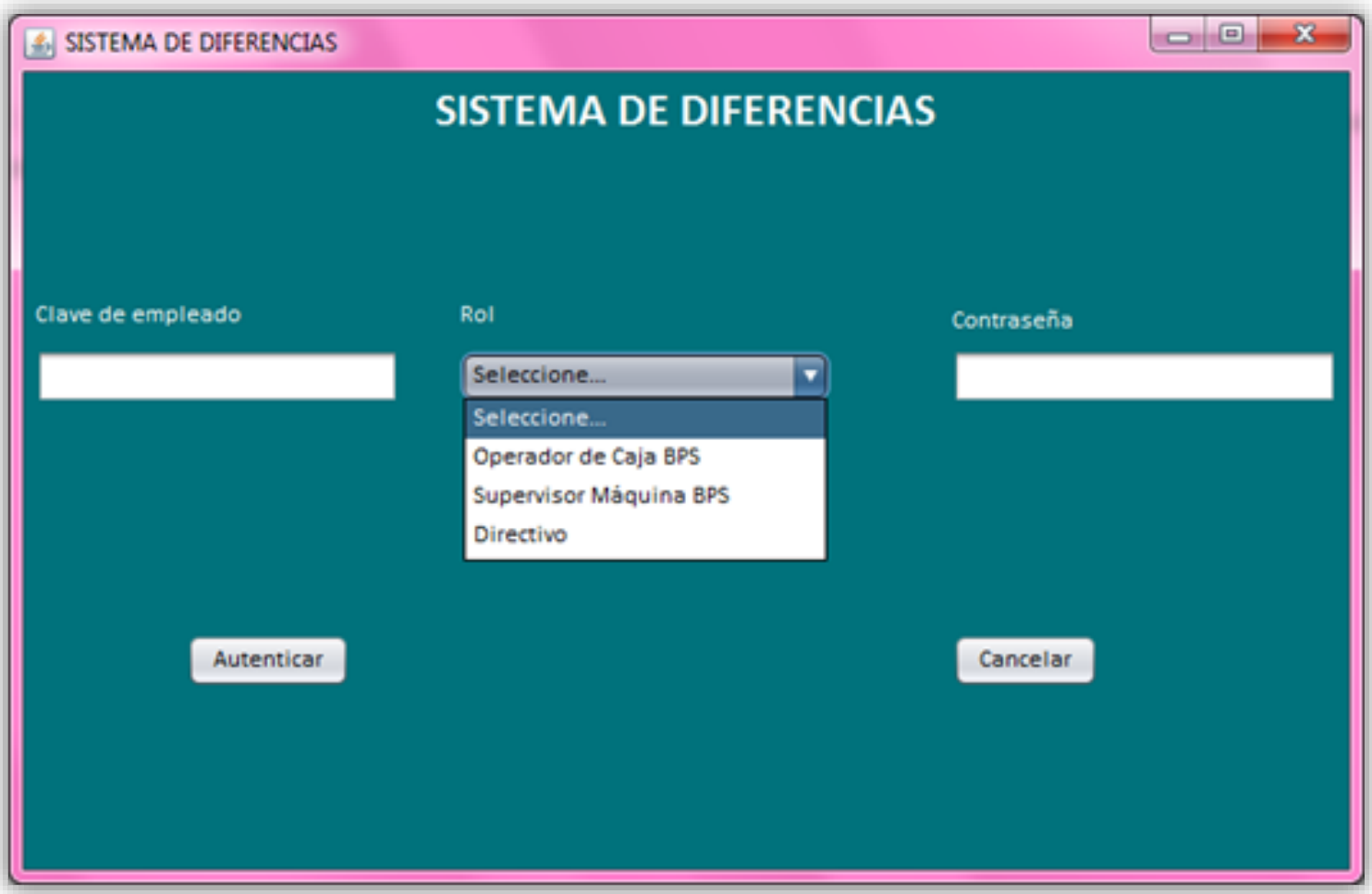


Figura 27. Ejemplo de ventana Sistema de Diferencias

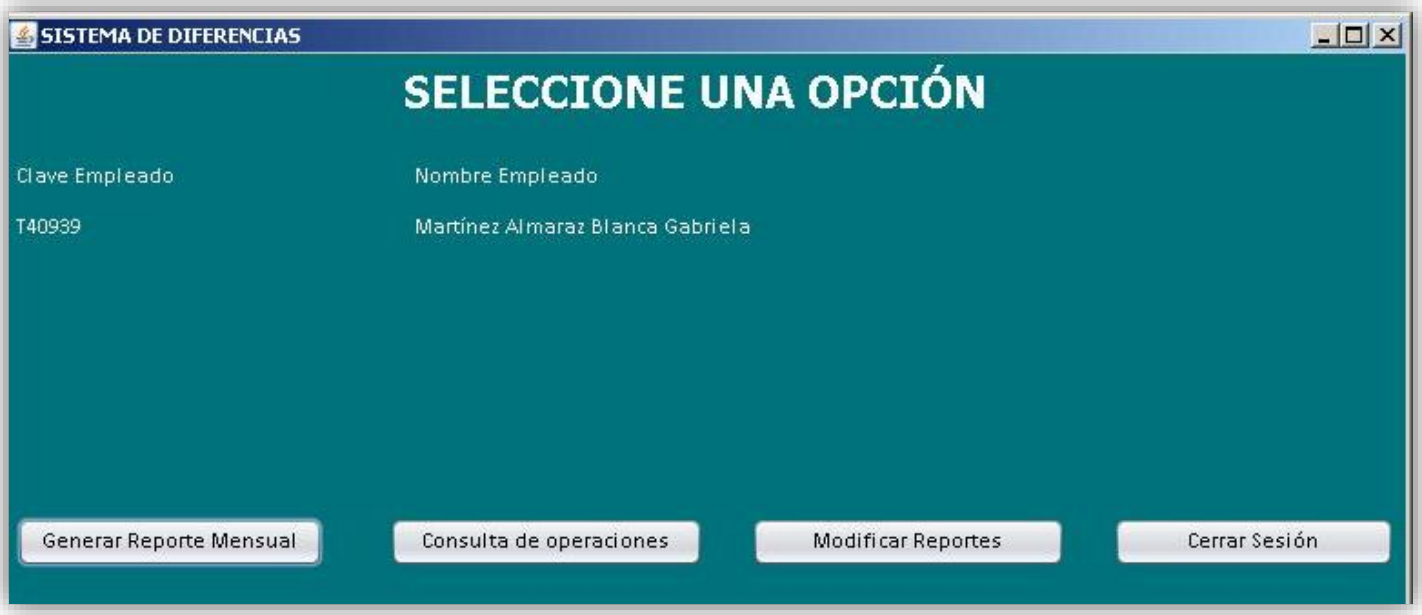


Figura 28. Ejemplo de ventana Sistema de Diferencias

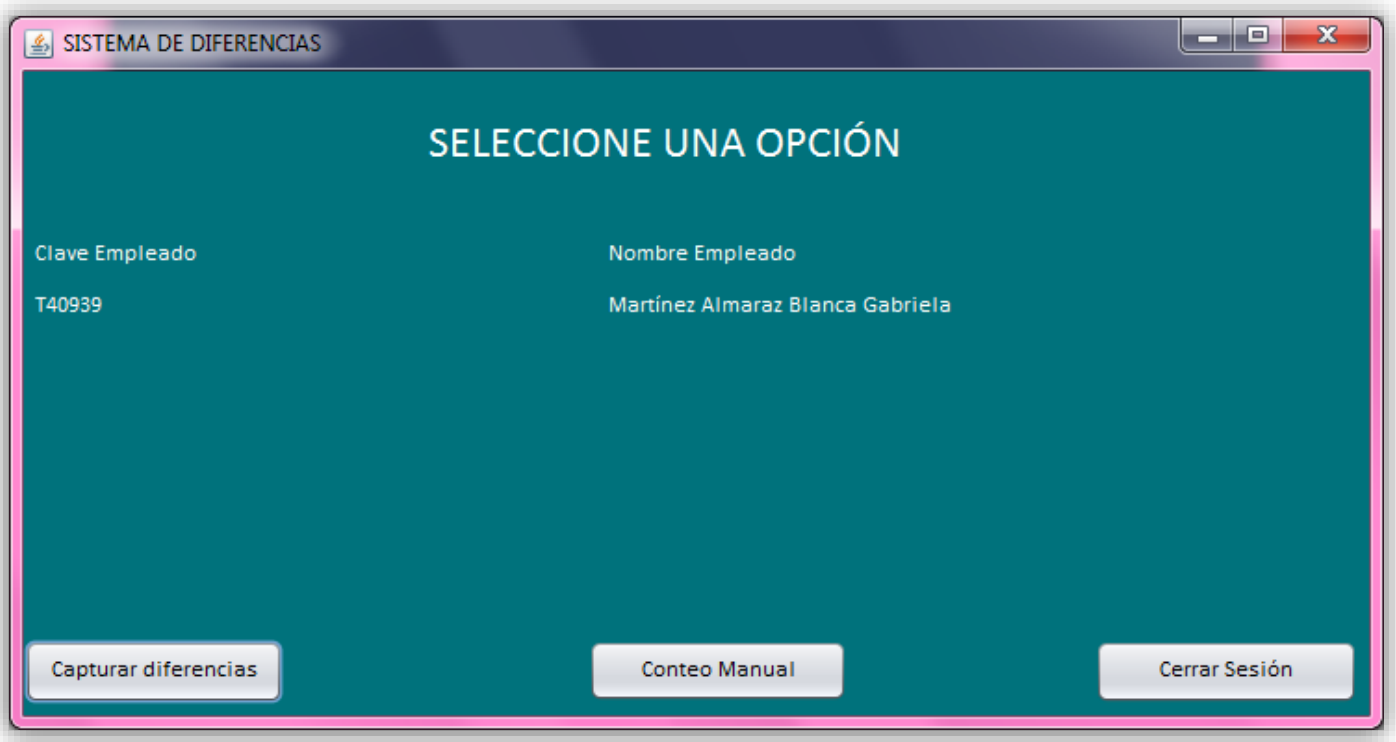


Figura 29. Ejemplo de ventana Sistema de Diferencias

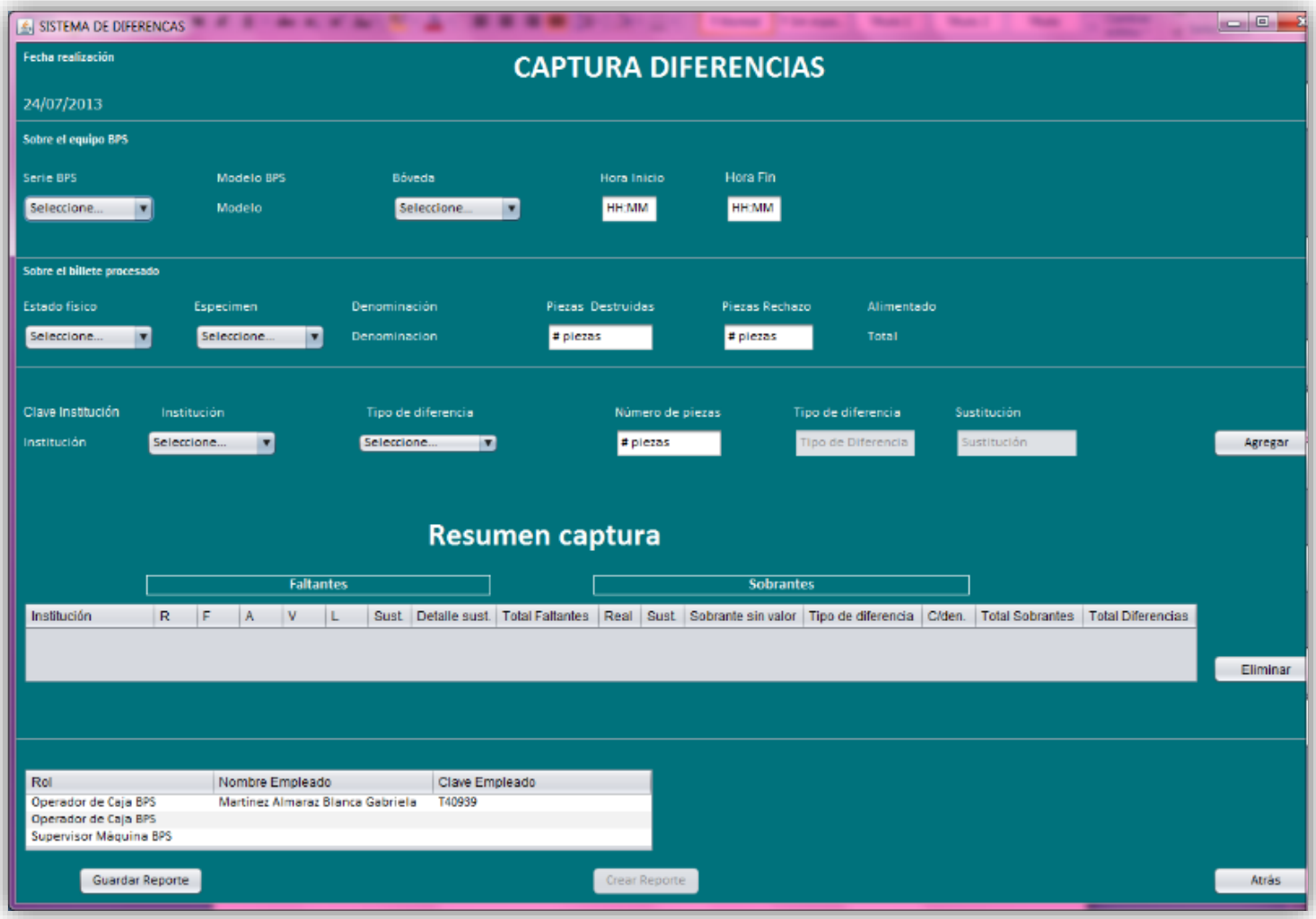


Figura 30. Ejemplo de ventana Sistema de Diferencias

Figura 31. Ejemplo de ventana Sistema de Diferencias

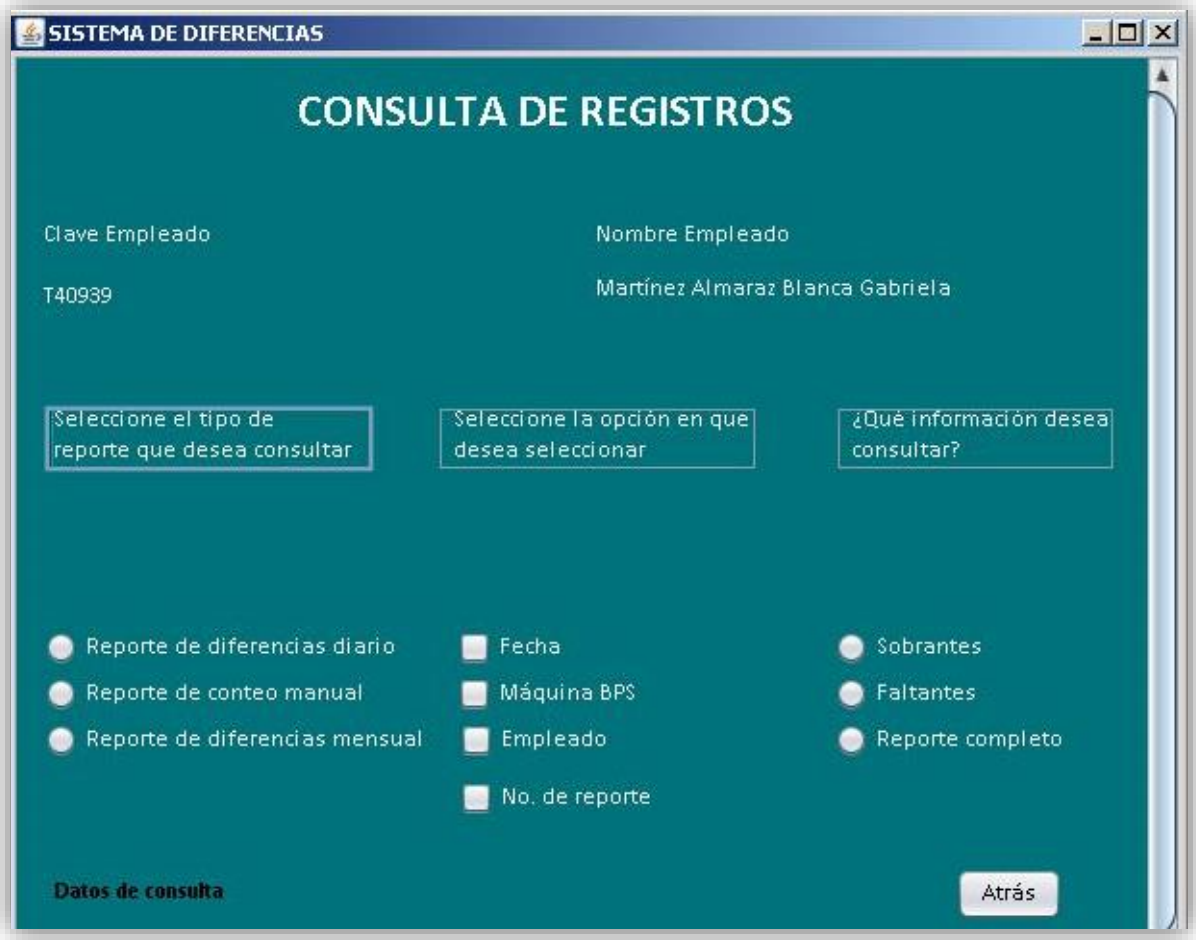


Figura 32. Ejemplo de ventana Sistema de Diferencias

IMPACTO

Para la captura de reportes en la Institución se usaba un archivo de Excel donde se escribían, a mano, las diferencias que se tenían al final del día, esto provocaba que hubiera demasiados errores por parte de los operadores en cada uno de estos reportes, con el desarrollo de este proyecto se redujo el número de errores en la captura de los mismos, a su vez el supervisor del área tenía que copiar los detalles de cada reporte diario para generar el reporte mensual, ahora con el sistema de diferencias los reportes mensuales se pueden hacer de manera dinámica lo cual también impacta en la disminución de errores, las búsquedas de reportes se simplificaron pues ya no es necesario buscar entre documentos impresos por que el sistema de diferencias lo lleva a cabo si el usuario lo requiere.

Por lo anterior el área también ha tenido una disminución de consumibles (hojas de papel, tinta de impresora, etc.), pues antes los reportes se imprimían diariamente ahora ya no es necesario pues se almacenan en una base de datos.

Con este proyecto el área de sistemas de la institución financiera pudo obtener un mayor desempeño de los empleados por que en menos tiempo podían capturar los reportes y presentarlos sin retrasos.

En este proyecto el usuario estuvo involucrado continuamente por lo que las mejoras se hacían continuamente, lo que reducía cambios en la etapa de pruebas. El proyecto fue liberado y a la par se tenía trabajando el sistema que previamente era utilizado, esto para que el usuario se habituara al nuevo sistema y se pudiera tener respaldo si es que el nuevo sistema tenía alguna falla.

CONCLUSIONES

El sistema actualmente se encuentra en funcionamiento, cumpliendo con los requerimientos solicitados por el usuario, fue realizado en tiempo y forma de acuerdo al plan de trabajo que teníamos al inicio del desarrollo. Este proyecto fue gratificante porque fue mi primera experiencia laboral exitosa.

Este proyecto fue de gran beneficio para mí pues mi participación desde el inicio me sirvió para saber cómo desarrollar un proyecto desde el inicio, además el manejo de los modelos UML me ayudo durante la comunicación con el usuario para que el proyecto diera a la Institución Financiera el producto que requería.

5.2 Segunda Institución

Primer proyecto

DEL DESARROLLO

- XP (Xtreme Programming) fue usado en el Sistema de Casa de Bolsa (SCB) de la Institución Financiera para la que trabajé, el desarrollo era completamente nuevo. Se asignó un líder de proyecto (Entrenador) quien asignaba las tareas y también era el encargado de entablar conversaciones, con respecto a las necesidades que se requerían para el sistema, con el cliente. Para este proyecto la mayoría de los desarrolladores eran nuevos, por lo que se requería que el equipo aprendiera sobre el negocio así como información técnica, todo el aprendizaje fue gracias a que los desarrolladores trabajaban en pequeños equipos de 3 personas, cada uno de estos equipos tenía su propio líder quien sabía sobre el negocio de Casa de Bolsa y les enseñaba un poco del negocio para llevar la parte técnica de manera correcta las dos personas restantes trabajaban en conjunto probablemente no se trabajaba en el mismo punto pero se desarrollaba a la par el trabajo para que al final se pudiera completar en tiempo y forma el desarrollo. El líder de este pequeño grupo de 3 personas también era quien llevaba a cabo la parte de pruebas (tester) y daba retroalimentación (tracker) a los desarrolladores sobre su trabajo.

  Otra de las características que persistieron en el desarrollo del proyecto, era el que todo el equipo tenía el conocimiento de todo el sistema, así que cualquier desarrollador podía hacer una mejora dentro del sistema sin complicación. Las pruebas unitarias también se hacía pero yo considero que el hacer este tipo de pruebas genera conflicto más si el desarrollo es complejo y con cambios distintos, al hacer pruebas unitarias no sabemos si nuestro nuevo desarrollo causo conflicto con desarrollo anterior, considero que es mejor hacer pruebas completas sobre todo el desarrollo o por lo menos entregas recientes con quien se comparten módulos. Pero esta parte también permitió que los errores que surgieran en el desarrollo se pudieran solucionar inmediatamente.

  Las mejoras que se le hacía al sistema eran pequeñas pero quedaban al gusto del cliente, pues participaba activamente en el desarrollo del sistema.

  Al inicio del proyecto se estableció que tendría un ciclo de vida con el modelo en cascada, pues permite el contacto con el cliente así como el avance incremental del sistema, esto hace que el proyecto esté completamente documentado. En cuanto a la base de datos, al principio se hizo un modelo ER (Entidad-Relación) genérico y a medida que crecía el sistema el modelo de la base de datos también lo hacía, no se tenía complicación al insertar las tablas pues como desde un principio se había hecho un modelo normalizado, las tablas sólo completaban el modelo.

En la figura 33 se muestra parte del modelo Entidad - Relación de la base de datos

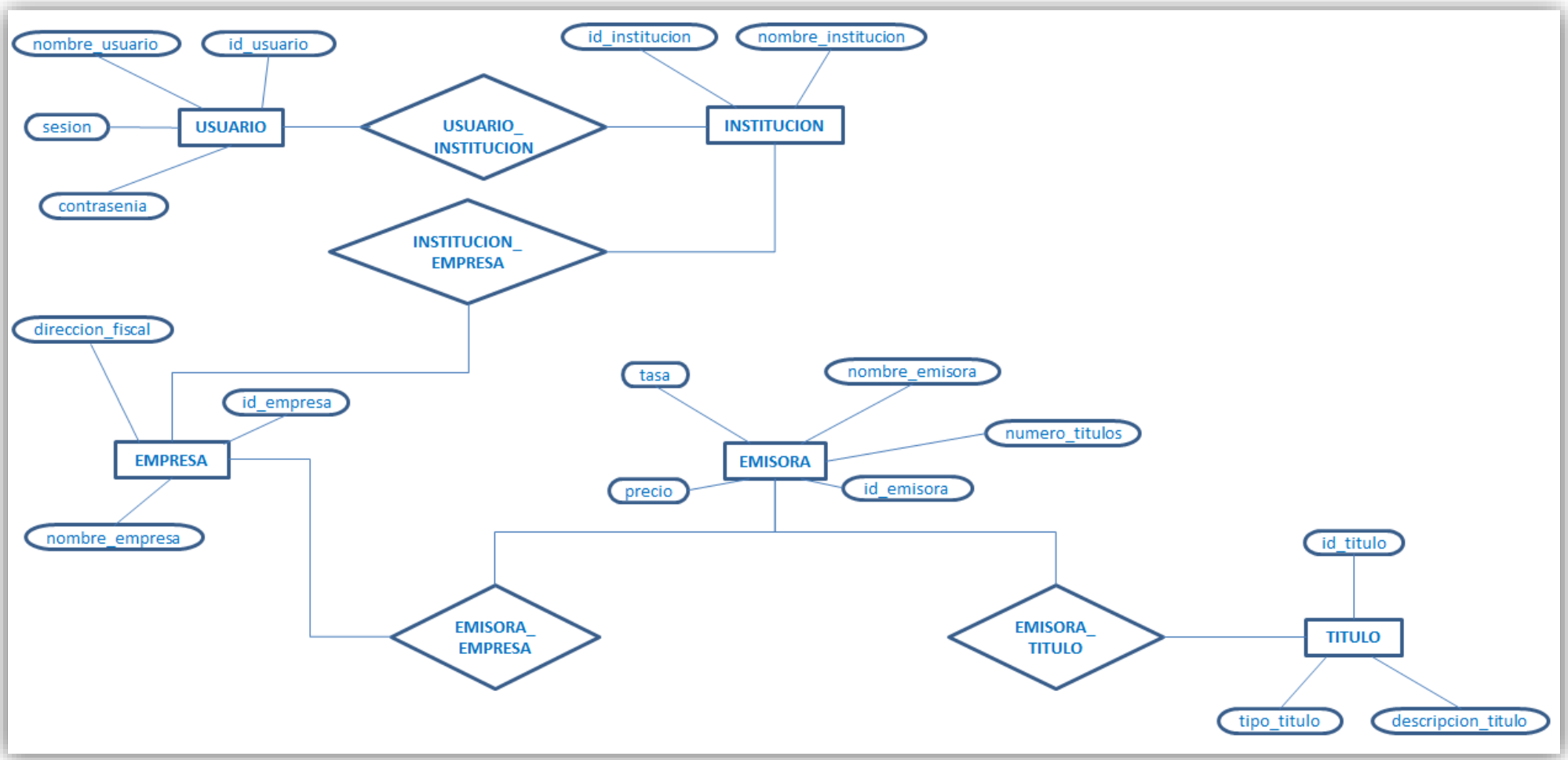


Figura 33. Modelo Entidad-Relación del Sistema de Casa Bolsa

Descripción de Entidades:

- USUARIO: Es un catálogo que contiene la información sobre los usuarios que podrán usar el sistema.

- INSTITUCION: Es un catálogo que contiene la información sobre la Institución que compra emisoras.

- EMISORA: Contiene la información de las emisoras que tiene disponible para compra y venta títulos de su empresa.

- EMPRESA: Es un catálogo que contiene la información sobre la empresa que está poniendo en venta emisoras

- TITULO: Es un catálogo que contiene la información sobre los documentos comprados a la empresa, como son el tipo de título, descripción, etc.

Diccionario de Datos

Notación del diccionario de datos.

= Está compuesto de.

+ Y.

() Optativo (puede o no estar presente).

[] Seleccionar una de varias alternativas.

** Comentarios.

@ Identificador (campo llave).

| Separa opciones alternativas.

Diccionario de Datos	
Campo	Descripción
CONTRASENIA	Contraseña de usuario, cifrada con algoritmo RSA. Campo de tipo carácter con longitud de 50
DESCRIPCION_TITULO	Descripción del tipo de título. Campo de tipo carácter con longitud de 50
DIRECCION_FISCAL	Dirección fiscal de la empresa que tiene en venta emisoras. Campo de tipo carácter con longitud de 50
EMISORA	Archivo que contiene la información sobre las emisoras que se encuentran en compra / venta @id_emisora
EMPRESA	Archivo que contiene la información sobre la empresa que tiene en venta emisoras @id_empresa
ID_EMISORA	Identificador de la emisora que se encuentra en compra / venta. Campo de tipo carácter con longitud de 9
ID_EMPRESA	Identificador de la empresa que tiene en ventas emisoras. Campo de tipo carácter con longitud 6
ID_INSTITUCION	Identificador de la Institución que puede comprar emisoras. Campo de tipo carácter con longitud de 6
ID_TITULO	Identificador del tipo de título. Campo de tipo carácter con longitud de 6
ID_USUARIO	Identificador del usuario, proporcionado por la Institución Financiera. Campo de tipo carácter con longitud de 6

Diccionario de Datos	
Campo	Descripción
INSTITUCION	Archivo que contiene la información sobre la Institución que comprara emisoras. @id_institucion
NOMBRE_EMISORA	Nombre de la emisora que se encuentra en compra / venta. Campo de tipo carácter con longitud de 50
NOMBRE_EMPRESA	Nombre de la empresa que tiene en venta emisoras. Campo de tipo carácter con longitud de 50
NOMBRE_INSTITUCION	Nombre de la institución que comprara emisoras. Campo de tipo carácter con longitud de 50
NOMBRE_USUARIO	Nombre del usuario del sistema. Campo de tipo carácter con longitud de 50
NUMERO_TITULOS	Número de títulos que posee la emisora que se encuentra en compra / venta. Campo de tipo numérico con longitud de 9
PRECIO	Precio que posee la emisora que se encuentra en compra / venta. Campo de tipo numérico con longitud de 9
SESION	Fecha de último acceso del usuario. Campo de tipo Fecha
TASA	Tasa con la que se pactó la compra / venta de la emisora. Campo de tipo numérico con longitud de 9
TIPO_TITULO	Tipo de título. Campo de tipo carácter con longitud de 15
TITULO	Archivo que contiene la información sobre el tipo de títulos que posee la emisora que se encuentra en compra / venta
USUARIO	Archivo que contiene la información sobre los usuario del sistema @id_usuario

A continuación se describen las entidades de acuerdo a los datos que se encuentran el diccionario de datos.

Nomenclatura.

> Llave primaria

E = Entero

N = Numérico

D = Decimal

C = Carácter

L = Lógico

DATE = Tipo fecha (longitud de 8 caracteres)

NOMBRE DE LA TABLA:	EMISORA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMISORA	C6	ID_EMISORA
ID_EMPRESA	C6	ID_EMPRESA
ID_TITULO	C6	ID_TITULO
NOMBRE_EMISORA	C50	NOMBRE_EMISORA
NUMERO_TITULOS	N E9 D0	NUMERO_TITULOS
PRECIO	N E7 D2	PECIO
TASA	N E7 D2	TASA

NOMBRE DE LA TABLA:	EMPRESA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMPRESA	C6	ID_EMPRESA
DIRECCION_FISCAL	C50	DIRECCION_FISCAL
NOMBRE_EMPRESA	C50	NOMBRE_EMPRESA

NOMBRE DE LA TABLA:	INSTITUCION	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_INSTITUCION	C6	ID_INSTITUCION
NOMBRE_INSTITUCION	C50	NOMBRE_INSTITUCION

NOMBRE DE LA TABLA:	INSTITUCION_EMPRESA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMPRESA	C6	ID_EMPRESA
>ID_INSTITUCION	C6	ID_INSTITUCION

NOMBRE DE LA TABLA:	TITULO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
> ID_TITULO	C6	ID_TITULO
DESCRIPCION_TITULO	50	DESCRIPCION_TITULO
TIPO_TITULO	C15	TIPO_TITULO

NOMBRE DE LA TABLA:	USUARIO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_USUARIO	C6	ID_USUARIO
CONTRASENIA	C50	CONTRASENIA
ID_INSTITUCION	C6	ID_INSTITUCION
NOMBRE_USUARIO	C50	NOMBRE_USUARIO
SESION	DATE	SESION

La arquitectura de los sistemas en la Institución Financiera se muestra en la Figura 34

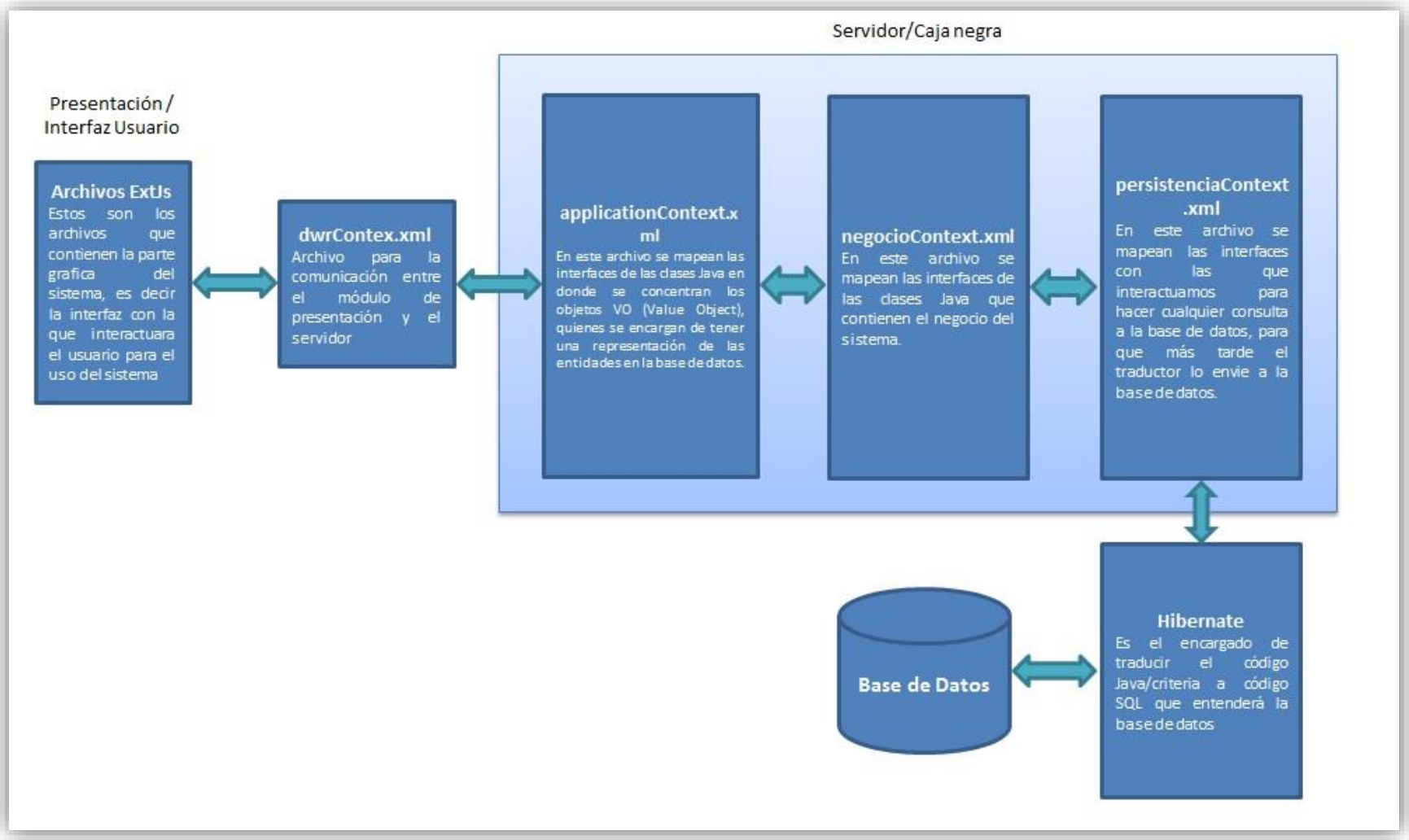


Figura 34. Diagrama PROYECTO GENERAL

En las figuras 35 a 38 se muestra un ejemplo de las ventanas que se crearon para el Sistema de Casa de Bolsa

Figura 35. Ejemplo de ventanas del Sistema Casa de Bolsa

Figura 36. Ejemplo de ventanas del Sistema Casa de Bolsa

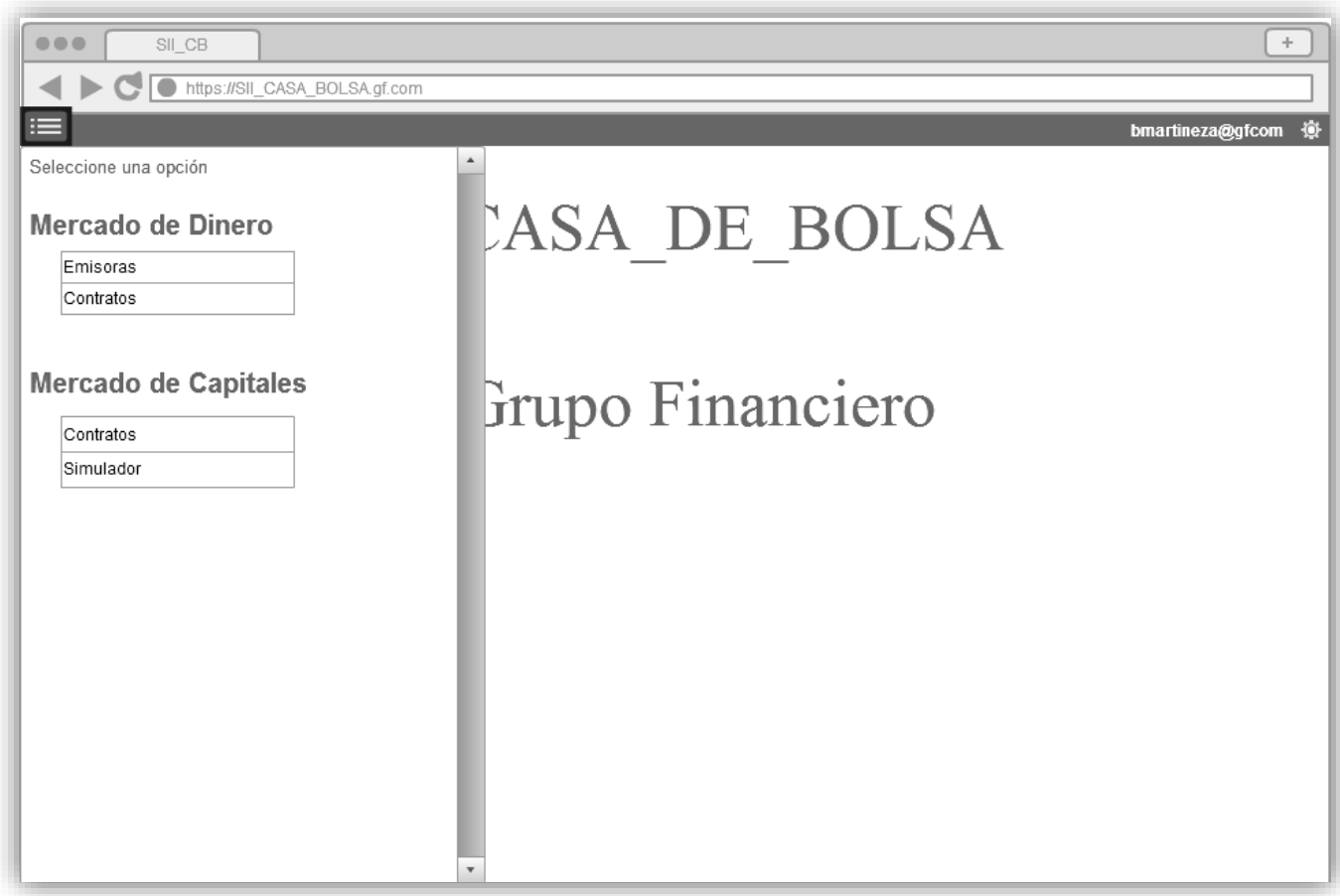


Figura 37. Ejemplo de ventanas del Sistema Casa de Bolsa

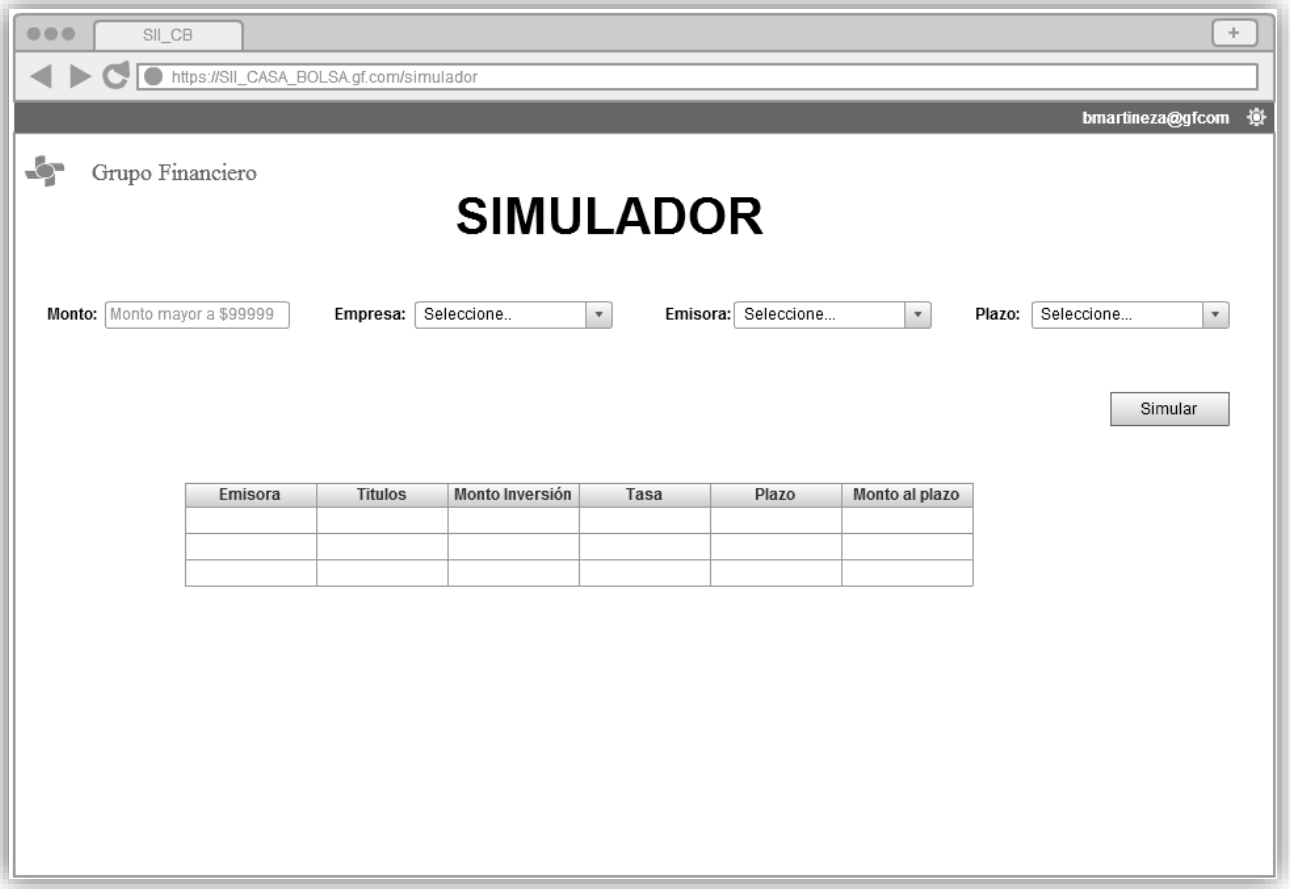


Figura 38. Ejemplo de ventanas del Sistema Casa de Bolsa

IMPACTO

Antes del SCB no se tenía ningún sistema que realizará las operaciones de Casa de Bolsa, por lo que este proyecto ayudó a tener más clientes no sólo como Institución Bancaria sino como un Grupo Financiero el que pudiera ofrecerle al cliente más opciones de productos, y su vez la institución tuviera más clientes generando mayores ingresos a la misma.

El SCB cumplió el objetivo que se tenía planteado, el cual era mantener a la institución bancaria de manera competitiva en la Bolsa Mexicana de Valores, actualmente el sistema se encuentra funcionando en la institución financiera. Una vez finalizado el proyecto se conformó un equipo de soporte quien dio la capacitación para los usuarios, esta nueva área es quien, ahora, toma nota sobre las mejoras y las hace llegar al equipo de desarrollo para continuar con el mantenimiento del sistema.

Con este proyecto la Institución Financiera obtuvo un gran número de clientes, ahora como inversionistas, lo que trajo grandes ganancias.

CONCLUSIONES

El manejo de una metodología ágil, llevo al equipo de programadores a tener un mejor control del tiempo en cuanto al desarrollo, además se podía identificar en que parte del sistema se encontraba trabajando cada miembro del equipo.

Con la metodología XP es más fácil la corrección de errores pues antes de que lleguen las pruebas al usuario se pasa por un equipo de test que ayuda a mitigar los errores de programación.

El que el usuario intervenga en cada una de las etapas del desarrollo hace que el producto cumpla con las necesidades obtenidas en la toma de requerimientos, además mejora el desempeño del sistema ya que se pueden obtener las mejoras antes de ponerlo en funcionamiento en producción.

Segundo Proyecto

DEL DESARROLLO

- *Scrum* fue usado en el Sistema de Inversiones (SI), esta metodología ágil funciono, el equipo de trabajo tenía actividades precisas asignadas por el *Scrum* Master (SM) quien en este caso, aunque la metodología habla de no tener un líder, era nuestro líder y quien hablaba con el Product Owner(PO) juntos priorizaban los entregables, dichas actividades se podían atender en tiempo y forma, para mantener en orden cada una de las actividades que desarrollaba el equipo de trabajo se tenía un diagrama de Gantt donde se tenía el tiempo estimado para el desarrollo para la petición del cliente, por otra parte para que se cumpliera con la estimación de tiempo o se pudiera recalcular el desarrollador diariamente escribía cuantas horas dedicó a la actividad y cuál fue su avance.

  Con las actividades asignadas a cada desarrollador se hacían entregables a revisar por el PO, conforme el sistema fue creciendo el equipo de desarrollo también fue incrementándose al ser más desarrolladores se solicitaba más tiempo al PO para pruebas a la par del desarrollo, como lo propone la metodología, por lo que fue necesario dividir el equipo de desarrollo para resolver las incidencias y evitar que en el equipo se presentaran retrasos al desarrollar, por un lado se contaba con un equipo especial quien mejoraba las áreas de oportunidad que nos proponía el usuario y el resto del equipo de desarrollo continuó con el plan de trabajo.

  Para este proyecto se migró casi por completo el Sistema de Casa de Bolsa (SCB), debía haber comunicación entre los dos sistemas, esta comunicación se llevaba a cabo mediante Web Services, con esto no se comprometía el código para ninguno de los 2 sistemas. Finalmente el propósito de un Web Service es brindar un servicio y dar respuesta del servicio brindado.

  La base de datos, heredada del Sistema de Casa de Bolsa (SCB) sistema que no conocían todos los desarrolladores por lo que se desconocían las tablas que se tenían en el SCB, esto trajo muchos problemas cuando se migro la base de datos al SI, pues no sabíamos los datos que eran de SCB, tampoco teníamos conocimiento sobre cuál era el uso de cada una de las tablas que se migraron. Como parte del desarrollo el PO tomo la iniciativa de hacer un diseño de base de datos para el nuevo sistema, para poder tener un control de las tablas que realmente eran usadas y así no ocupar un mayor espacio, de memoria en el servidor de base de datos, que el requerido para el sistema.

En la figura 39 se muestra parte del diagrama Entidad - Relación de la base de datos del Sistema de Inversiones

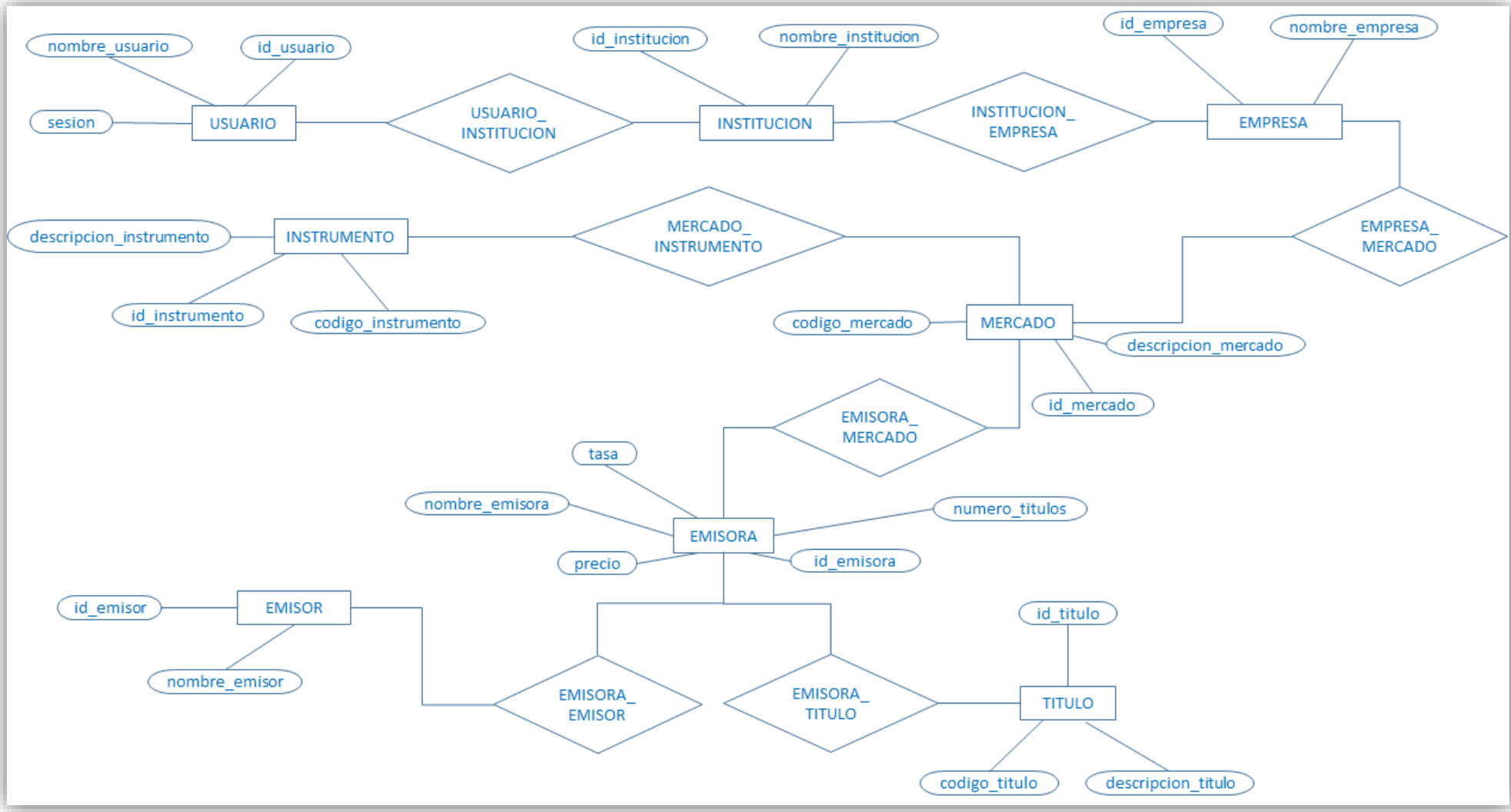


Figura 39. Diagrama Entidad - Relación del Sistema de Inversiones

Descripción de Entidades:

- USUARIO: Es un catálogo que contiene la información sobre los usuarios que podrán usar el sistema.

- INSTITUCION: Es un catálogo que contiene la información sobre la Institución Inversora

- EMPRESA: Es un catálogo que contiene la información sobre las empresas que pueden participar en la compra / venta productos (Emisoras, Forwards SWAP, Futuros, etc.).

  INSTRUMENTO: Es un catálogo que contiene el tipo de Instrumento Financiero con el que se pacta la Inversión.

- EMISOR: Es un catálogo que contiene la información sobre las empresas que tienen en venta emisoras.

- EMISORA: Contiene la información de las emisoras que tiene disponible para venta de títulos.

- MERCADO: Es un catálogo que contiene la información sobre el tipo de mercado en que se pacta la Inversión

- TITULO: Es un catálogo que contiene la información sobre los documentos comprados a la empresa, como son el tipo de título, descripción, etc.

Diccionario de Datos

Notación del diccionario de datos.

= Está compuesto de.

+ Y.

() Optativo (puede o no estar presente).

[] Seleccionar una de varias alternativas.

** Comentarios.

@ Identificador (campo llave).

| Separa opciones alternativas.

Diccionario de Datos	
Campo	Descripción
CODIGO_INSTRUMENTO	Código con el que se reconocerá el Instrumento Financiero en el sistema. Campo de tipo carácter con longitud de 3.
CODIGO_MERCADO	Código con el que se reconocerá el tipo de mercado en el que se invertirá dentro del sistema. Campo de tipo carácter con longitud de 3
CODIGO_TITULO	Código con el que se reconocerá el tipo de título que tiene la emisora en compra / venta. Campo de tipo carácter con longitud de 3.
DESCRIPCION_INSTRUMENTO	Descripción del tipo de Instrumento Financiero con el que se pacta la inversión. Campo de tipo carácter con longitud de 50
DESCRIPCION_MERCADO	Descripción del tipo de mercado en el que se invertirá. Campo de tipo carácter con longitud de 50.
DESCRIPCION_TITULO	Descripción del tipo de título. Campo de tipo carácter con longitud de 50
EMISOR	Archivo que contiene la información sobre las empresas que tienen en venta emisoras
EMISORA	Archivo que contiene la información sobre las emisoras que se encuentran en venta @id_emisora
EMPRESA	Archivo que contiene la información sobre la empresa que puede realizar compra / venta de productos @id_empresa
ID_EMISOR	Identificador del emisor. Campo de tipo carácter con longitud de 6

Diccionario de Datos	
Campo	Descripción
ID_EMISORA	Identificador de la emisora que se encuentra en venta. Campo de tipo carácter con longitud de 9
ID_EMPRESA	Identificador de la empresa puede realizar compra / venta de productos. Campo de tipo carácter con longitud 6
ID_INSTITUCION	Identificador de la Institución Inversora. Campo de tipo carácter con longitud de 6
ID_INSTRUMENTO	Identificador del Instrumento Financiero con que se pacta la inversión. Campo de tipo carácter con longitud de 6
ID_MERCADO	Identificador del tipo de mercado en el que se invertirá. Campo de tipo carácter con longitud de 6.
ID_TITULO	Identificador del tipo de título. Campo de tipo carácter con longitud de 6
ID_USUARIO	Identificador del usuario del sistema, proporcionado por la Institución Financiera. Campo de tipo carácter con longitud de 6
INSTITUCION	Archivo que contiene la información sobre la Institución Inversosa. @id_institucion
INSTRUMENTO	Archivo que contiene la información sobre el tipo de Instrumento Financiero con el que se pacta la inversión @id_instrumento
MERCADO	Archivo que contiene la información sobre el tipo de mercado en el que se invertirá @id_mercado
NOMBRE_EMISOR	Nombre del emisor. Campo de tipo carácter con longitud de 50
NOMBRE_EMISORA	Nombre de la emisora que se encuentra en venta. Campo de tipo carácter con longitud de 50
NOMBRE_EMPRESA	Nombre de la empresa que puede realizar compra / venta de productos. Campo de tipo carácter con longitud de 50
NOMBRE_INSTITUCION	Nombre de la institución Inversora. Campo de tipo carácter con longitud de 50
NOMBRE_USUARIO	Nombre del usuario del sistema. Campo de tipo carácter con longitud de 50
NUMERO_TITULOS	Número de títulos que posee la emisora que se encuentra en venta. Campo de tipo numérico con longitud de 9
PRECIO	Precio que posee la emisora que se encuentra en venta. Campo de tipo numérico con longitud de 9

Diccionario de Datos	
Campo	Descripción
SESION	Fecha de último acceso del usuario. Campo de tipo Fecha
TASA	Tasa con la que se pactó la venta de la emisora. Campo de tipo numérico con longitud de 9
TITULO	Archivo que contiene la información sobre el tipo de títulos que posee la emisora que se encuentra en compra / venta
USUARIO	Archivo que contiene la información sobre los usuario del sistema @id_usuario

A continuación se describen las entidades de acuerdo a los datos que se encuentran el diccionario de datos.

Nomenclatura.

> Llave primaria

E = Entero

N = Numérico

D = Decimal

C = Carácter

L = Lógico

DATE = Tipo fecha (longitud de 8 caracteres)

NOMBRE DE LA TABLA:	EMISOR	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMISOR	C6	ID_EMISOR
NOMBRE_EMISOR	C50	NOMBRE_EMISOR

NOMBRE DE LA TABLA:	EMISORA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMISORA	C6	ID_EMISORA
ID_EMPRESA	C6	ID_EMPRESA
ID_TITULO	C6	ID_TITULO
NOMBRE_EMISORA	C50	NOMBRE_EMISORA
NUMERO_TITULOS	N E9 D0	NUMERO_TITULOS
PRECIO	N E7 D2	PECIO
TASA	N E7 D2	TASA

NOMBRE DE LA TABLA:	EMPRESA	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMPRESA	C6	ID_EMPRESA
NOMBRE_EMPRESA	C50	NOMBRE_EMPRESA

NOMBRE DE LA TABLA:	EMPRESA_MERCADO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_EMPRESA	C6	ID_EMPRESA
>ID_MERCADO	C6	ID_MERCADO

NOMBRE DE LA TABLA:	INSTITUCION	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_INSTITUCION	C6	ID_INSTITUCION
NOMBRE_INSTITUCION	C50	NOMBRE_INSTITUCION

NOMBRE DE LA TABLA:	INSTRUMENTO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_INSTRUMENTO	C6	ID_INSTRUMENTO
DESCRIPCION_INSTRUMENTO	C50	NOMBRE_INSTRUMENTO

NOMBRE DE LA TABLA:	MERCADO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
> ID_MERCADO	C6	ID_MERCADO
CODIGO_MERCADO	C3	CODIGO_MERCADO
DESCRIPCION_MERCADO	50	DESCRIPCION_MERCADO

NOMBRE DE LA TABLA:	MERCADO_INSTRUMENTO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
> ID_MERCADO	C6	ID_MERCADO
>ID_INSTRUMENTO	C6	ID_INSTRUMENTO

NOMBRE DE LA TABLA:	TITULO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
> ID_TITULO	C6	ID_TITULO
CODIGO_TITULO	C3	CODIGO_TITULO
DESCRIPCION_TITULO	50	DESCRIPCION_TITULO

NOMBRE DE LA TABLA:	USUARIO	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>ID_USUARIO	C6	ID_USUARIO
ID_INSTITUCION	C6	ID_INSTITUCION
NOMBRE_USUARIO	C50	NOMBRE_USUARIO
SESION	DATE	SESION

NOMBRE DE LA TABLA:	USUARIO_INSTITUCION	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
> ID_USUARIO	C6	ID_USUARIO
>ID_INSTITUCION	C6	ID_INSTITUCION

A continuación presento algunos diagramas de Gantt (Figuras 40 y 41) que describen el plan de trabajo para el proyecto de Inversiones.

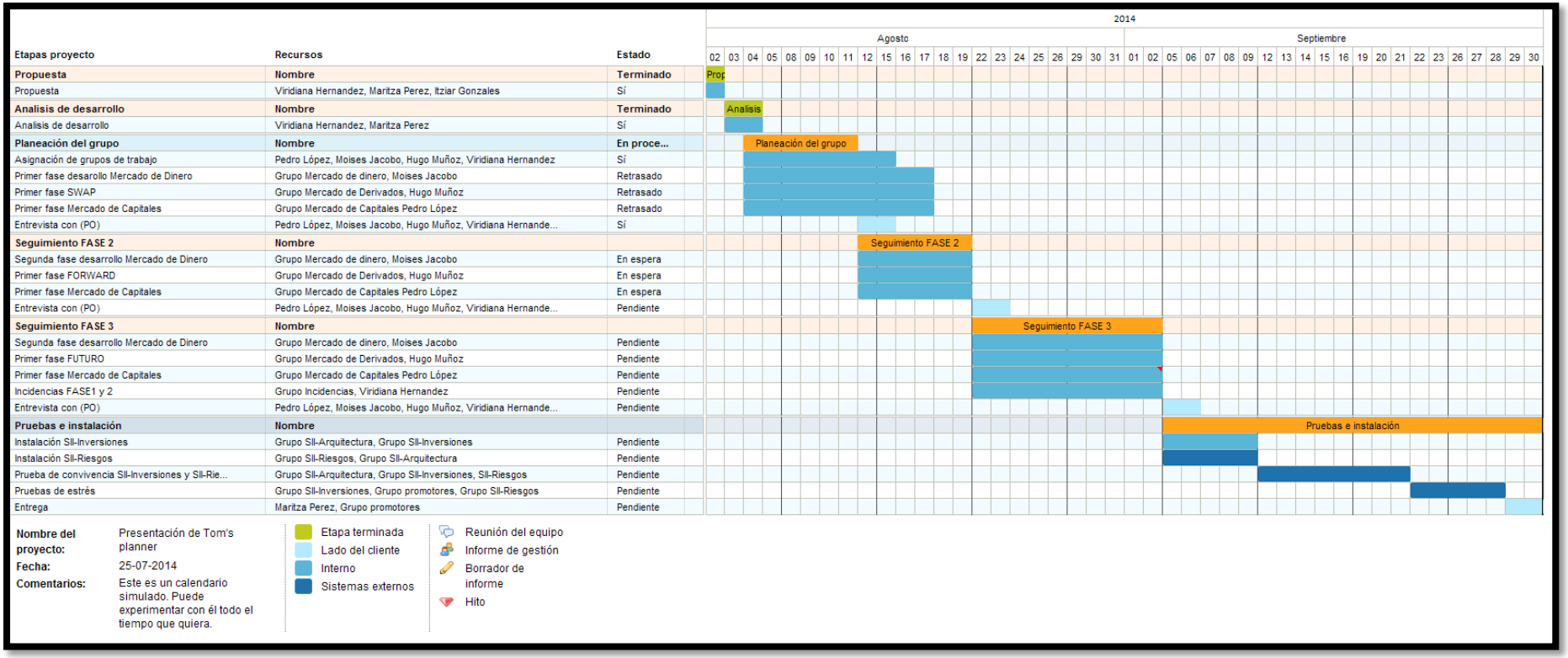


Figura 40. Diagrama Gantt PROYECTO GENERAL

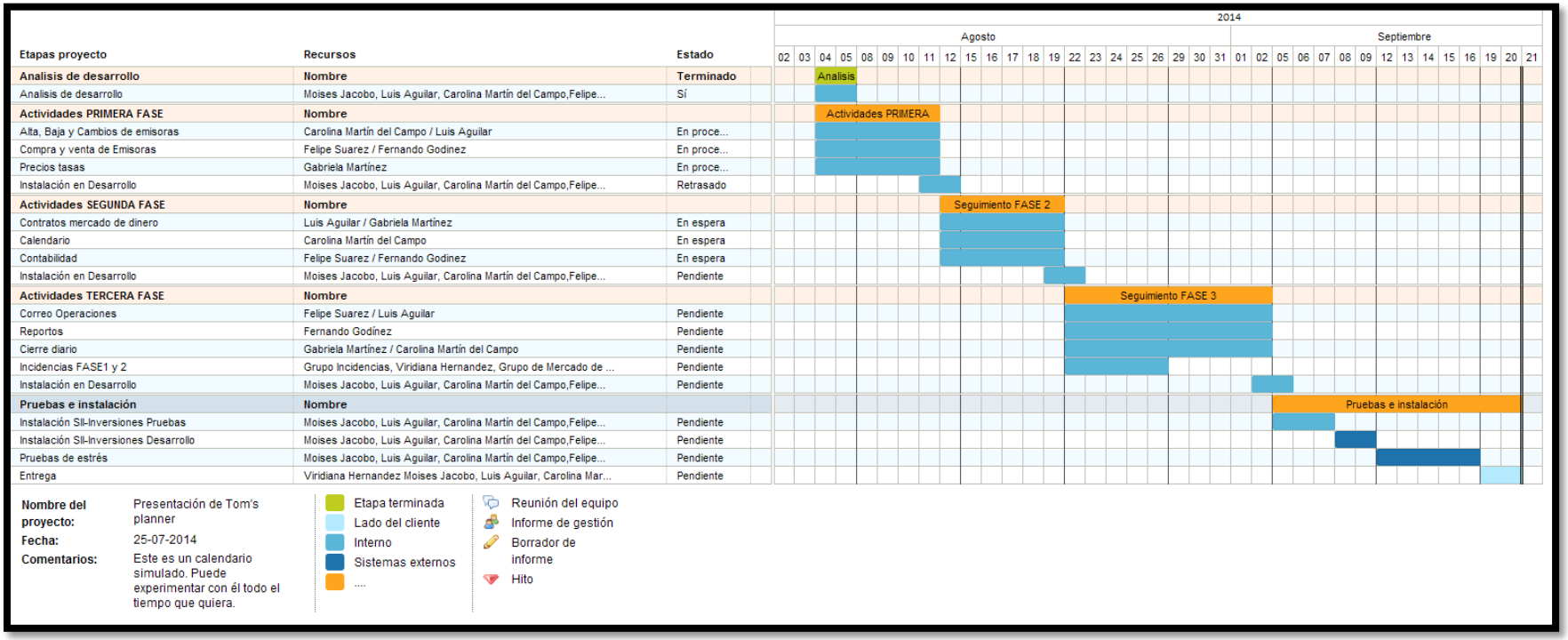


Figura 41. Diagrama Gantt Mercado de Dinero

En las figuras 42 a 46 se muestra un ejemplo de las ventanas que se crearon para el Sistema de Inversiones

Figura 42. Ejemplo de ventana del Sistema de Inversiones

Figura 43. Ejemplo de ventana del Sistema de Inversiones

Figura 44. Ejemplo de ventana del Sistema de Inversiones

Figura 45. Ejemplo de ventana del Sistema de Inversiones

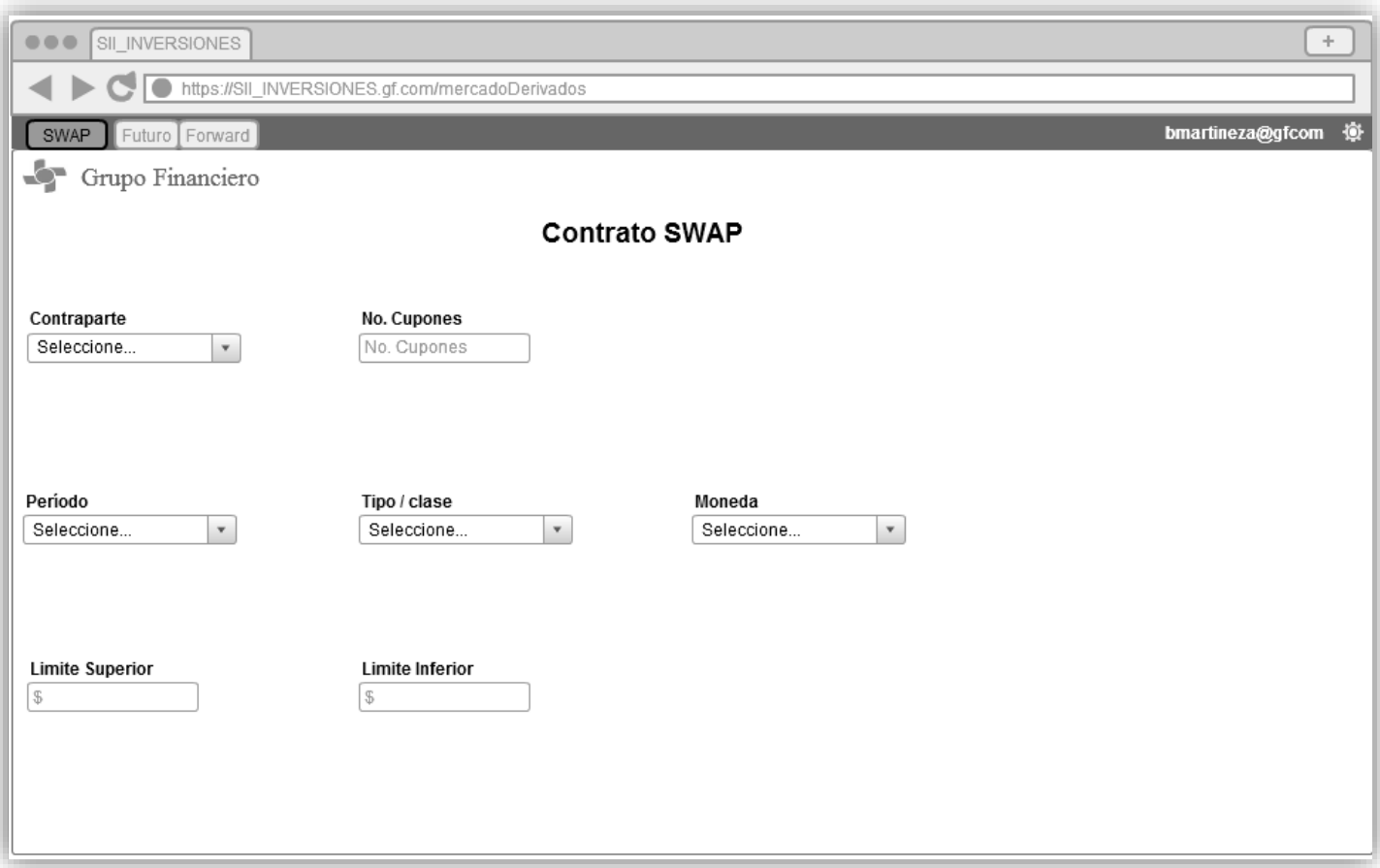


Figura 46. Ejemplo de ventana del Sistema de Inversiones

IMPACTO

Antes de este sistema la Institución Financiera no contaba con un servicio específico de Inversiones sólo existía el SCB, donde podían invertir con renta fija sin ningún riesgo de pérdida de capital. Este sistema llevo a la Institución Financiera a dar un gran salto en ganancias porque para los clientes había otra opción para invertir.

Este proyecto fue liberado sin consecuencias de tiempo, es decir se terminó en tiempo y forma, antes de ser liberado se llevaron sesiones de capacitación para el usuario esto para que el sistema fuera completamente entendido por la parte de promoción de la institución bancaria. La interacción con el SCB no fue compleja pues los desarrolladores de ambos sistemas mantenían comunicación continua para realizar pruebas y desarrollo conjunto.

CONCLUSIONES

Con la metodología ágil que se usó para este proyecto, se pudo mantener un orden en las actividades que cada desarrollador tenía a su cargo, además de controlar los tiempos en que cada tarea se realizaba.

Por otra parte en este tipo de proyectos es importante mantener la comunicación con el equipo de trabajo al que perteneces pues el progreso del proyecto depende de cada una de las actividades que se realizan y es importante que el sistema siempre funcione de la misma manera aun cuando se agregan cambios o funcionalidades.

El que este sistema dependiera de otro muy parecido y que los desarrolladores pertenecieran al SCB hizo que la programación fuera menos compleja y que siempre se tuviera un sistema estable.

El sistema cumplió con los requerimientos solicitados por el usuario, actualmente el sistema es sado en producción por la institución bancaria.

5.3 Tercera Institución

DEL DESARROLLO

Aquí he desarrollado la banca móvil /aplicación móvil de la institución. Este sistema estuvo más complicado que los anteriores, pues a pesar de que se tenía un modelo de ciclo de vida, la integración del equipo de trabajo fue complicada, pues el front de la aplicación fue hecho por un proveedor externo, se pretendía que el proveedor trabajará a la par del equipo de la institución pero no se tenía una coordinación con el proveedor y el equipo de desarrollo de la institución.

Como ciclo de vida se eligió el modelo en cascada, en este proyecto el tiempo destinado para documentación se cumplió de hecho, después se empezó con el desarrollo, por parte de la institución financiera sólo se expusieron los servicios requeridos para la aplicación mediante Web Services que el proveedor consumió para el funcionamiento del sistema. Los Web Services fueron desarrollados en Java publicados en un servidor JBoss.

Antes de las pruebas con el usuario se tenían sesiones de pruebas con el proveedor para verificar la conexión entre los servicios que él proporcionaba con los que la Institución Financiera tenía expuestos. Esto hacia que en la pruebas con el usuario no hubiera fallas de conectividad o de desarrollo (campos nulos o vacíos, errores en índices de arreglos, etc.). Las fallas que se reportaban por parte del usuario eran corregidas para la siguiente sesión de pruebas, y las fallas durante la sesión con el proveedor se solucionaban en el momento esto con el fin de no exceder el tiempo de desarrollo ni de unificación en la etapa de Integración.

Durante la etapa de prueba se hizo una versión “Friends and Family” donde sólo empleados de la Institución Bancaria teníamos acceso a la aplicación con lo que nos permitió tener un panorama más amplio de próximas mejoras en la etapa de mantenimiento.

En cuanto a la base de datos, sólo se usaban datos que ya se tenían para la banca en línea sólo se hicieron las tablas que permitían identificar al usuario de banca en línea, esto fue al usuario que se tenía en Banca en Línea se le asocio un número telefónico con el que se puede hacer uso de la aplicación. Hubo muchos cambios en cuanto a tiempos de entrega por parte del proveedor del servicio de pantallas, pues no se coincidía en pruebas en conjunto con la institución para la que trabajo.

En la figura 47 se muestra la parte esencial del diagrama Entidad – Relación de la base de datos de la Banca Móvil

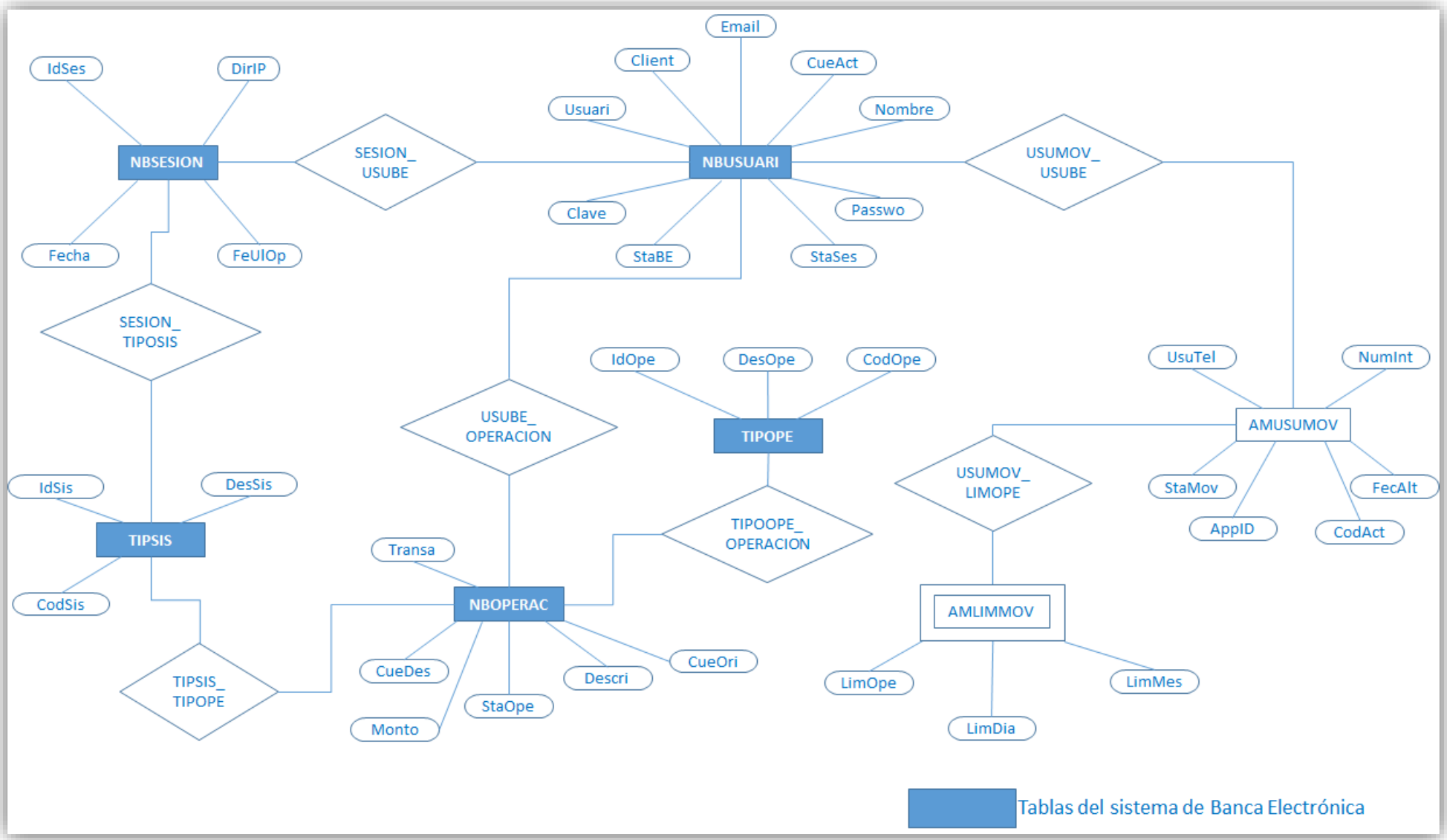


Figura 47. Diagrama Entidad – Relación de la Banca Móvil

Descripción de Entidades:

- NBSESION: Contiene la información sobre si el usuario de Banca Móvil o Banca Electrónica tiene una sesión en alguno de los dos sistemas.

- NBUSUARI: Contiene la información sobre el usuario de Banca Electrónica

- AMUSUMOV: Contiene la información sobre el usuario de Banca Móvil

- NBOPERAC: Contiene la información sobre las operaciones realizadas (SPEI, Transferencias o SPEI móvil)

- AMLIMMOV: Contiene la información sobre los límites de operación que tiene el usuario de Banca Móvil, limites ajustados por el usuario

- TIPSIS: Catalogo que contiene información sobre el tipo de sistema con el que se identifiquen desde donde realiza conexiones u operaciones.

Diccionario de Datos

Notación del diccionario de datos.

= Está compuesto de.

+ Y.

() Optativo (puede o no estar presente).

[] Seleccionar una de varias alternativas.

** Comentarios.

@ Identificador (campo llave).

| Separa opciones alternativas.

Diccionario de Datos	
Campo	Descripción
AMLIMMOV	Archivo que contiene los límites de operación por usuario de Banca Móvil.
AMUSUMOV	Archivo que contiene la información sobre los usuarios de Banca Móvil @UsuTel
APPID	Identificador del teléfono del que se activa el servicio de Banca Móvil. Campo de tipo carácter con longitud de 50
CLAVE	Clave con la que el usuario tiene acceso a la Banca Electrónica. Campo de tipo Carácter con longitud de 50
CLIENT	Numero de cliente dentro de la Institución Financiera. Campo de tipo carácter con longitud de 8
CODACT	Número para activar el servicio de Banca Móvil. Campo de tipo numérico con longitud de 4
CODOPE	Código que se usará dentro del sistema para asignar un tipo de operación. Campo de tipo carácter con longitud de 3
CODSIS	Código con el que se identificara a cada sistema. Campo de tipo carácter con longitud de 3
CUEACT	Número de cuenta activa del cliente de la Institución Financiera. Campo de tipo carácter con longitud de 12
CUEDES	Cuenta destino de la operación. Campo de tipo carácter con longitud de 18

Diccionario de Datos	
Campo	Descripción
CUEORI	Cuenta origen de la operación. Campo de tipo carácter con longitud de 12
DESCRI	Descripción de la operación. Campo de tipo carácter con longitud de 50
DESOPE	Descripción del tipo de operación realizada o por realizar. Campo de tipo carácter con longitud de 50
DESSIS	Descripción del sistema. Campo de tipo carácter con longitud de 50
DIRIP	Dirección IP desde la que se conecta el usuario a Banca Móvil o Banca Electrónica. Campo de tipo carácter con longitud de 15
EMAIL	Correo electrónico del usuario de Banca Electrónica. Campo de tipo carácter con longitud de 20
FECALT	Fecha de alta del servicio de Banca Móvil. Campo de tipo Fecha
FECHA	Fecha en la que se inició la sesión. Campo de tipo Fecha
FEULOP	Fecha en la que se realizó la última operación en Banca Móvil o Banca Electrónica. Campo de tipo fecha
IDOPE	Identificador del tipo de operación realizada o por realizar. Campo de tipo carácter con longitud de 6.
IDSES	Identificador de la sesión. Campo de tipo carácter con longitud de 9
IDSIS	Identificador del sistema. Campo de tipo carácter con longitud de 6
LIMDIA	Monto limite por día para realizar operaciones. Campo de tipo numérico con longitud de 9
LIMMES	Monto limite por mes para realizar operaciones. Campo de tipo numérico con longitud de 9
LIMOPE	Monto limite por operación realizada. Campo de tipo numérico con longitud de 9
MONTO	Monto de la operación realizada o por realizar. Campo de tipo de numérico con longitud de 9
NBOPERAC	Archivo que contiene la información sobre las operaciones que ha realizado o va a realizar el usuario en Banca Móvil o Banca Electrónica @Transa

Diccionario de Datos	
Campo	Descripción
NBSESION	Archivo que contiene los registros de sesiones activas en Banca Móvil o Banca Electrónica @IdSes
NBUSUARI	Archivo que contiene la información del usuario de Banca Electrónica @Usuari
NOMBRE	Nombre del cliente de la Institución Financiera. Campo de tipo carácter con longitud de 50
NUMINT	Número de intentos erróneos para ingresar a la Banca Móvil. Campo de tipo numérico con longitud de 1
PASSWO	Contraseña del usuario en Banca Electrónica y Banca Móvil. Campo de tipo carácter con longitud de 50
STABE	Estatus del usuario en Banca electrónica, activo, inactivo, bloqueado, cancelado [A, I, B, C]. Campo de tipo carácter con longitud de 1
STAMOV	Estatus del servicio de Banca Móvil, activo, inactivo, pendiente, cancelado, bloqueado [A, I, P, C, B]
STAOPE	Estatus de la operación, registrada, enviada, procesando, cancelada [R, E, P, C]. Campo de tipo carácter con longitud de 1
STASES	Estatus de sesión, activa o inactiva [A, I]. Campo de tipo carácter con longitud de 1
TIPOOPE	Archivo que contiene el tipo de operación realizada o por realizar del usuario en Banca Móvil o Banca Electrónica @IdOpe
TIPSIS	Archivo que contiene la información sobre el tipo del sistema del que se hace la conexión o alguna operación por parte del usuario @IdSis
TRANSA	Número de transacción, se genera con un número consecutivo a través de la base de datos. Campo de tipo entero con longitud de 9
USUARI	Numero consecutivo dado por el sistema de Banca Electrónica. Campo de tipo carácter con longitud de 6
USUTEL	Número telefónico con el que se tiene activada la Banca Móvil. Campo de tipo carácter con longitud de 10

A continuación se describen las entidades de acuerdo a los datos que se encuentran el diccionario de datos.

Nomenclatura.

> Llave primaria

E = Entero

N = Numérico

D = Decimal

C = Carácter

L = Lógico

DATE = Tipo fecha (longitud de 8 caracteres)

NOMBRE DE LA TABLA:	AMLIMMOV	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>USUTEL	C10	USUTEL
LIMDIA	N E7 D2	LIMDIA
LIMMES	N E7 D2	LIMMES
LIMOPE	N E7 D2	LIMOPE

NOMBRE DE LA TABLA:	AMUSUMOV	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>USUTEL	C10	USUTEL
APPID	C50	APPID
CODACT	C4	CODACT
NUMINT	N E1	NUMINT
STAMOV	C1	STAMOV
USUARI	C6	USUARI

NOMBRE DE LA TABLA:	NBOPERAC	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>TRANSA	C9	TRANSA
CUEORIGEN	C12	CUEORI
DESCRI	C50	DESCRI
MONTO	N E7 D2	MONTO
STAOPE	C1	STAOPE
CUEDES	C18	CUEDES
IDSIS	C6	IDSIS
USUARI	C6	USUARI
IDOPE	C6	IDOPE

NOMBRE DE LA TABLA:	NBSESION	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>IDSESION	C9	IDSESION
DIRIP	C15	DIRIP
FECHA	DATE	FECHA
FEULOP	DATE	FEULOP

NOMBRE DE LA TABLA:	NBUSUARI	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>USUARI	C6	USUARI
CLAVE	C50	CLAVE
CLIENT	C8	CLIENT
CUEACT	C12	CUEACT
EMAIL	C20	EMAIL
NOMBRE	C50	NOMBRE
PASSWO	C50	PASSWO
STABE	C1	STABE
STASES	C1	STASES

NOMBRE DE LA TABLA:	TIPOPE	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>IDOPE	C9	IDOPE
CODOPE	C3	CODOPE
DESOPE	C50	DESOPE

NOMBRE DE LA TABLA:	TIPSIS	
NOMBRE EN LA TABLA	TIPO DE DATO	NOMBRE EN DICCIONARIO DE DATOS
>IDSIS	C9	IDSIS
CODSIS	C3	CODSIS
DESSIS	C50	DESSIS

A continuación, en la figura 48, se muestra el diagrama general del proyecto:

Figura 39. Diagrama general del Proyecto de Aplicación Móvil

Por parte del equipo de desarrollo de la institución para la que trabajo el diagrama del proyecto se encuentra de la siguiente manera (figura 49)

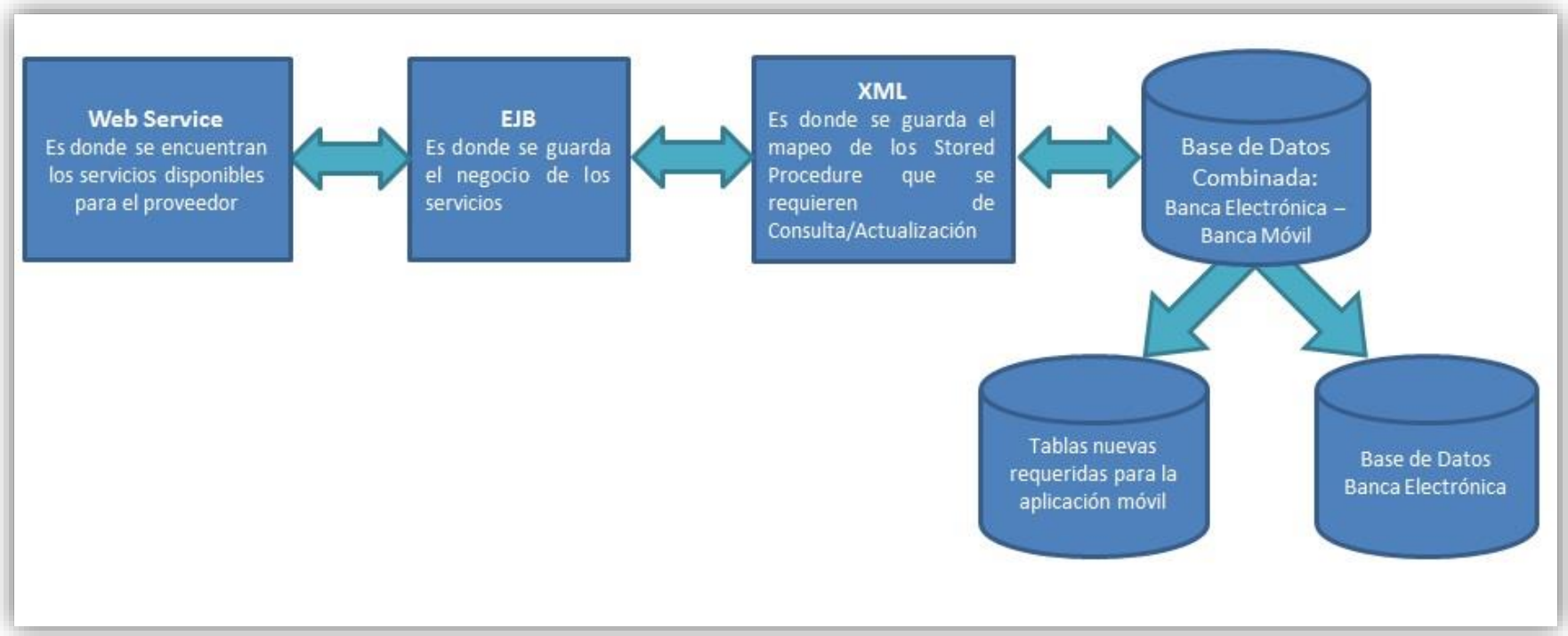


Figura 49. Diagrama de la parte de desarrollo en Java

Diagrama del ciclo de vida del proyecto de aplicación móvil (Figura 50)

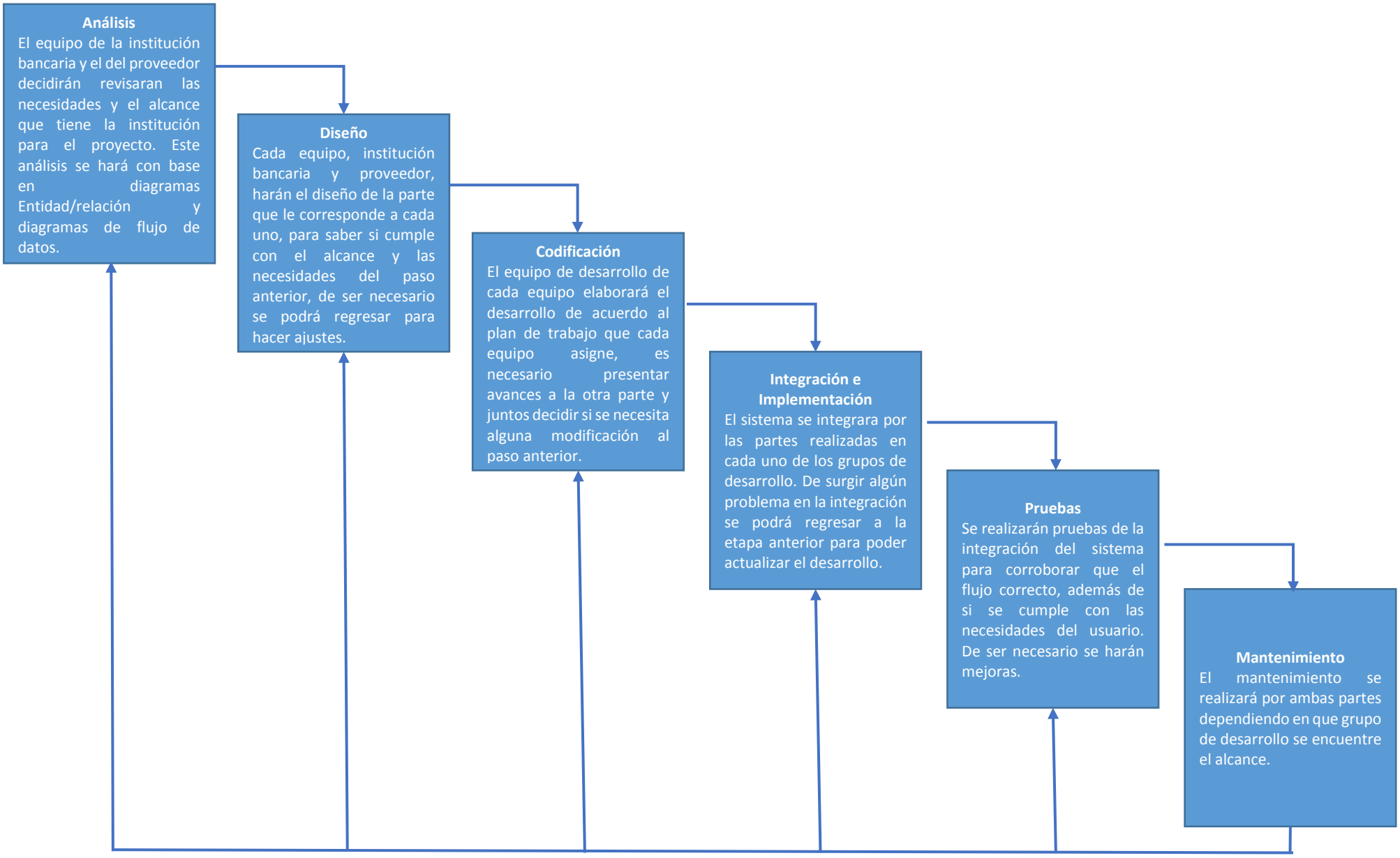
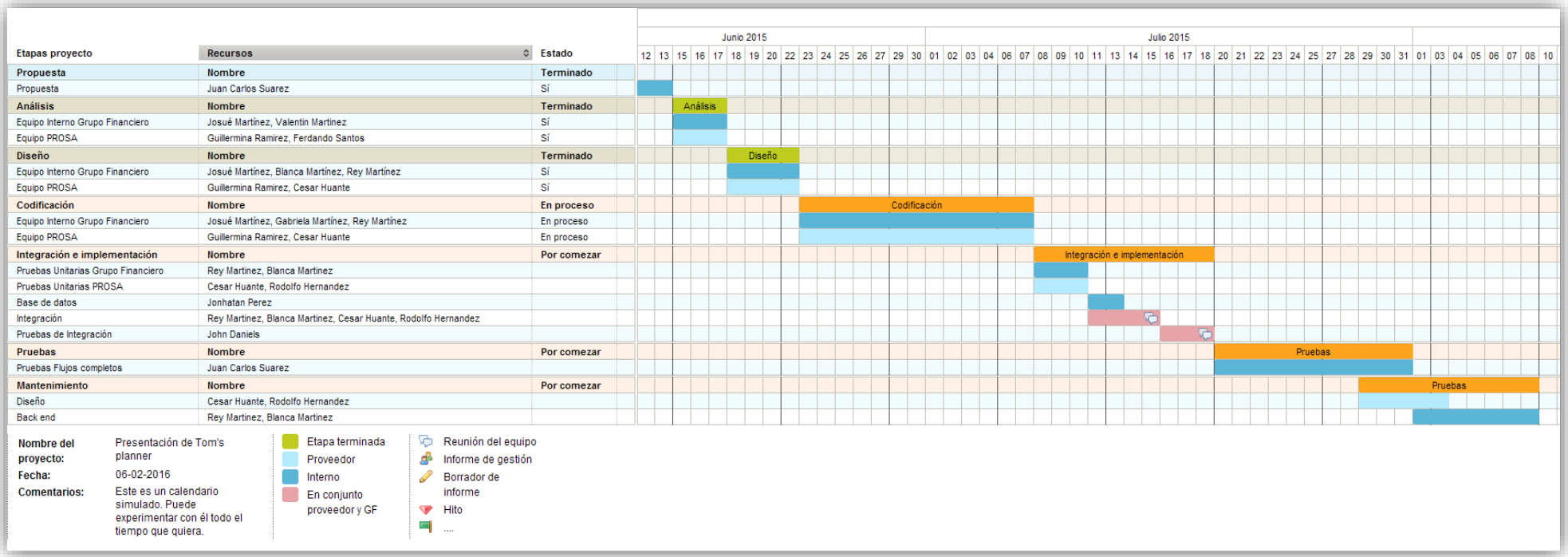


Figura 50. Ciclo de vida en Cascada

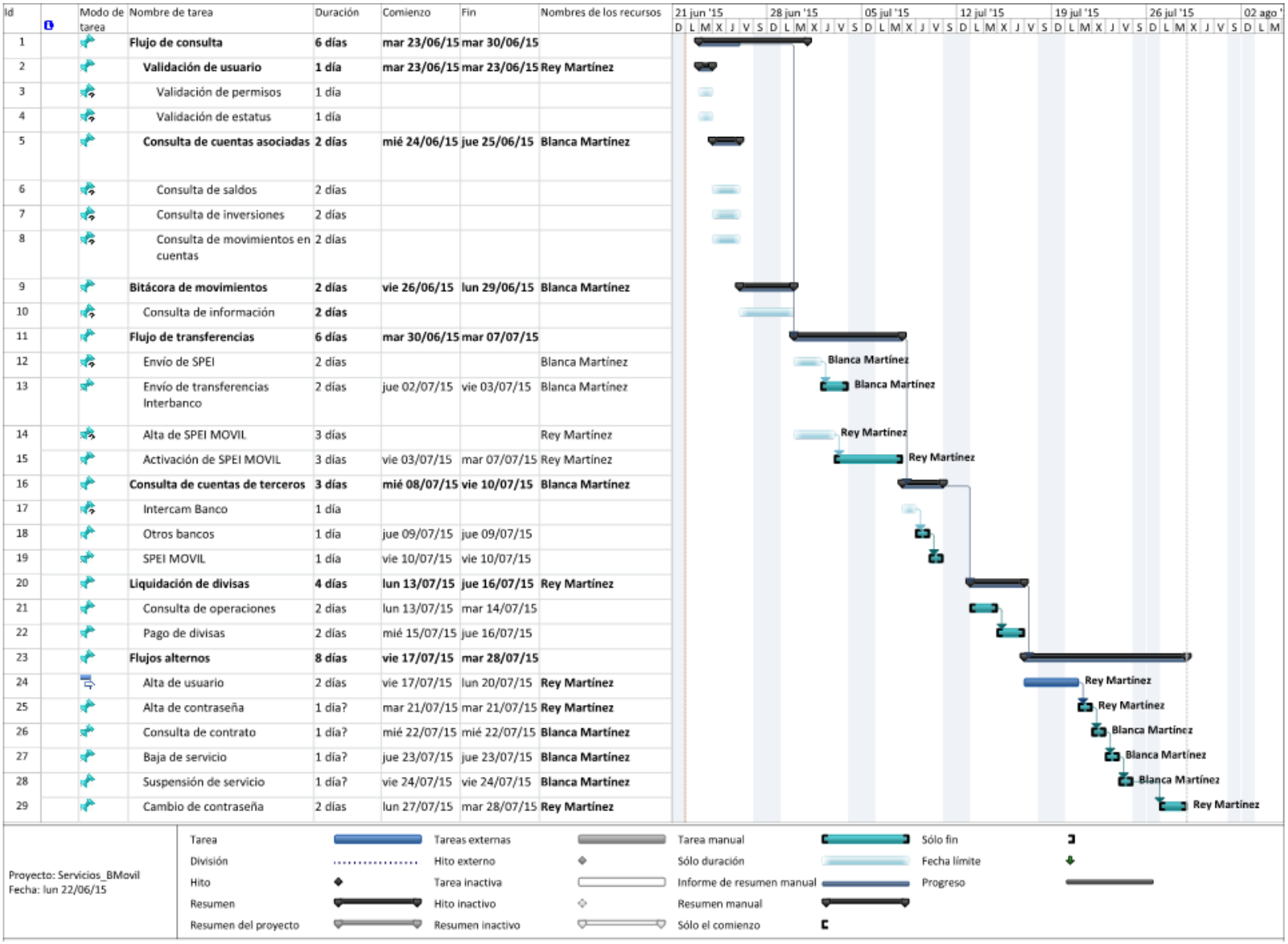
La Figura 51 muestra el diagrama de Gantt por etapas del ciclo de vida presentado en la figura anterior.



Etapas proyecto	Recursos	Estado
Propuesta	**Nombre**	**Terminado**
Propuesta	Juan Carlos Suarez	Sí
Análisis	**Nombre**	**Terminado**
Equipo Interno Grupo Financiero	Josué Martínez, Valentin Martinez	Sí
Equipo PROSA	Guillermina Ramirez, Ferdando Santos	Sí
Diseño	**Nombre**	**Terminado**
Equipo Interno Grupo Financiero	Josué Martínez, Blanca Martínez, Rey Martínez	Sí
Equipo PROSA	Guillermina Ramirez, Cesar Huante	Sí
Codificación	**Nombre**	**En proceso**
Equipo Interno Grupo Financiero	Josué Martínez, Gabriela Martínez, Rey Martínez	En proceso
Equipo PROSA	Guillermina Ramirez, Cesar Huante	En proceso
Integración e implementación	**Nombre**	**Por comezar**
Pruebas Unitarias Grupo Financiero	Rey Martinez, Blanca Martinez	
Pruebas Unitarias PROSA	Cesar Huante, Rodolfo Hernandez	
Base de datos	Jonhatan Perez	
Integración	Rey Martinez, Blanca Martinez, Cesar Huante, Rodolfo Hernandez	
Pruebas de Integración	John Daniels	
Pruebas	**Nombre**	**Por comezar**
Pruebas Flujos completos	Juan Carlos Suarez	
Mantenimiento	**Nombre**	**Por comezar**
Diseño	Cesar Huante, Rodolfo Hernandez	
Back end	Rey Martinez, Blanca Martinez	

Figura 51. Diagrama de Gantt, ciclo de vida

Plan de trabajo (Figura 52) por parte del desarrollo interno de la Institución financiera (Etapa de Codificación).



Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
1		**Flujo de consulta**	**6 días**	**mar 23/06/15**	**mar 30/06/15**	
2		**Validación de usuario**	**1 día**	**mar 23/06/15**	**mar 23/06/15**	**Rey Martínez**
3		Validación de permisos	1 día			
4		Validación de estatus	1 día			
5		**Consulta de cuentas asociadas**	**2 días**	**mié 24/06/15**	**jue 25/06/15**	**Blanca Martínez**
6		Consulta de saldos	2 días			
7		Consulta de inversiones	2 días			
8		Consulta de movimientos en cuentas	2 días			
9		**Bitácora de movimientos**	**2 días**	**vie 26/06/15**	**lun 29/06/15**	**Blanca Martínez**
10		Consulta de información	**2 días**			
11		**Flujo de transferencias**	**6 días**	**mar 30/06/15**	**mar 07/07/15**	
12		Envío de SPEI	2 días			Blanca Martínez
13		Envío de transferencias Interbanco	2 días	jue 02/07/15	vie 03/07/15	Blanca Martínez
14		Alta de SPEI MOVIL	3 días			Rey Martínez
15		Activación de SPEI MOVIL	3 días	vie 03/07/15	mar 07/07/15	Rey Martínez
16		**Consulta de cuentas de terceros**	**3 días**	**mié 08/07/15**	**vie 10/07/15**	**Blanca Martínez**
17		Intercam Banco	1 día			
18		Otros bancos	1 día	jue 09/07/15	jue 09/07/15	
19		SPEI MOVIL	1 día	vie 10/07/15	vie 10/07/15	
20		**Liquidación de divisas**	**4 días**	**lun 13/07/15**	**jue 16/07/15**	**Rey Martínez**
21		Consulta de operaciones	2 días	lun 13/07/15	mar 14/07/15	
22		Pago de divisas	2 días	mié 15/07/15	jue 16/07/15	
23		**Flujos alternos**	**8 días**	**vie 17/07/15**	**mar 28/07/15**	
24		Alta de usuario	2 días	vie 17/07/15	lun 20/07/15	**Rey Martínez**
25		Alta de contraseña	1 día?	mar 21/07/15	mar 21/07/15	**Rey Martínez**
26		Consulta de contrato	1 día?	mié 22/07/15	mié 22/07/15	**Blanca Martínez**
27		Baja de servicio	1 día?	jue 23/07/15	jue 23/07/15	**Blanca Martínez**
28		Suspensión de servicio	1 día?	vie 24/07/15	vie 24/07/15	**Blanca Martínez**
29		Cambio de contraseña	2 días	lun 27/07/15	mar 28/07/15	**Rey Martínez**

Figura 52. Plan de Trabajo

En la Figuras 53 y 54 se muestran algunas pantallas de la banca móvil

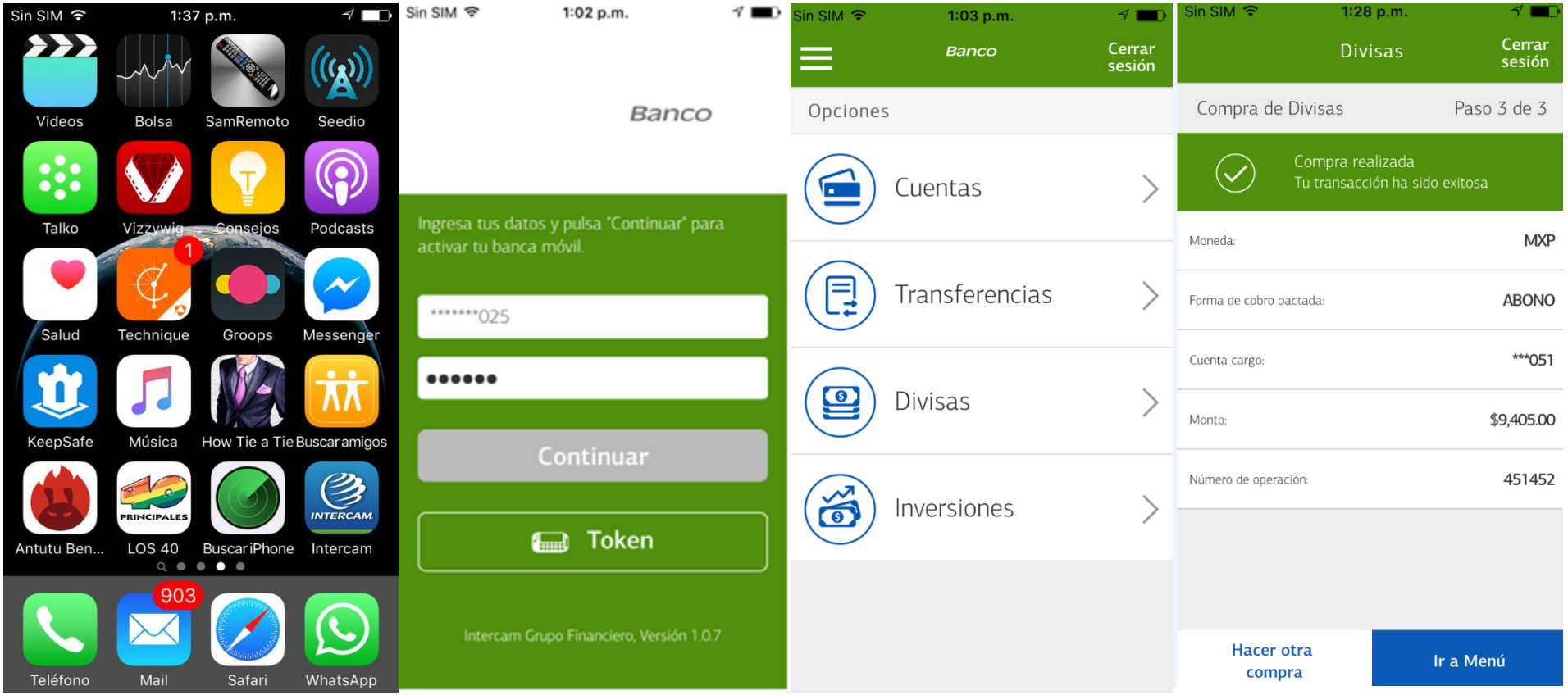


Figura 53. Ejemplos de pantallas de la aplicación móvil

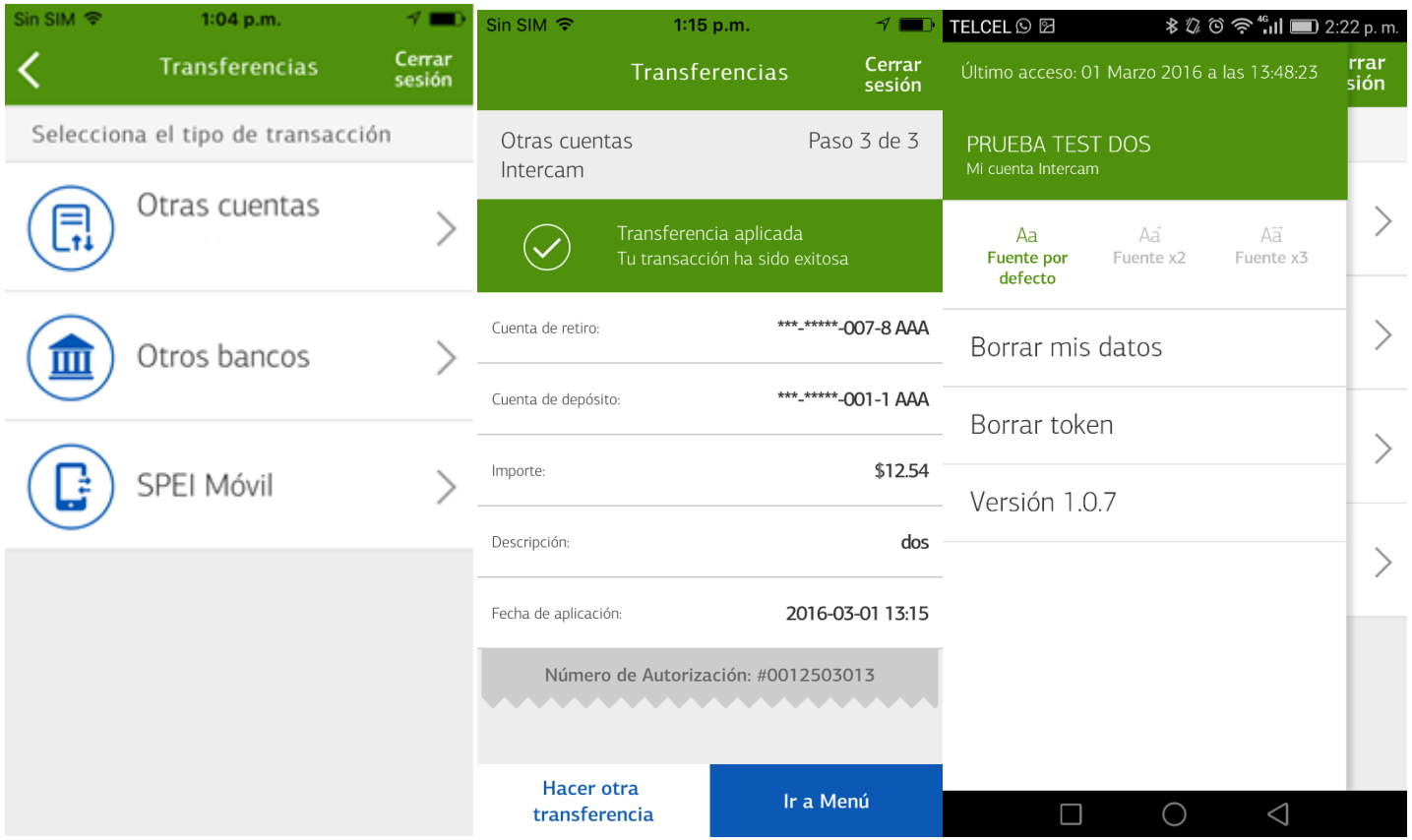


Figura 54. Ejemplos de pantallas de la aplicación móvil

IMPACTO

Antes del desarrollo de esta aplicación no se tenía alguna aplicación que permitiera al cliente tener el control de tu cuenta en un solo clic y desde el sito donde se encuentre, por lo que la aplicación incremento la portabilidad de sus servicios pues sus clientes lo pueden tener todo en la palma de sus manos, además de que hace más eficiente el manejo de las operaciones bancarias, al poder hacerlo desde cualquier lugar en el que te encuentres, con la seguridad que siempre debe brindar una Institución Financiera.

La aplicación cumplió el objetivo que se tenía planteado para la institución financiera, que era estar presentes en medio digitales para poder ser una institución competitiva, además se cumplió con los requerimientos solicitados por parte del área de marketing. Actualmente la aplicación se encuentra funcionando para los clientes que son personas físicas.

CONCLUSIONES

El ciclo de vida permitió que el desarrollo se realizará de forma ordenada, pues se tenía una especificación sobre el tiempo en que se debía llevar el proyecto, además así se podía interactuar con el proveedor para poder unir el trabajo.

Al tener pruebas con el proveedor y con el usuario hacia que el proyecto avanzara sin mayores complicaciones y se podían atender todas las mejoras a tiempo, antes de que el desarrollo llegara a producción y fuera usado por los clientes de la Institución Financiera.

El área de call center tuvo capacitación por parte del equipo de desarrollo para también dar apoyo vía telefónica. Este proyecto ha sido en gran medida un escalón más de crecimiento pues es la primera vez que colaboro en una aplicación móvil.

6. Conclusiones

Se establecen las siguientes conclusiones del presente:

- Se cumplen con los objetivos tratados en un inicio y servirá el presente como herramienta para estudiantes y futuros egresados de la universidad pues se muestra un marco general de cómo se manejan algunos proyectos en la vida laboral.

- También es importante, para mí, que en el aula no sólo aprendamos lo que los profesores enseñan, es decir no sólo seguir el temario para acreditar una materia, sino saber el trasfondo de las cosas, ser más autodidactas y aprender muchos más que el contenido de una materia.

- Durante mi vida como estudiante aprendí mucho sobre maestros que nos daban no sólo una materia guiada por un temario, sino profesores que nos daban enseñanzas de vida y creo que en la vida laboral es muy útil hacer uso de esas enseñanzas para saber diferenciar de un buen proyecto dirigido con bases sólidas como los son las metodologías ágiles, el buen uso y manejo de bases datos.

Hablando respecto a cada punto que se toca en el presente podemos concluir:

- Para llevar un buen desarrollo de *software* es necesario considerar las nuevas tecnologías que nos facilitan el trabajo como desarrollador y además involucran mucho más al cliente finalmente es parte esencial del proyecto.

- En la vida laboral es muy común encontrar a los cliente fuera del desarrollo del sistema y verlo hasta el final con infinidad de cambios, ya sea necesidades que no fueron consideradas por el mismo cliente, nuevas necesidades, o bien malos entendidos durante la toma de requerimientos por parte nuestra. Es nuestro deber como programadores, diseñadores o, incluso, líderes de proyecto tener claras cada una de las peticiones hechas por el usuario así como incluirlo en durante todo el proyecto.

- Como parte de un buen trabajo de *software* es importante tomar en cuenta los riesgos que tenemos al desarrollar el proyecto, como son las enfermedades de algún recurso, el tiempo que el cliente pueda tomarse para sostener reuniones con el equipo de desarrollo, pero también considero que entre mayor sea nuestro acercamiento a un plan de trabajo podemos definir el camino hacia el que vamos y así saber que todos vamos hacia el mismo punto.

- Se debe considerar que la buena organización del equipo de trabajo influye considerablemente en el proyecto, pues el delegar tareas hace que el trabajo sea uniforme y se tenga menos tiempos muertos y la presión sobre los recursos no sea demasiada.

- La base de datos es una buena forma de organización de datos, pero también se debe considerar importante su diseño para que el sistema no se vea afectado por errores cometidos en la misma, también es muy difícil lograr un buen diseño pues en la industria contamos con tiempos muy reducidos para el diseño de la misma.

- Cabe destacar que el uso de la buenas costumbres harían nuestro trabajo más sencillo, no hay que olvidar en cada proyecto por lo menos llevar un plan de trabajo con estimaciones reales en tiempo de desarrollo, planificación, etc; y de alguna manera tratar de llevarlo a cabo y sino hacer los ajustes pertinentes tanto para el cliente como para el equipo de trabajo.

Tomando en cuenta mi experiencia laboral aplicada al presente puedo concluir:

- La experiencia sobre los proyecto no te lo da más que la práctica, es decir que sólo hasta que estas dentro de un proyecto te puedes dar cuenta que tan fácil y/o difícil llevar a cabo la teoría impartida en un salón de clases, pero que si es tan importante como te dijeron en esos momentos.

- También debemos considerar que lo aprendido en el aula no es suficiente para poder desempeñarnos completamente en la industria, esta carrera, Ingeniería en Computación, tiene cambios muy rápidos pues va de la mano con el avance de la tecnología, por lo que es necesario aprender por mérito propio las nuevas tecnologías, así como mantenernos en constante actualización para poder emplear estas nuevas tecnologías en nuestro empleo.

- Considero importante el aprender a trabajar en equipo para que la carga de trabajo sea uniforme, hablar claramente sobre el tiempo que nos lleva el desarrollo que sean tiempos considerables incluyendo pruebas unitarias y globales.

7. Bibliografía y mesografía

Ahmad, M. O., Markkula, J., & Oivo, M. *KANBAN IN SOFTWARE DEVELOPMENT: A SYSTEMATIC LITERATURE REVIEW. 39TH EUROMICRO CONFERENCE ON ENGINEERING AND ADVANCED APPLICATIONS (SEAA).* IEEE, 2013.

Alfonzo, P., Mariño, S., & Godoy, M. *PROPUESTA METODOLÓGICA PARA LA GESTIÓN DE PROYECTO DE SOFTWARE ÁGIL BASADO EN LA WEB.* Multiciencias, 2011.

Amine, M., & Ahmed-Nacer, M. *AN AGILE METHODOLOGY FOR IMPLEMENTING KNOWLEDGE MANAGEMENT SYSTEMS: A CASE STUDY IN COMPONENT-BASED SOFTWARE ENGINEERING.* International Journal of Software Engineering and Its Applications, 2011.

Anderson, D. J., & Roock, A. *AN AGILE EVOLUTION: WHY KANBAN IS CATCHING ON IN GERMANY AND AROUND THE WORLD.* CUTTER IT JOURNAL, 2011.

Bohem, B. *A SPIRAL MODEL OF SOFTWARE DEVELOPMENT AND ENHANCEMENT.* (I. C. Press, Ed.) Computer, Mayo 1988.

Bozheva de Berriprocess, Teodora. KANBAN: 6 PRÁCTICAS PARA AUMENTAR LA EFICIENCIA EN PROYECTOS TIC. *Dyna* 88, nº 5 (2013): 490-495.

Crnkovic, Ivica. COMPONENT-BASED SOFTWARE ENGINEERING — NEW CHALLENGES IN SOFTWARE DEVELOPMENT. *Software focus* (John Wiley & Sons, Ltd.) 2, nº 4 (Winter 2001): 127-133.

Cusumano, Michael, Alan MacCormack, Chris F. Kemerer, y William Crandall. SOFTWARE DEVELOPMENT WORLDWIDE: THE STATE OF THE PRACTICE. *IEEE Software* (IEEE Computer Society), Diciembre 2003: 2-8.

Dybå, Tore, y Torgeir Dingsøyr. EMPIRICAL STUDIES OF AGILE SOFTWARE DEVELOPMENT: A SYSTEMATIC REVIEW. *Information and Software Technology* 50, nº 9-10 (Agosto 2008): 833–859.

Elizo, Esau Alonso. *MANUAL AVANZADO DE ORACLE.* EUA: Anaya Multimedia, 1998.

Erickson, John, Kalle Lyytinen, y Keng Siau. AGILE MODELING, AGILE SOFTWARE DEVELOPMENT, AND EXTREME PROGRAMMING: THE STATE OF RESEARCH. *Journal of database Management* 16, nº 4 (Octubre - Diciembre 2005): 88-100.

Fernandes, João M., y Sónia M. Sousa. PLAYSCRUM - A CARD GAME TO LEARN THE SCRUM AGILE METHOD. *Second International Conference on Games and Virtual Worlds for Serious Applications.* 2010. 53-59.

Herbsleb, James D., y Deependra Moitra. GLOBAL SOFTWARE DEVELOPMENT. *IEEE Software*, Marzo-Abril 2001: 16-20.

Ikonen, Marko. LEADERSHIP IN KANBAN SOFTWARE DEVELOPMENT PROJECTS: A QUASI-CONTROLLED EXPERIMENT. *1st. International Conference on Lean Enterprise Software and Systems (LESS).* Berlin Heidelberg: Springer, 2010. 85-98.

Kumar, Gaurav, y Pradeep Kumar Bhatia. Mpact of Agile Methodology on Software Development Process. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)* 2, nº 4 (2012): 46-50.

Letelier Torres, Patricio (Ed), y Emilio (Ed.) Sánchez López. METODOLOGÍAS ÁGILES EN EL DESARROLLO DE SOFTWARE. *VIII Jornadas de Ingeniería del Software y Bases de Datos.* Alicante: Grupo ISSI, 2003. 1-59.

Lindstrom, Lowell, y Ron Jeffries. EXTREME PROGRAMMING AND AGILE SOFTWARE DEVELOPMENT METHODOLOGIES. *Information systems management* 21, nº 3 (2004): 41-52.

Livermore, Jeffrey A. FACTORS THAT SIGNIFICANTLY IMPACT THE IMPLEMENTATION OF AN AGILE SOFTWARE DEVELOPMENT METHODOLOGY. *Journal of software* (Academy publisher) 3, nº 4 (Abril 2008): 31-36.

Maurer, Frank, y Sebastien Martel. EXTREME PROGRAMMING: RAPID DEVELOPMENT FOR WEB-BASED APPLICATIONS. *IEEE Internet computing* 6, nº 1 (Enero - Febrero 2002): 86-90.

Mendes Calo, Karla, Estévez Elsa Clara, y Pablo Fillottranni. A QUANTITATIVE FRAMEWORK FOR THE EVALUATION OF AGILE METHODOLOGIES. *Journal of Computer Science & Technology* 10, nº 2 (Junio 2010): 68-73.

Orjuela Duarte, Ailin, y Mauricio Rojas. LAS METODOLOGÍAS DE DESARROLLO ÁGIL COMO UNA OPORTUNIDAD PARA LA INGENIERÍA DE SOFTWARE EDUCATIVO. *Revista Avances en Sistemas e Informática* 5, nº 2 (Junio 2008): 159-171.

Paulk, Mark C. AGILE METHODOLOGIES AND PROCESS DISCIPLINE. *Institute for software research*, Octubre 2002: 15-18.

Paulk, Mark. EXTREME PROGRAMMING FROM A CMM PERSPECTIVE. *IEEE Software* 18, nº 6 (2001): 19-26.

Pikkarainen, M., J. Haikara, O. Salo, P. Abrahamsson, y J. Still. THE IMPACT OF AGILE PRACTICES ON COMMUNICATION IN SOFTWARE DEVELOPMENT. *Empirical software engineering* (Springer) 13, nº 3 (Junio 2008): 303-337.

Potter, Neil, y Mary Sakry. IMPLEMENTING SCRUM (AGILE) AND CMMI TOGETHER. *The Process Group-Post newsletter* 16, nº 2 (Marzo 2009): 1-6.

Pressman, Roger S. *INGENIERÍA DEL SOFTWARE, UN ENFOQUE PRÁCTICO.* Madrid: Mc Graw-Hill, 2006.

Rey, Susana Eisbel, y Silvia Lanza Castelli. APLICACIÓN DE SOFTWARE COLABORATIVOS O GROUPWARE EMPLEANDO METODOLOGÍA SCRUM EN SISTEMAS Y ORGANIZACIONES. *XV Workshop de investigadores en ciencias de la computación.* Paraná - Entre Ríos, 2013. 505-509.

Roughley, Ian. *STARTING STRUTS2.* Toronto: C4Media, 2007.

Royce, W. W. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS: CONCEPTS AND TECHNIQUES. Editado por IEEE Computer Society Press. *ICSE '87 Proceedings of the 9th international conference on Software Engineering*, 1987: 328-338.

Sáez Martínez, Pedro Javier, Vicente Rodríguez Montequín, Joaquín Villanueva Balsera, y Marcos Cueto Cuiñas. SELECTION OF AGILE MODELS AND METHODOLOGIES FOR SOFTWARE PROJECTS. *18th International Congress on Project Management and Engineering* . Alcañiz, 2014. 1862-1863.

Selic, Bran. THE PRAGMATICS OF MODEL-DRIVEN DEVELOPMENT. *IEEE Software* (IEEE Computer Society) 20, nº 5 (Octubre 2003): 19-25.

Shari, Lawrence Pfleeger. *INGENIERÍA DE SOFTWARE TEORÍA Y PRÁCTICA.* Argentina: Prentice Hall, 2002.

Sharma, Sheetal, Darothi Sarkar, y Divya Gupta. AGILE PROCESSES AND METHODOLOGIES: A CONCEPTUAL STUDY. *International Journal on Computer Science and Engineering* 4, nº 5 (Mayo 2012): 892-898.

Sommerville, Ian. *INGENIERÍA DE SOFTWARE.* Madrid: Pearson Addison Wesley, 2005.

Sutherland, Jeff, Anton Viktorov, Jack Blount, y Nikolai Puntikov. DISTRIBUTED SCRUM: AGILE PROJECT MANAGEMENT WITH OUTSOURCED DEVELOPMENT TEAMS. Editado por IEEE. *40th Annual Hawaii International Conference on System Sciences.* Hawaii: IEEE, 2007. 274a-274a.

Syed-Abdullah, Sharifah, Mike Holcombe, y Marian Gheorge. THE IMPACT OF AN AGILE METHODOLOGY ON THE WELL BEING OF DEVELOPMENT TEAMS. Editado por Bill Curtis. *Empir Software Eng* (Springer Science + Business Media) 11 (2006): 143–167.

Theunissen, W.H. Morkel, Derrick G. Kourie, y Bruce W. Watson. STANDARDS AND AGILE SOFTWARE DEVELOPMENT. *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology.* South African Institute for Computer Scientists and Information Technologists, 2003. 178-188.

Völter, Markus, y Iris Groher. PRODUCT LINE IMPLEMENTATION USING ASPECT-ORIENTED AND MODEL-DRIVEN SOFTWARE DEVELOPMENT. *Software Product Line Conference, 2007. SPLC 2007. 11th International*, 2007: 233-242.

Wallin, Christina, Fredrik Ekdahl, y Stig Larsson. INTEGRATING BUSINESS AND SOFTWARE DEVELOPMENT MODELS. *IEEE Software* (IEEE Computer Society) 19 (Noviembre-Diciembre 2002): 28-33.

Walls, Craig. *SPRING IN ACTION.* EUA: Manning, 2001.

Williams, Laurie, William Krebs, Lucas Layman, Annie I. Antón, y Pekka Abrahamsson. TOWARD A FRAMEWORK FOR EVALUATING EXTREME PROGRAMMING. *Empirical Assessment in Software Eng.(EASE)*, 2004: 11-20

Cuando despertó, el dinosaurio todavía estaba allí.

Augusto Monterroso